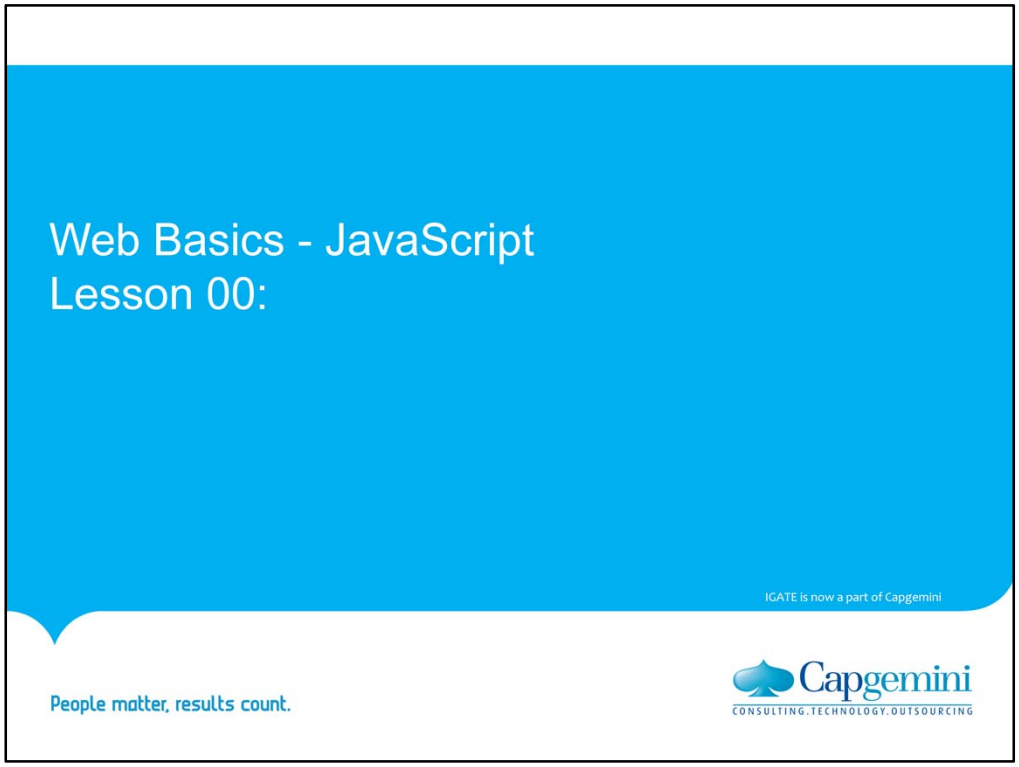


# Document History

| Date | Course Version No. | Software Version No. | Developer / SME | Change Record Remarks |
|---|---|---|---|---|
| 19-Sep-2009 | 2.0 | NA | Pradnya Jagtap | Updated to new template. And incorporated new examples. |
| Apr-2011 | 3.0 | NA | Anu Mitra | Integration updates |
| Apr-2015 | 3.1 | NA | Rathnajothi P | Revamped according to revised curriculum |
| May-2016 | 3.2 | NA | Kavita Arora | Revamped according to revised curriculum |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

# Course Goals and Non Goals

- Course Goals:
  - At the end of this course you will be able to:
  - Add interactivity to the static HTML pages
  - Validate the input data provided by users, on the client side
  - Manipulate style sheets on the fly to give a sophisticated look to any website
- Course Non Goals:
  - Server-side scripting not covered.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

# Pre-requisites

- Prerequisites for this course are:
  - Familiarity with Windows, GUI Concept and Web Browser Application
  - HTML
  - Experience of creation of a web page using HTML
  - Object Oriented Programming Concepts

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4

# Intended Audience

- This course is designed for:
  - Trainee Programmers
  - Software Professionals

# Day Wise Schedule

- Day 1
  - Lesson 1: Introduction to JavaScript
  - Lesson 2: JavaScript Language
  - Lesson 3: Working with Predefined Core Objects
  - Lesson 4: Working with Arrays
  - Lesson 5: Document Object Model
- Day 2
  - Lesson 6: Working with Document object
  - Lesson 7: Working with Form object
  - Lesson 8: Working with Regular Expressions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

# Table of Contents



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

# Table of Contents



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

# Table of Contents



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

# References

- JavaScript: A Beginner's Guide - by John Pollock
- http://www.w3schools.com

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

## Next Step Courses (if applicable)

- Servlet
- JSP

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

## Other Parallel Technology Areas

- VBScript

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

# Web basics-JavaScript

Lesson 1: Introduction to JavaScript

# Lesson Objectives

- To understand the following topics:
- Basic Concepts of JavaScript
- Embedding JavaScript in HTML
- Writing JavaScript

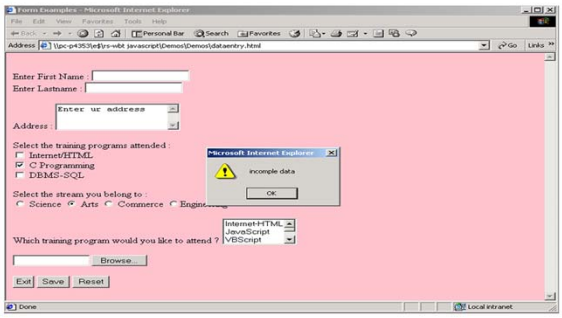
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

1.1: Basic Concepts of JavaScript

## Basic Concepts of JavaScript

- JavaScript is the scripting language of the Web
- JavaScript is used for placing dynamic content into HTML page and for client side validation which cannot be performed with HTML 5 elements and attributes




Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

Basic Concepts of JavaScript

JavaScript History :

Web pages made using only HTML are somewhat static with no interactivity and negligible user involvement.

HTML tags are just instructions on document and the display of the document is dependent on the browser.

Interactive pages cannot be built with only HTML, we need a programming language.

So Netscape came out with a client-side language called as JavaScript.

What is JavaScript?

JavaScript is THE scripting language of the Web.

JavaScript is used in Web pages to add functionality, validate forms, detect browsers, and much more.

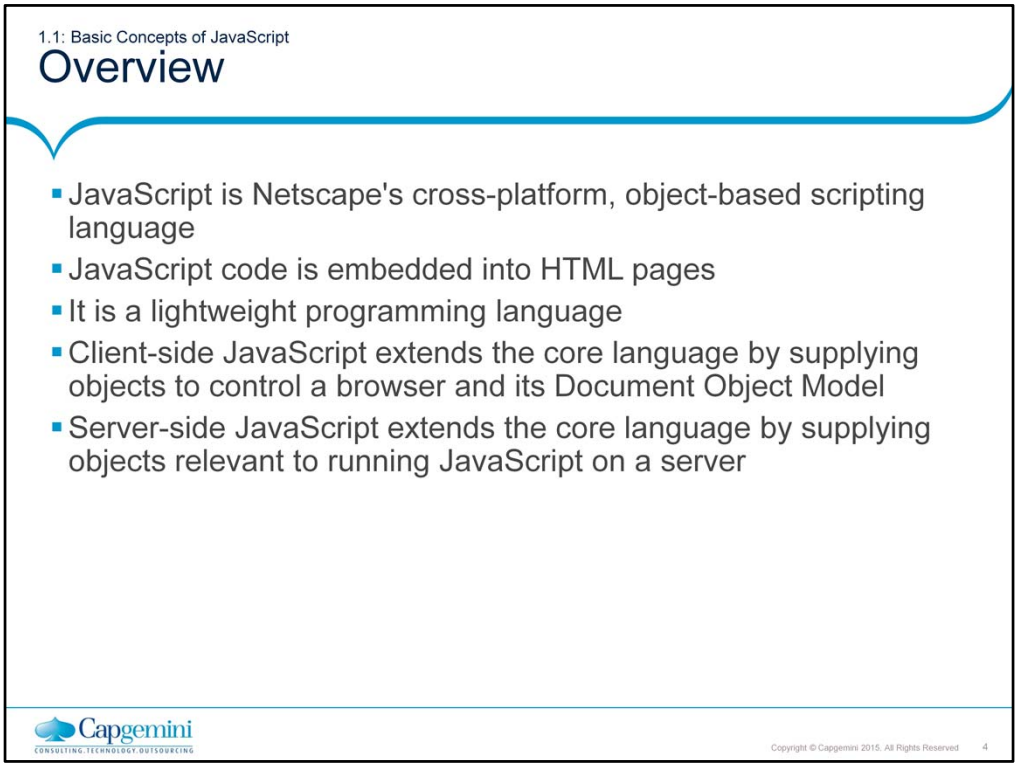
JavaScript is the most popular scripting language on the internet and works on all major browsers available like IE, Firefox, Chrome etc..

Need of JavaScript:

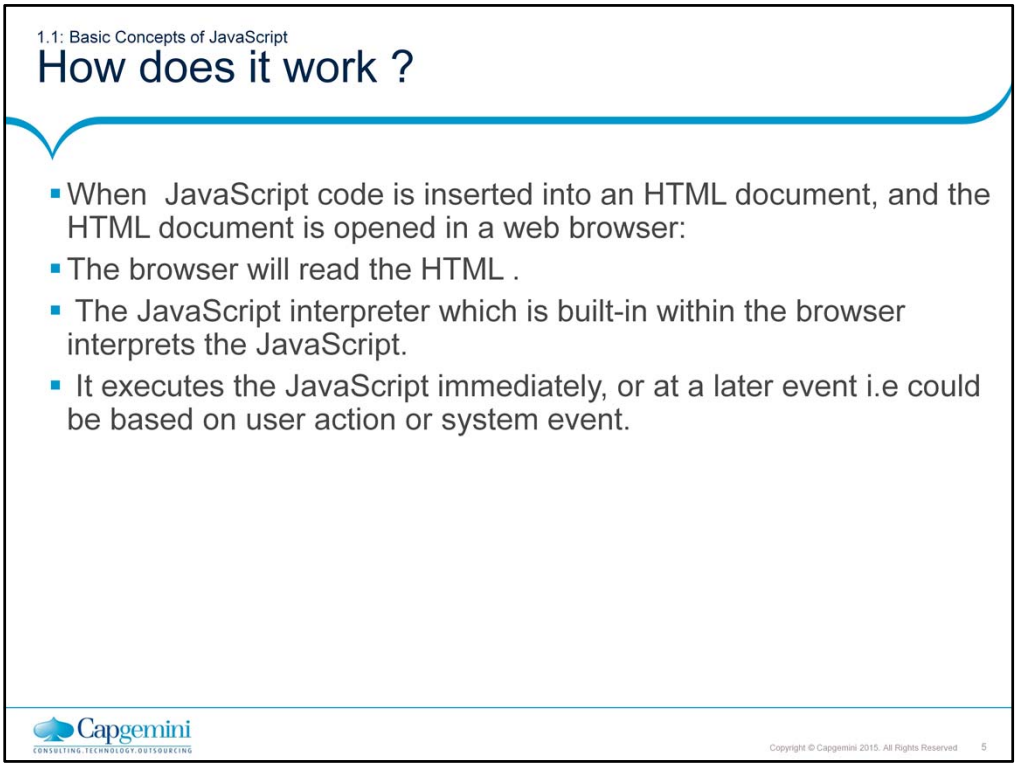
For placing dynamic content into an HTML Page.

For client side Validation.

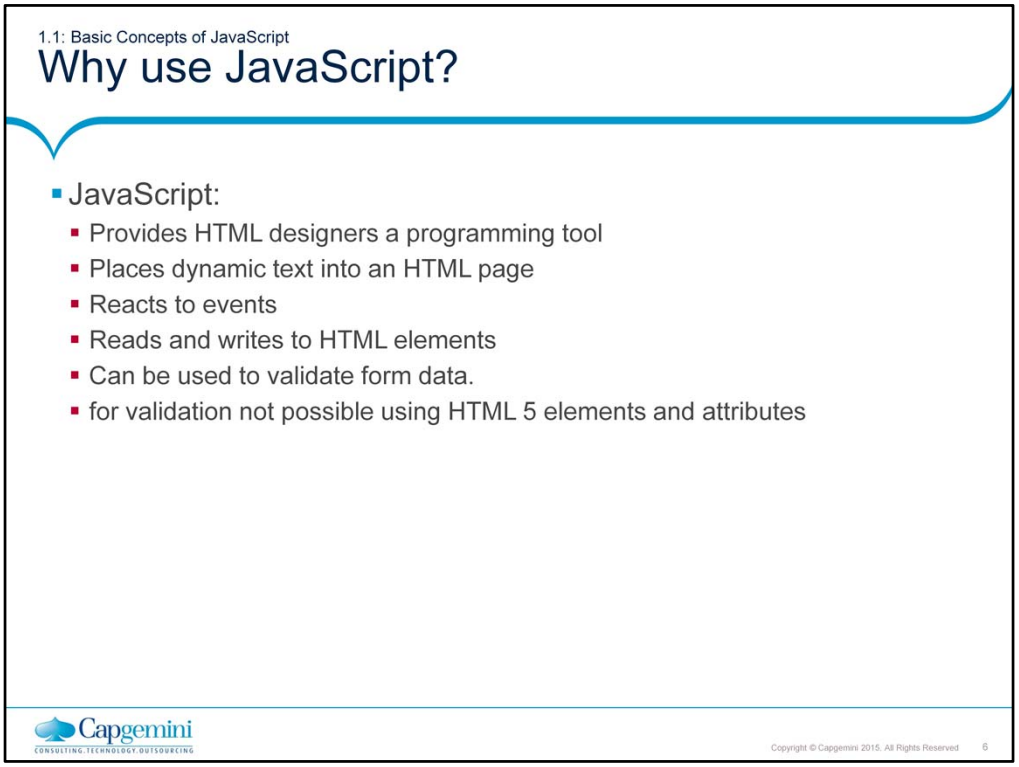
For storing and retrieving client's information in the form of Cookies.




What is JavaScript?
JavaScript is Netscape's cross-platform, object-based scripting language. Core JavaScript contains a core set of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements.
Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example:
Client-side JavaScript extends the core language by supplying objects to control a browser (Navigator or another web browser) and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
Server-side JavaScript extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a relational database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.
JavaScript lets you create applications that run over the Internet. Client applications run in a browser, such as Internet Explorer/Firefox, and server applications run on a server, such as Netscape Enterprise Server. Using JavaScript, you can create dynamic HTML pages that process user input and maintain persistent data using special objects, files, and relational databases.




How does JavaScript work?
When a JavaScript is inserted into an HTML document, the JavaScript interpreter which is built-in within the Internet browser will read the HTML and interpret the JavaScript. The JavaScript can be executed immediately, or at a later event.




Why use JavaScript?

JavaScript gives HTML designers a programming tool :

JavaScript is a simple scripting language which can be used by HTML authors who essentially are not familiar with programming. Hence the HTML authors can easily put small JavaScript code snippets into HTML pages.

JavaScript can place dynamic text into an HTML page : A simple HTML text which displays static content such as <h1>Welcome</h1> can be written in JavaScript to display dynamic content using document.write("<h1>" + orgname + "</h1>")

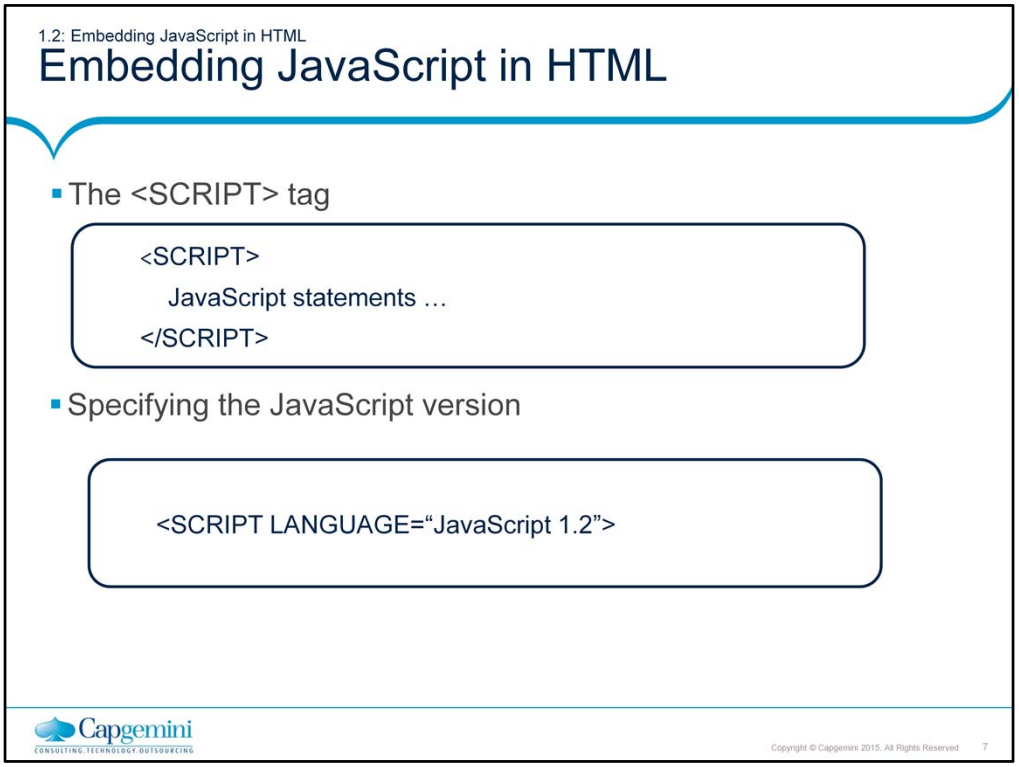
JavaScript can react to events : A JavaScript can be set to execute when some action takes place, like when a page has finished loading or when a user clicks on an HTML element.

JavaScript can reads and writes HTML elements: A JavaScript can read values of HTML elements and also write/change the content of an HTML element.

JavaScript can be used to validate data : Client side validation can easily taken care of by JavaScript. This reduces the burden on the server.

With the advent of HTML 5, most of the form data validation can be easily performed using HTML 5 elements like input type=email and attributes like required, pattern, maxlength etc

Apart from this JavaScript can also be used to create cookies which is stores and retrieves information about the user preferences. JavaScript can also be used to detect browser which helps in loading a page specifically designed for the browser.




Embedding JavaScript in HTML:
The <SCRIPT> tag is an extension to HTML that can enclose any number of JavaScript statements as shown on the slide.
A document can have multiple <SCRIPT> tags, and each can enclose any number of JavaScript statements.
The Script tag has the following attributes:
Language – This attribute specifies the scripting language. It can have values like VBScript, JavaScript. Optionally you can also specify the scripting language version as mentioned on the slide.
Type – Specifies the MIME type of the scripting language
Src – This attribute is for specifying the path and filename of an external .js file which contains the script code.
Some browsers may not support JavaScript and the JavaScript code is displayed as page content. To prevent this, the HTML comment should be used which will hide the JavaScript as shown on the slide. Note the two forward slashes at the end of the comment line. This is JavaScript comment symbol which prevents JavaScript from executing the -- > tag.
Note that Javascript is a case-sensitive scripting language.

1.2: Embedding JavaScript in HTML

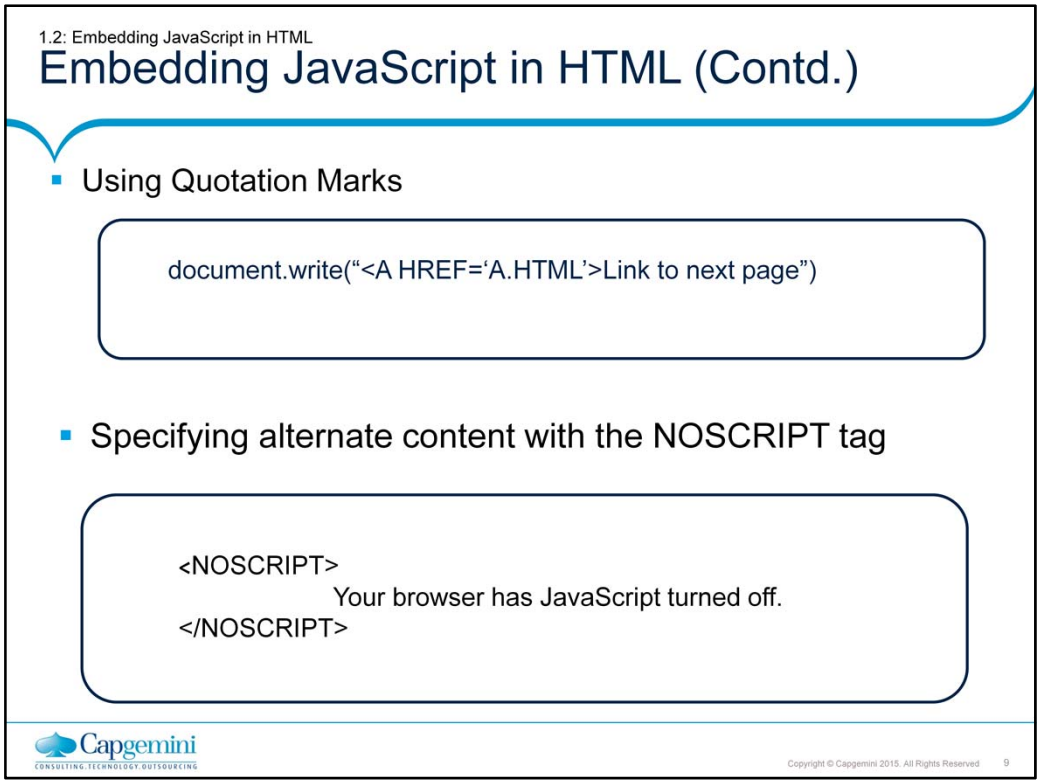
## Embedding JavaScript in HTML(contd)

- Hiding Scripts with Comment tags

```
<script type="text/javascript">
    <!--
            some statements …
    // -->
</script>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8




**Embedding JavaScript in HTML (contd..):**

Whenever you want to indicate a quoted string inside a string literal, use single quotation marks (') to delimit the string literal. This allows the script to distinguish the literal inside the string. In the following example,

The attribute values are in double quotes, but in the call to the function myfunc the argument passed is a string which is enclosed in a single quotes.

```
<INPUT TYPE="button" VALUE="Press Me" onClick="myfunc('astring')">.
```

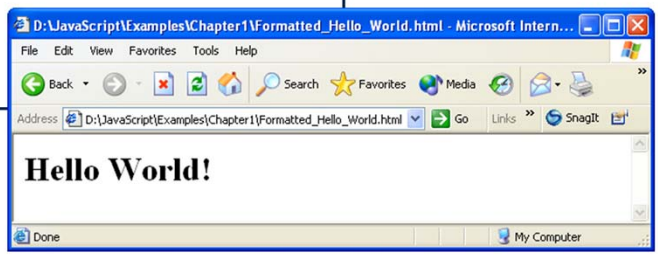
Use the <NOSCRIPT> tag to specify alternate content for browsers that do not support JavaScript. HTML enclosed within a <NOSCRIPT> tag is displayed by browsers that do not support JavaScript; code within the tag is ignored by browser. In case, if the user has disabled JavaScript from the Advanced tab of the Preferences dialog, the browser displays the code within the <NOSCRIPT> tag.

1.2: Embedding JavaScript in HTML

## Embedding JavaScript in HTML (Contd.)

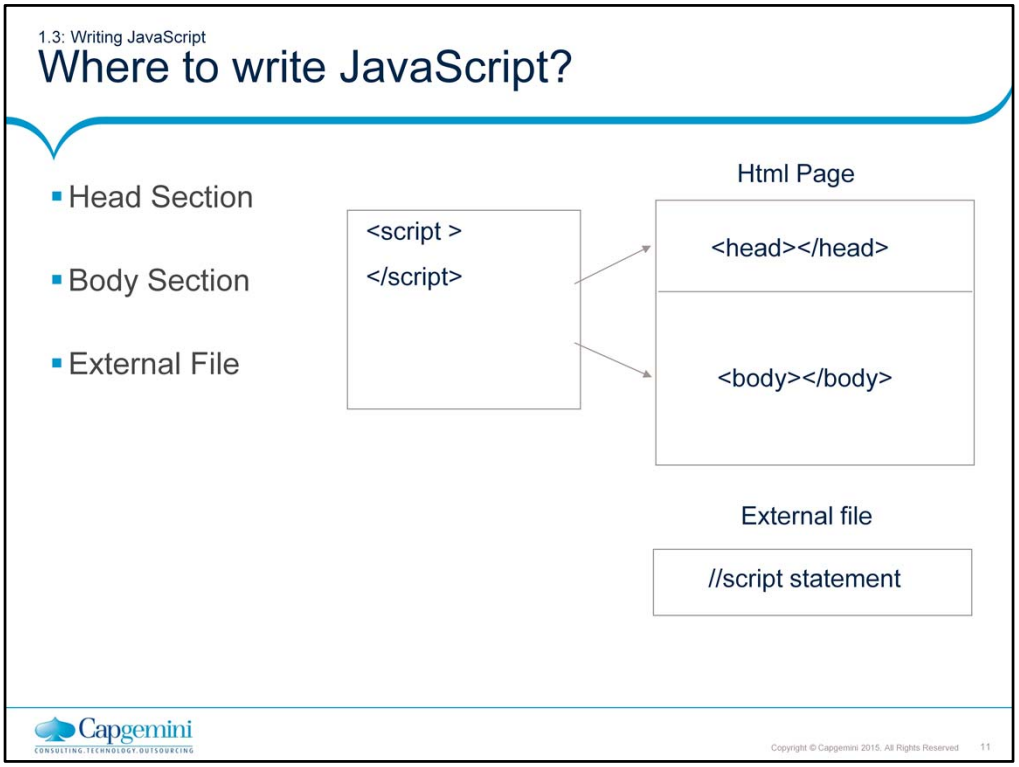
- Including text-formatting features

```
<!DOCTYPE html>
<html>
<head> </head>
<body>
<script>
document.write("<H1>Hello World!</H1>")
</script>
</body>
</html>
```




Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING




Embedding JavaScript in HTML: Where to Write JavaScript?
Physically the script code can be placed at three different places in the html file.
Head Section: If you want the Scripts to be executed when they are called, or whenever a user action happens, put script tag in the head section.
Body Section : If you want the Scripts to be executed when the page loads then put script tag in the body section
External File : If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file.

1.3: Writing JavaScript

## JavaScript in Head Section

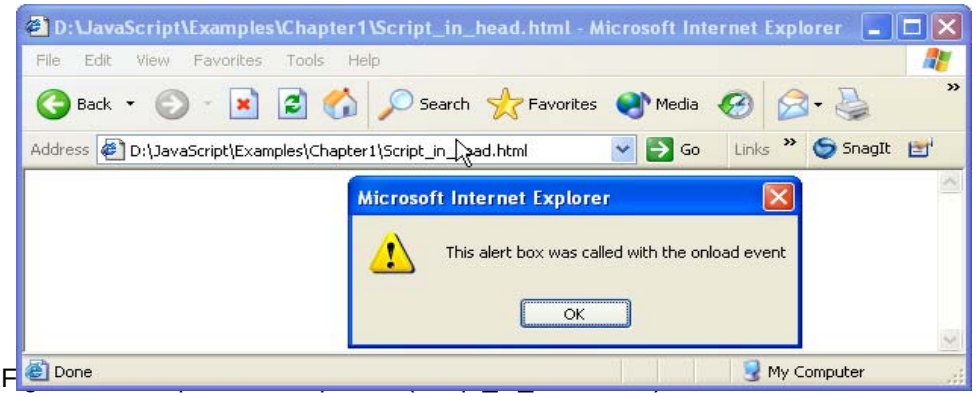
- Syntax

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function message()
      {
        alert("This alert box was called with the
              onload event")
       }
</script>
</head>
<body onload="message()">
</body>
</html>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

Where to Write JavaScript? In the Head Section:
Head Section : JavaScripts in an HTML page will be executed when the page loads. This might always not be the case. Sometimes we want to execute a JavaScript when an event occurs, such as when a user clicks a button. In such scenarios, we can put the script inside a function. Scripts that contain functions go in the head section of the document. Then we can be sure that the script is loaded before the function is called.
The example shown on the slide shows the following output:




Fig

1.3: Writing JavaScript

## JavaScript in Body Section

- Syntax

```
<!DOCTYPE html>
<html>
<head>
<title>script tag in body</title>
</head>
<body >
<script language="javascript">
        document.write("this message is written
when the page loads")
</script>
</body>
</html>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

Where to Write JavaScript? In the Body Section:
Body Section : Execute a script that is placed in the body section. The example on the slide shows script written in the body section.
And it produces this output:

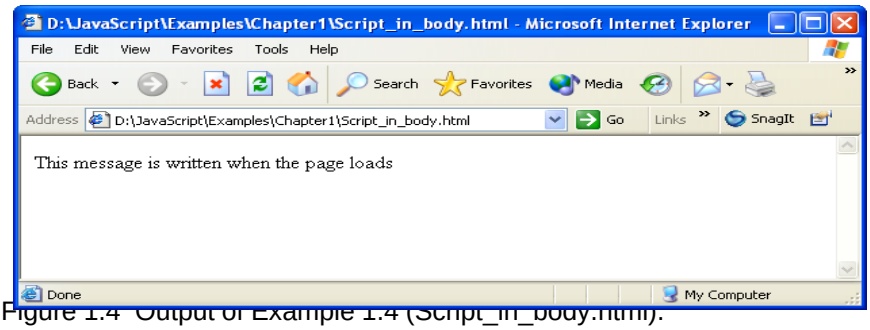
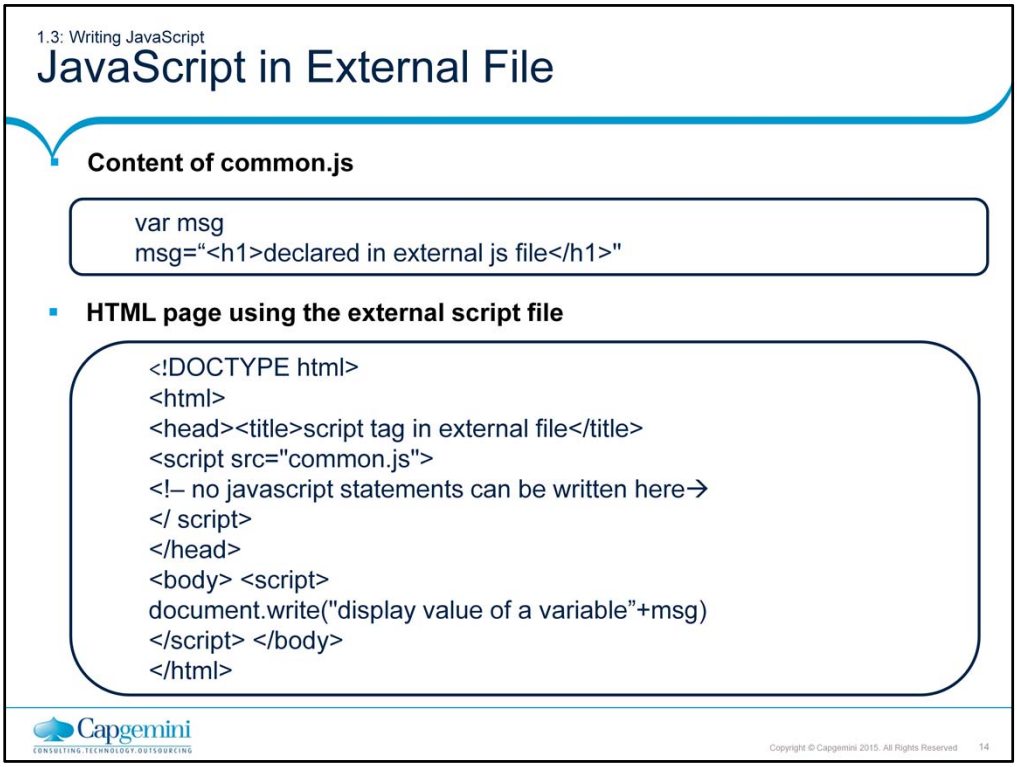



Figure 1.4 Output of Example 1.4 (Script_in_body.html).




**Where to Write JavaScript? In External File:**

The example on the slide demonstrates how to write JavaScript code in an external file. The extension of the external file has to be .js and the it does not contain the script tag.

# Demo

- Hello.html
- Head_section.html
- Extern_file.html
- Comm.js
- Var_ex.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

# Lab

- Lab 1:
  - Basic concepts of JavaScript

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

# Summary

- JavaScript is the client side scripting language
- When JavaScript code is placed inside a web page, the browser loads the page & the built-in interpreter reads the script & executes the same
- JavaScript is used in Web pages for
  - Validating client side data.
  - Placing dynamic content into an HTML page.
  - Storing client's information in the form of Cookies.


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

## Review Question

- Question 1: JavaScript, a scripting language, is:
  - Option 1: Cross-platform, object-oriented
  - Option 2: Cross-platform, object-based
  - Option 3: Non cross-platform, object-oriented
- Question 2:The <SCR> tag is an extension to HTML that can enclose any number of JavaScript statements.
  - True/False
- Question 3:HTML enclosed within a _____ tag is displayed by browsers that do not support JavaScript.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

## Review Question: Match the Following

| | |
|---|---|
| 1. Embed JavaScript in an HTML document as statements and functions within this tag | 1. <NOSCRIPT> |
| 2. To specify alternate content for browsers that do not support JavaScript | 2. Head, Body |
| 3. Feature of JavaScript | 3. SCRIPT |
| 4. JavaScript can be written in these sections | 4. To validate and process user data |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

# Web basics- JavaScript

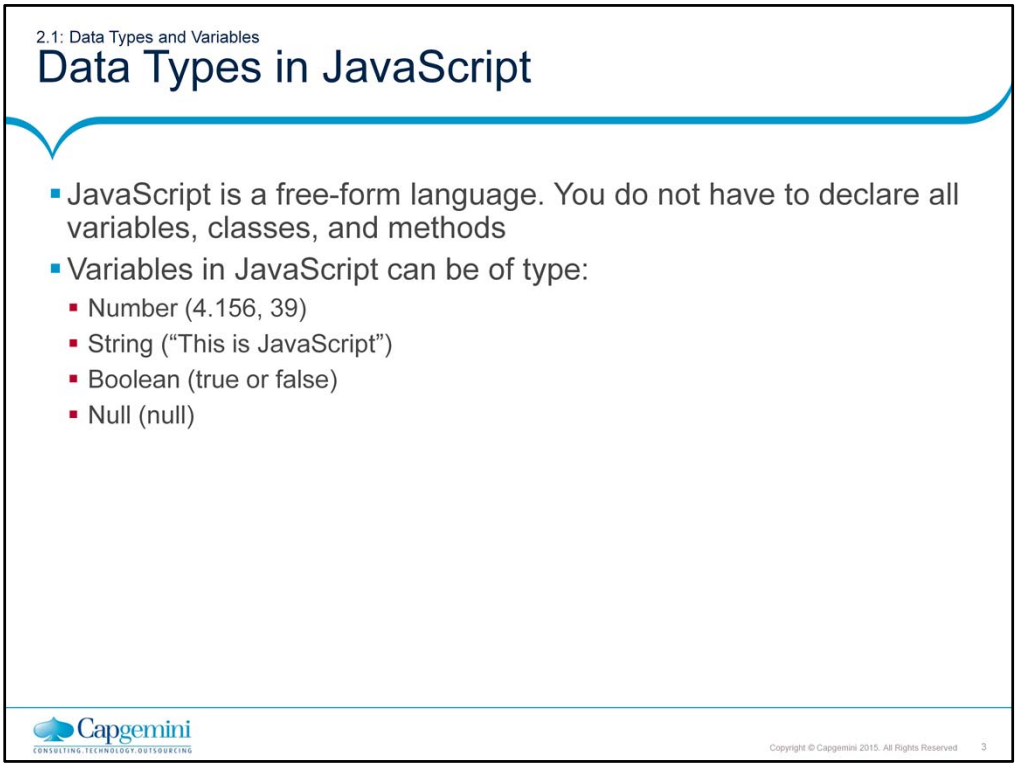
Lesson 2: JavaScript Language

# Lesson Objectives

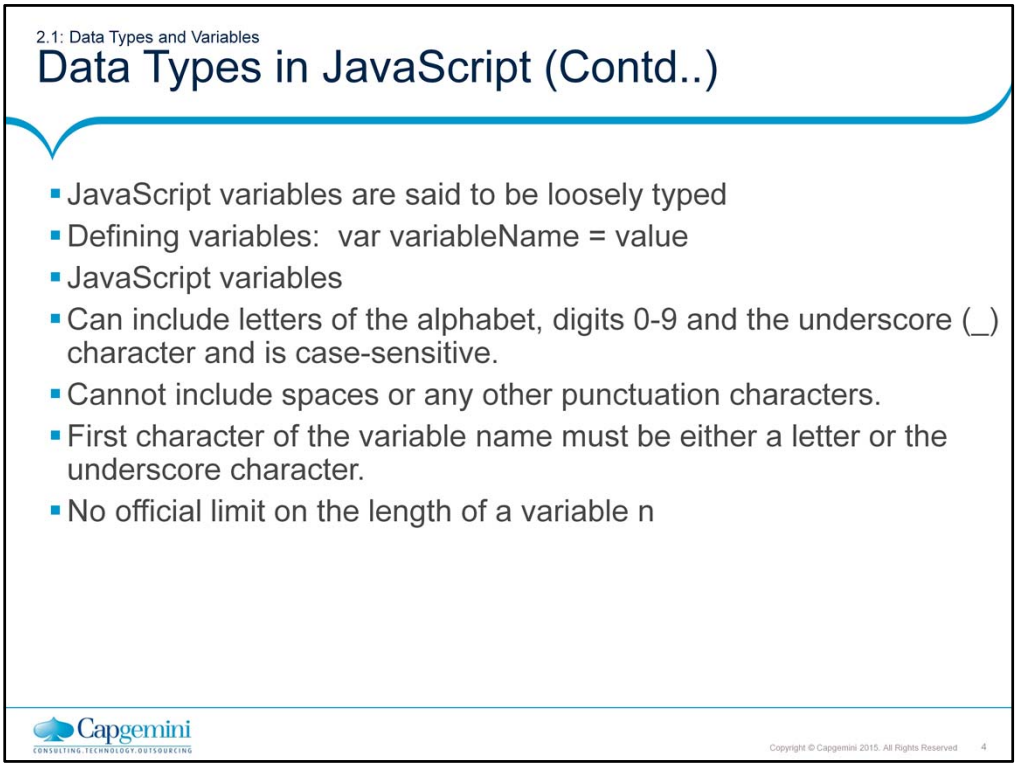
- Data Types and Variables
- JavaScript Operators
- Control Structures and Loops
- JavaScript Functions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2




Data Types in JavaScript:
Although the number of data types is small, they are sufficient for the tasks that JavaScript performs. Notice that there is no distinction between integers and real numbers; both types are just numbers. JavaScript does not provide an explicit data type for a date. However, there are related functions and a built-in date object that enable the Web page designer to manage dates.




Defining Variables:
There are specific rules you must follow when choosing variable names.

Variable names can include letters of the alphabet, both upper and lowercase. They can also include the digits 0-9 and the underscore (_) character.

Variable names cannot include spaces or any other punctuation characters.

The first character of the variable name must be either a letter or the underscore character.

Variable names are case-sensitive; totalnum, Totalnum, and TotalNum are separate variable names.

There is no official limit on the length of a variable name, but it must fit within one line.

JavaScript variables are said to be loosely typed. To declare a variable for a JavaScript program, you would write this:

```
var variablename = value ;
```

2.2: JavaScript Operators

## Arithmetic Operator

| Operator | Description | Example | Result |
|---|---|---|---|
| + | Addition | 2 + 2 | 4 |
| - | Subtraction | 5 – 2 | 3 |
| * | Multiplication | 4 * 5 | 20 |
| / | Division | 5 / 2 | 2.5 |
| % | Modulus | 10 % 8 | 2 |
| ++ | Increment | x = 5; x++ | x = 6 |
| -- | Decrement | x = 5; x-- | x = 4 |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

2.2: JavaScript Operators

## Comparison Operator

| Operator | Description | Example | Result |
|---|---|---|---|
| == | is equal to | 5 == 8 | false |
| != | is not equal | 5 != 8 | true |
| > | is greater than | 5 > 8 | false |
| < | is less than | 5 <= 8 | true |
| >= | is greater or equal | 5 >= 8 | false |
| <= | is less or equal | 5 <= 8 | true |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

2.2: JavaScript Operators

## Assignment Operator

| Operator | Example | Is same as |
|---|---|---|
| += | x += y | x = x + y |
| -= | x -= y | x = x – y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

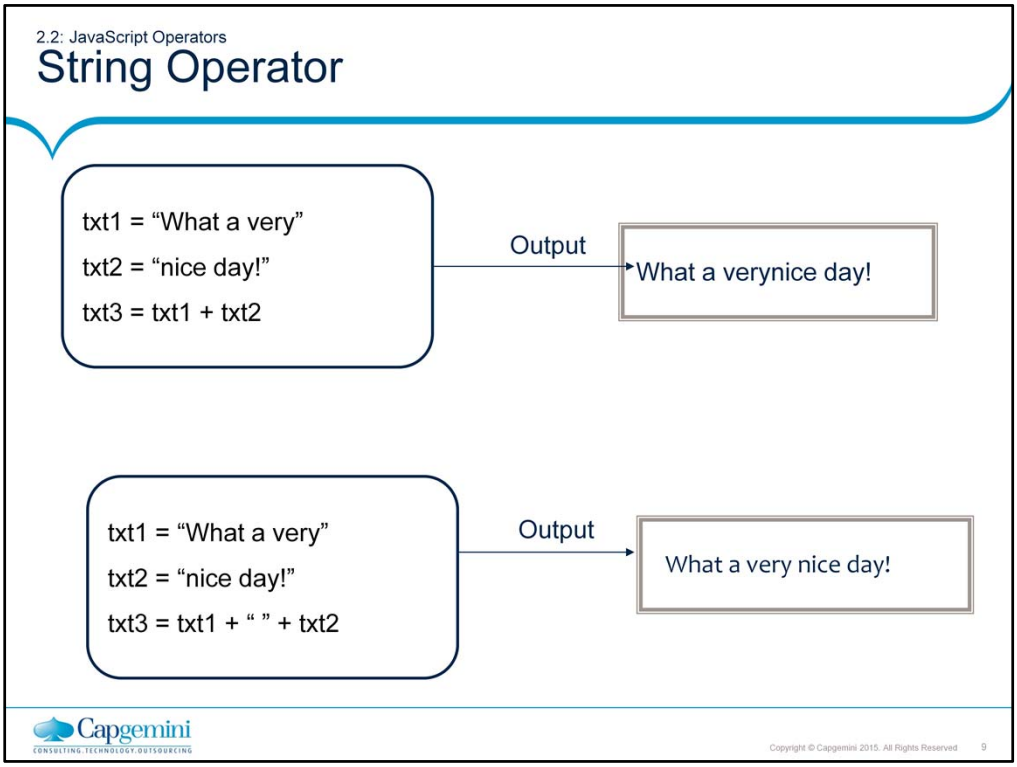
2.2: JavaScript Operators

## Logical Operator

| Operator | Description | Example |
|---|---|---|
| && | and | x = 6; y = 3<br>x < 10 && y > 1 returns true |
| \|\| | or | x = 6; y = 3<br>x < 10 \|\| y > 5 returns true |
| ! | not | x = false<br>!x returns true |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8




String Operator:

```
txt1="What a very"
txt2="nice day!"
 txt3=txt1+txt2
```

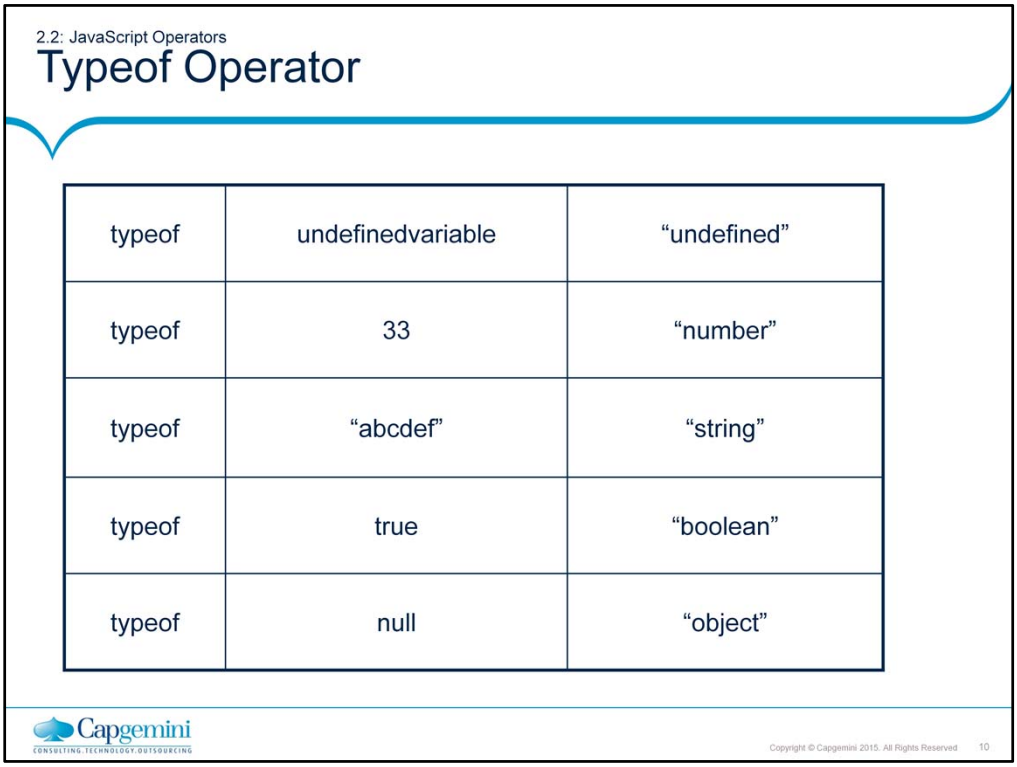
A string is most often a text, for example "Hello World!". To stick two or more string variables together, use the + operator.
The variable txt3 now contains "What a verynice day!". To add a space between two string variables, insert a space into the expression, OR in one of the strings.

```
txt1="What a very"
txt2="nice day!"
 txt3=txt1+" "+txt2
```

Or

```
txt1="What a very "
txt2="nice day!"
txt3=txt1+txt2
```

The variable txt3 now contains "What a very nice day!".




Typeof Operator:

The typeof operator returns the type of data that its operand currently holds.

```
<HTML>
<HEAD>
<TITLE>Using typeof</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--  var num1=20
  var str1="abc"
  var bool1=true
  var num2=null
  var var1;
  document.write("type of str1 : "+typeof(str1)+"<BR>")
  document.write("type of num1 : "+typeof(num1)+"<BR>")
document.write("type of bool1 : "+typeof(bool1)+"<BR>")
document.write("type of num2 : "+typeof(num2)+"<BR>") -->
</SCRIPT>
</BODY>
</HTML>
```

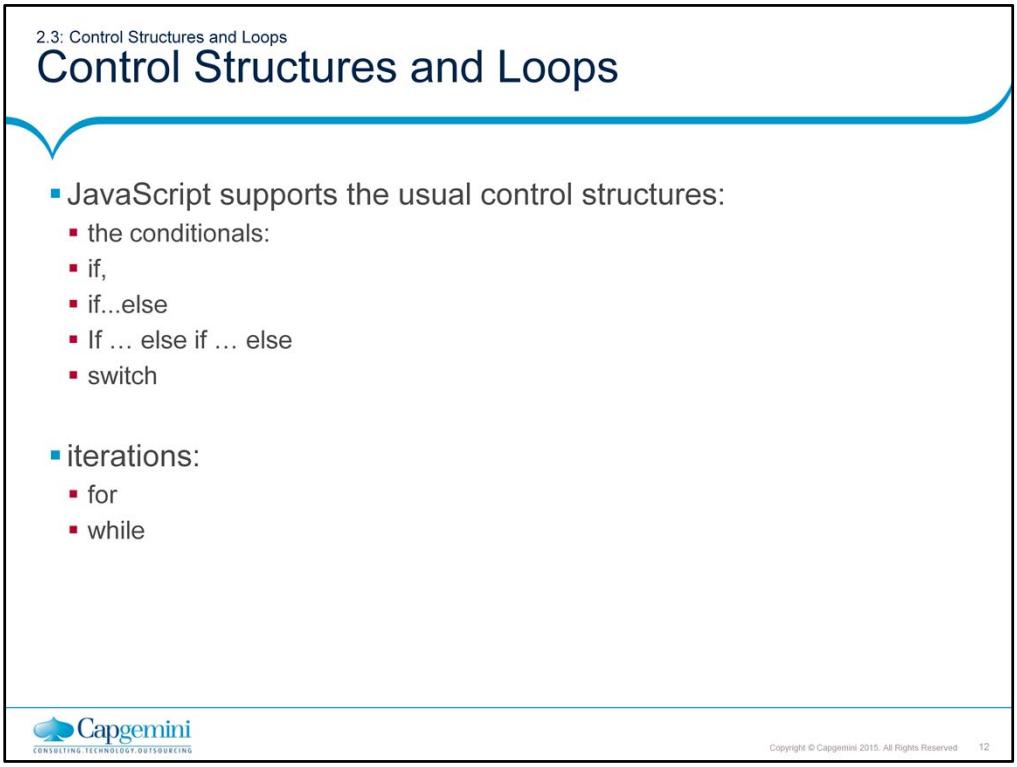
Example 2.1 typeof operator (typeof_ex.html)

## Demo

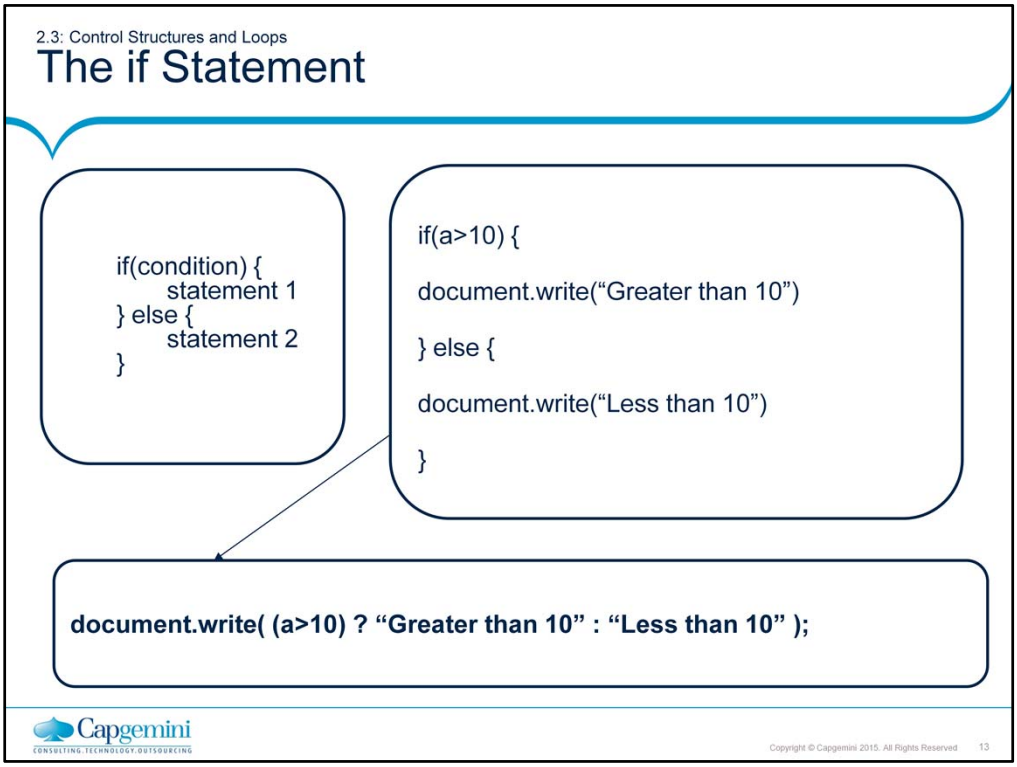
- Typeof_ex.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING




Control Structures and Loops
Conditional statements are used to perform different actions based on different conditions. Loops execute a block of code for specified number of times or while a specified condition is true.




The if Statement:
The condition is any JavaScript expression that evaluates to the Boolean type, either true or false.
The example as shown on the slide.
A shorthand method can also be used for these types of statements, where ? indicates the 'if' portion and : indicates the 'else' portion. This statement is equivalent to the previous example:
The equivalent shorthand method is also seen on the slide.

2.3: Control Structures and Loops

## The Switch Statement

- Syntax

```
switch (variable) {
  case outcome1 :{
//stmts for outcome 1
 break; }
  case outcome2 :{
 //stmts outcome 2
 break; }
  …
default: {
 //none of the outcomes
 is chosen }
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

2.3: Control Structures and Loops

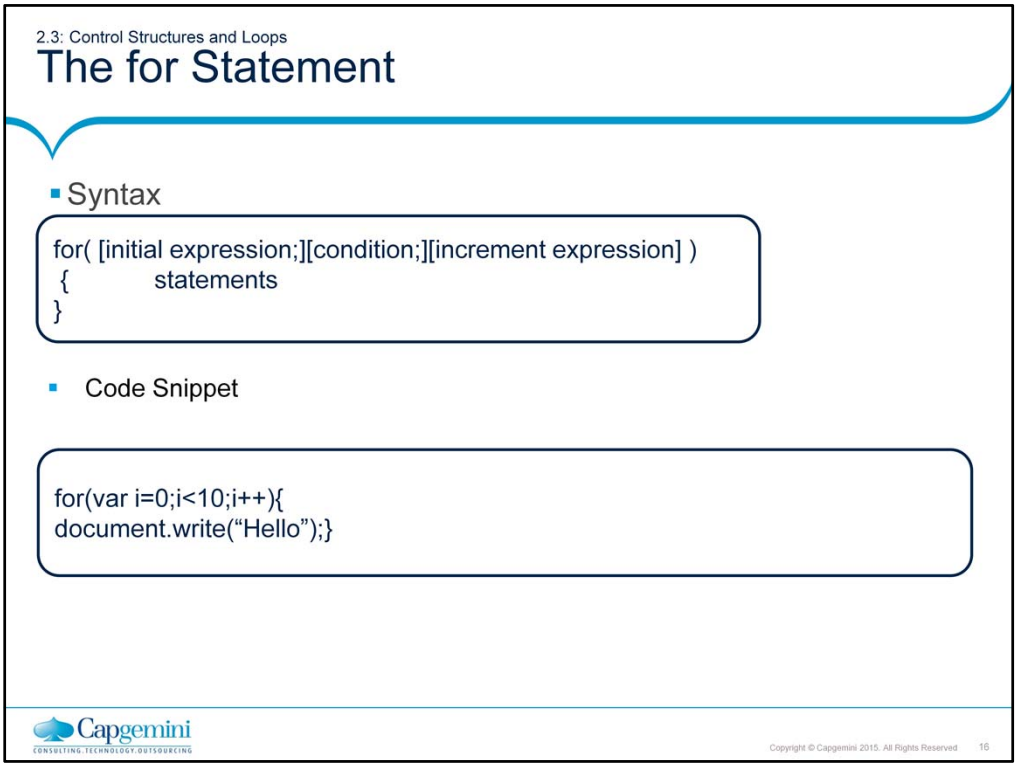
## The Switch Statement

- Code Snippet

```
switch (day) {
  case “Monday” : {
document.write(“weekday”)
break;}
 case “Saturday”: {
document.write(“weekday”)
break}
  …
 default: {
document.write(“Invalid day of the week”)
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

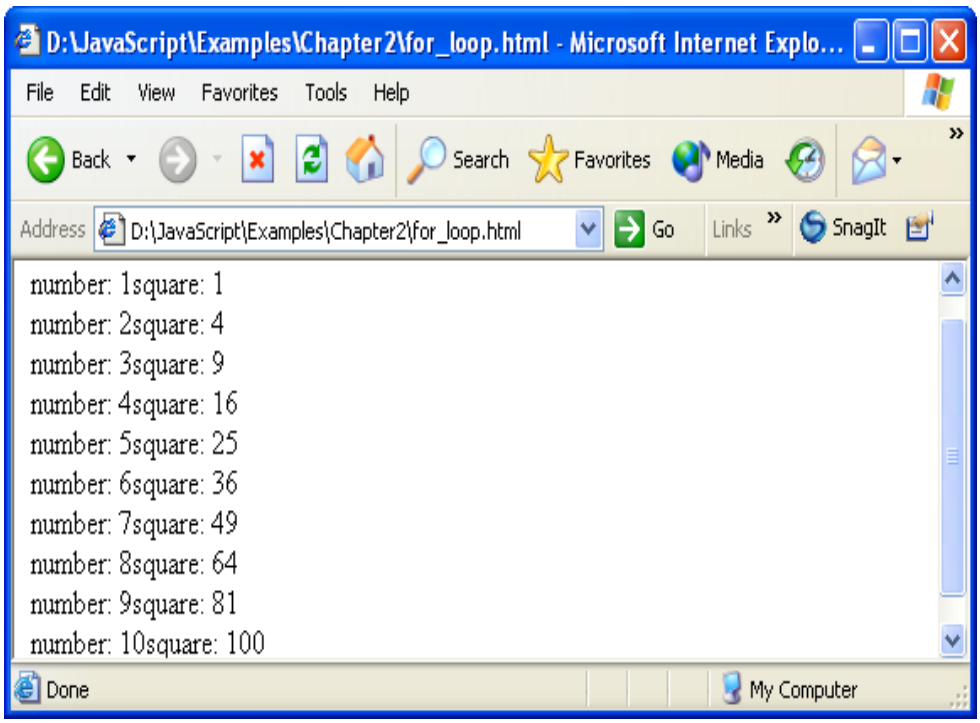
 15




Looping Statements
The 'for' Statement:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- hide script
for (i=1; i<=10; i++)
{
  sq=i*i
  document.write("number: " + i + "square: " + sq + "<BR>")
}
// end script hiding -->
</SCRIPT>
</HEAD>
<BODY></BODY></HTML>
```

Example 2.2 For Construct (for_ex.html)
And it produces the output as:




**The 'while' Statement**

The while statement continues to repeat the loop as long as the condition is true. The syntax for the while statement is as follows:

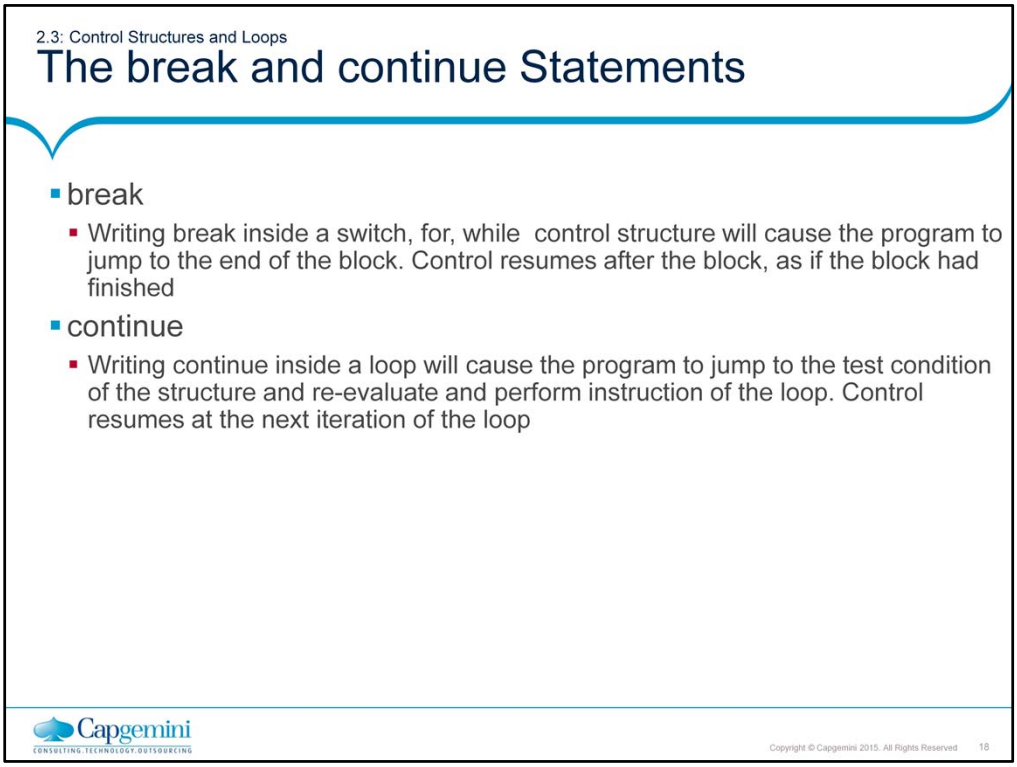
while (condition):

```
{
   statements
}
```

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- hide script
i=1
while (i<=10)
{
   sq=I*i
   document.write("number: " + i + "square: " + sq + "<BR>")
   i++
}
// end script hiding -->
</SCRIPT>
</HEAD>
<BODY></BODY>
</HTML>
```

Example 2.3 While Construct

And it produces the output as in the previous screen shot.



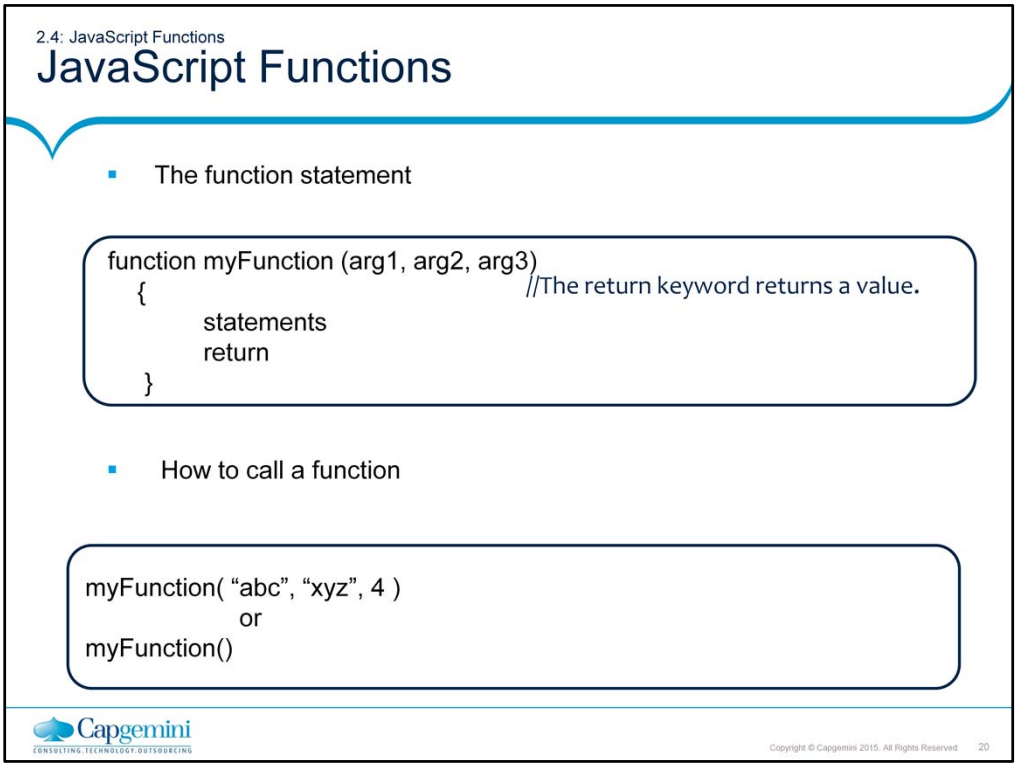

The break statement:
This statement is used to break out of the current ‘for’ or ‘while’ loop. Control resumes after the loop, as if it had finished.
The continue Statement:
This statement continues a ‘for’ or ‘while’ loop without executing the rest of the loop. Control resumes at the next iteration of the loop.

## Demo

- For_ex.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



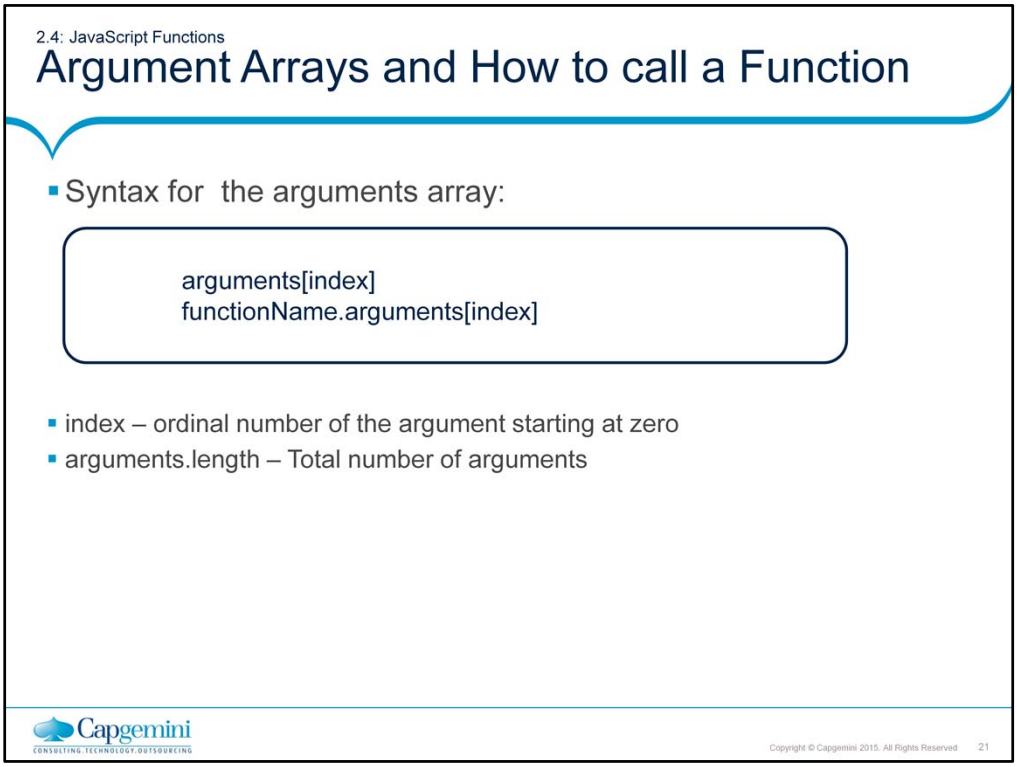

The function Statement:
A function contains some code that will be executed by an event or a call to that function. A function is a set of statements. You can reuse functions within the same script, or in other documents. You define functions at the beginning of a file (in the head section), and call them later in the document.

The syntax of a typical function is as follows:

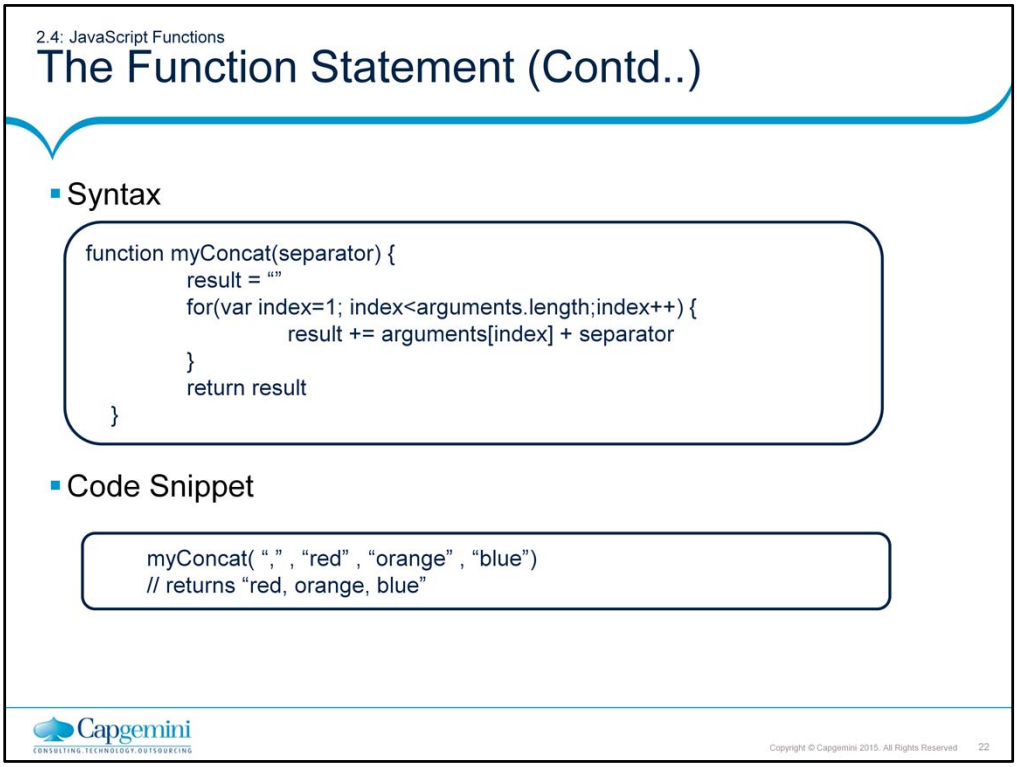
```
function myfunction(arg1, arg2, arg3)

{
  statements
}
```

How to Call a Function?
A function is not executed before it is called. You can call a function containing arguments:

```
myfunction(“abc”,”xyz”,4)
or without arguments:
myfunction()
```

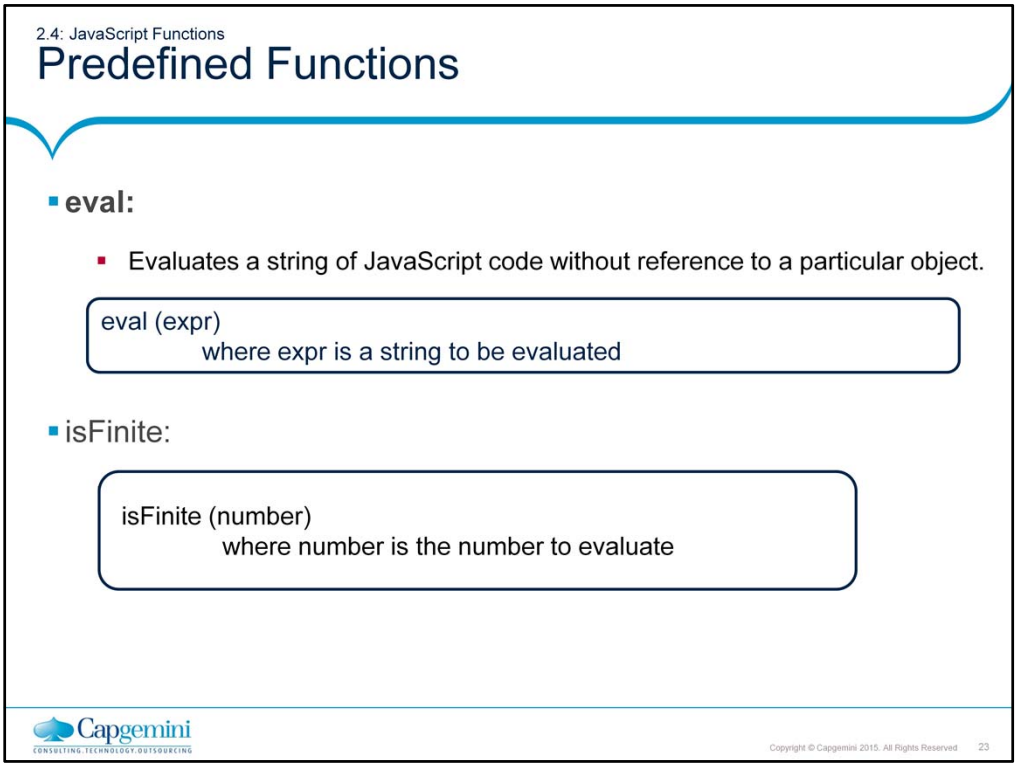



Using the arguments Array
The arguments of a function are maintained in an array. Within a function, you can address the parameters passed to it as follows:
arguments[i]
functionName.arguments[i]
where i is the ordinal number of the argument, starting at zero. So, the first argument passed to a function would be arguments[0]. The total number of arguments is indicated by arguments.length. Using the arguments array, you can call a function with more arguments than it is formally declared to accept. This is often useful if you don't know in advance how many arguments will be passed to the function. You can use arguments.length to determine the number of arguments actually passed to the function, and then treat each argument using the arguments array.




For example, consider a function that concatenates several strings. The only formal argument for the function is a string that specifies the characters that separate the items to concatenate. The function is defined as shown on the slide.
You can pass any number of arguments to this function, and it creates a list using each argument as an item in the list.

```
// returns "red, orange, blue, "
myConcat(", ","red","orange","blue“)
// returns "elephant; giraffe; lion; cheetah;"
myConcat("; ","elephant","giraffe","lion", "cheetah“)
// returns "sage. basil. oregano. pepper. parsley. "
myConcat(". ","sage","basil","oregano", "pepper", "parsley")
```

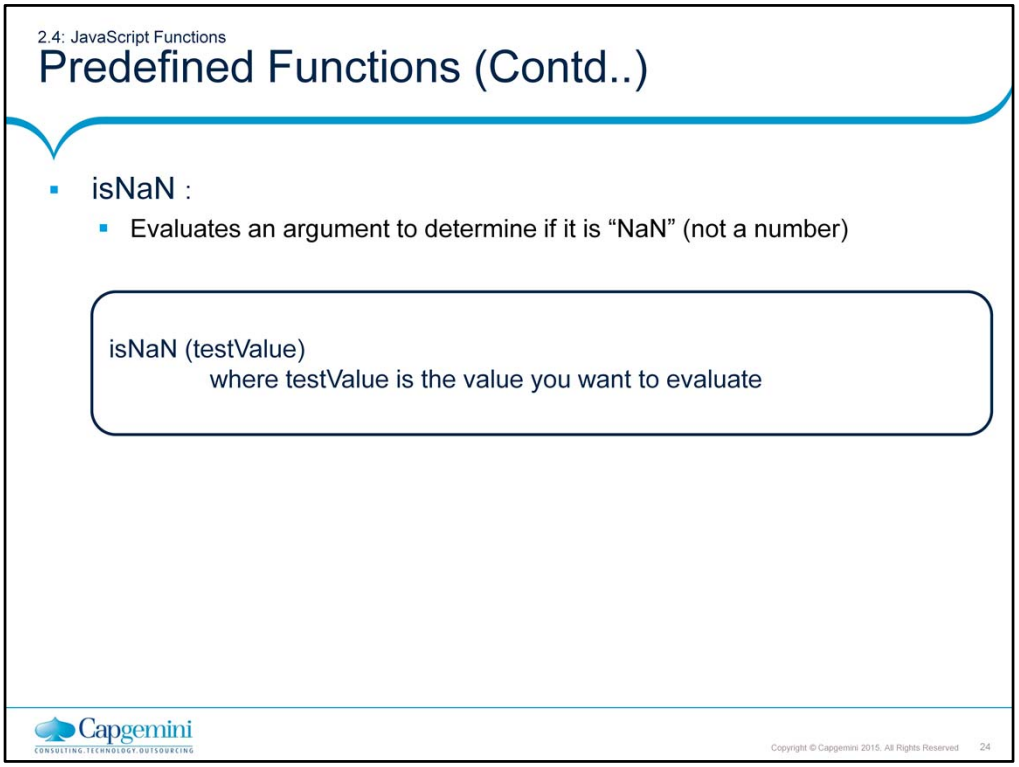



Predefined Functions:
JavaScript has several top-level predefined functions:
Eval, isFinite, isNaN, parseInt and parseFloat, Number and String

eval Function
The eval function evaluates a string of JavaScript code without reference to a particular object. The syntax of eval is:
eval(expr) where expr is a string to be evaluated.
If the string represents an expression, eval evaluates the expression. If the argument represents one or more JavaScript statements, eval performs the statements. Do not call eval to evaluate an arithmetic expression; JavaScript evaluates arithmetic expressions automatically.

isFinite Function
The isFinite function evaluates an argument to determine whether it is a finite number. The syntax of isFinite is:
isFinite(number) where number is the number to evaluate.
If the argument is NaN, positive infinity or negative infinity, this method returns false, otherwise it returns true. The following code checks client input to determine whether it is a finite number.

```
if(isFinite(ClientInput) == true)
{
   /* take specific steps */
}
```



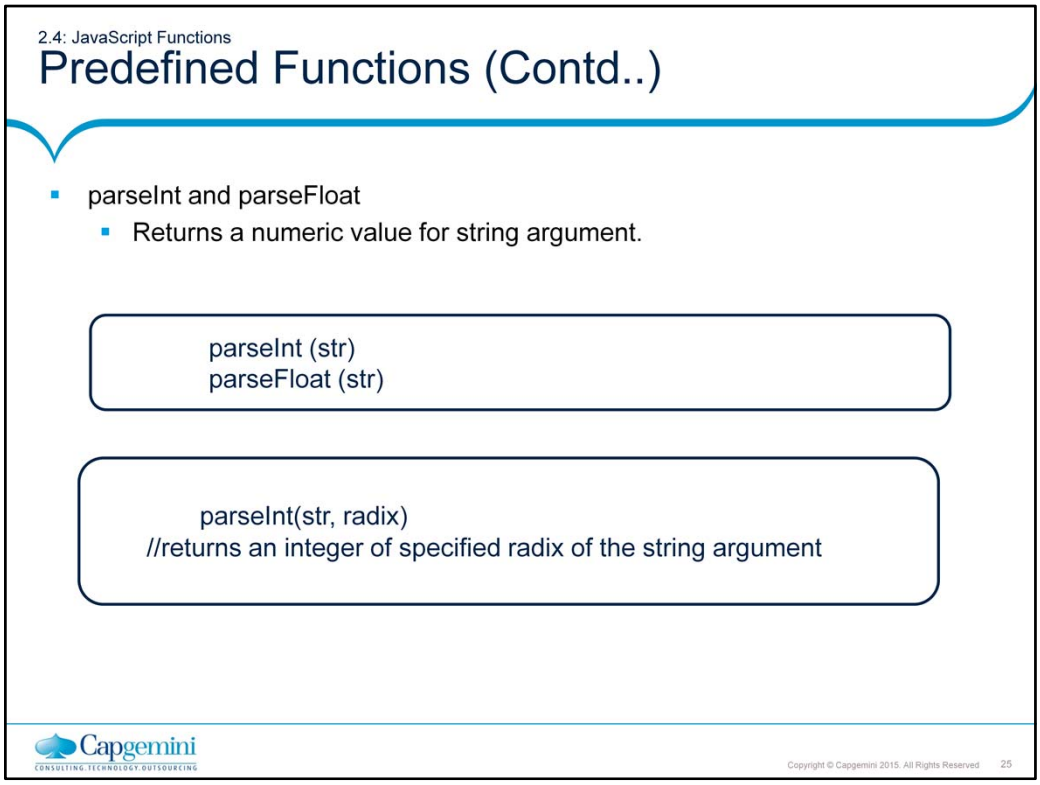

isNaN Functions:
The isNaN function evaluates an argument to determine if it is "NaN" (not a number). The syntax of isNaN is:

```
isNaN(testValue)
```

where testValue is the value you want to evaluate.
The parseFloat and parseInt functions return "NaN" when they evaluate a value that is not a number. isNaN returns true if passed "NaN," and false otherwise.

The following code evaluates floatValue to determine if it is a number and then calls a procedure accordingly:

```
floatValue=parseFloat(toFloat)
        if (isNaN(floatValue)) {
           notFloat()
        } else {
           isFloat()
        }
```




**parseInt and parseFloat Functions:**

The two "parse" functions, parseInt and parseFloat, return a numeric value when given a string as an argument.

The syntax of parseFloat is:

parseFloat(*str*)

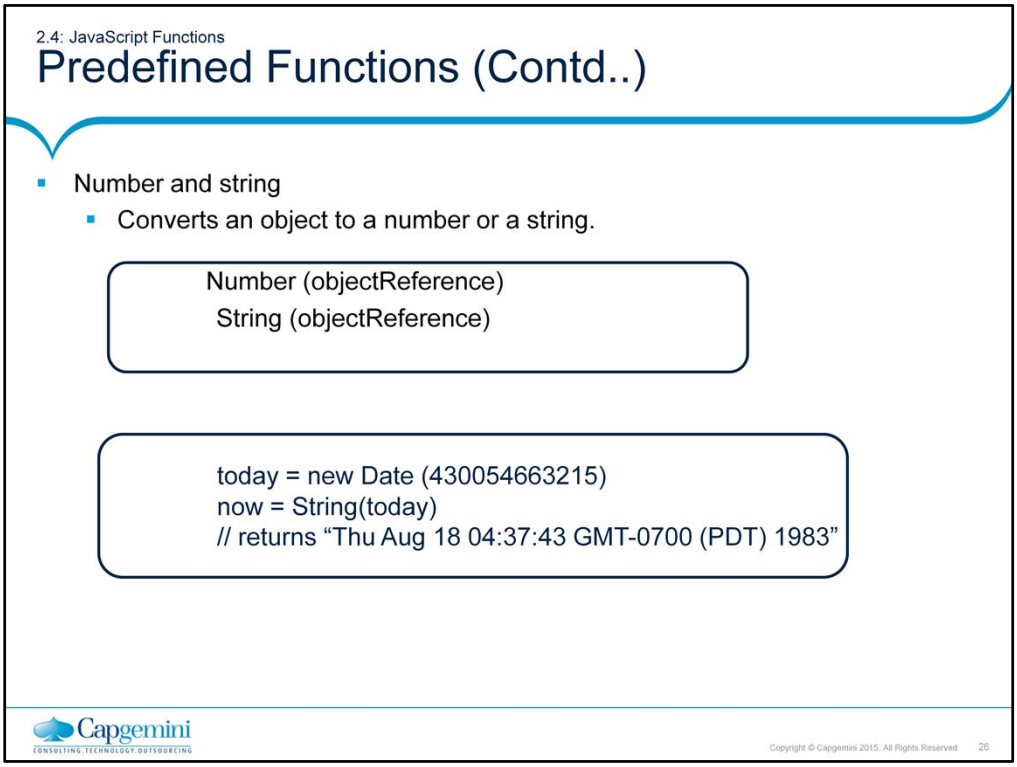
where parseFloat parses its argument, the string str, and attempts to return a floating-point number. If it encounters a character other than a sign (+ or -), a numeral (0-9), a decimal point, or an exponent, then it returns the value up to that point and ignores that character and all succeeding characters. If the first character cannot be converted to a number, it returns "NaN" (not a number).

The syntax of parseInt is:

parseInt(str [, radix])

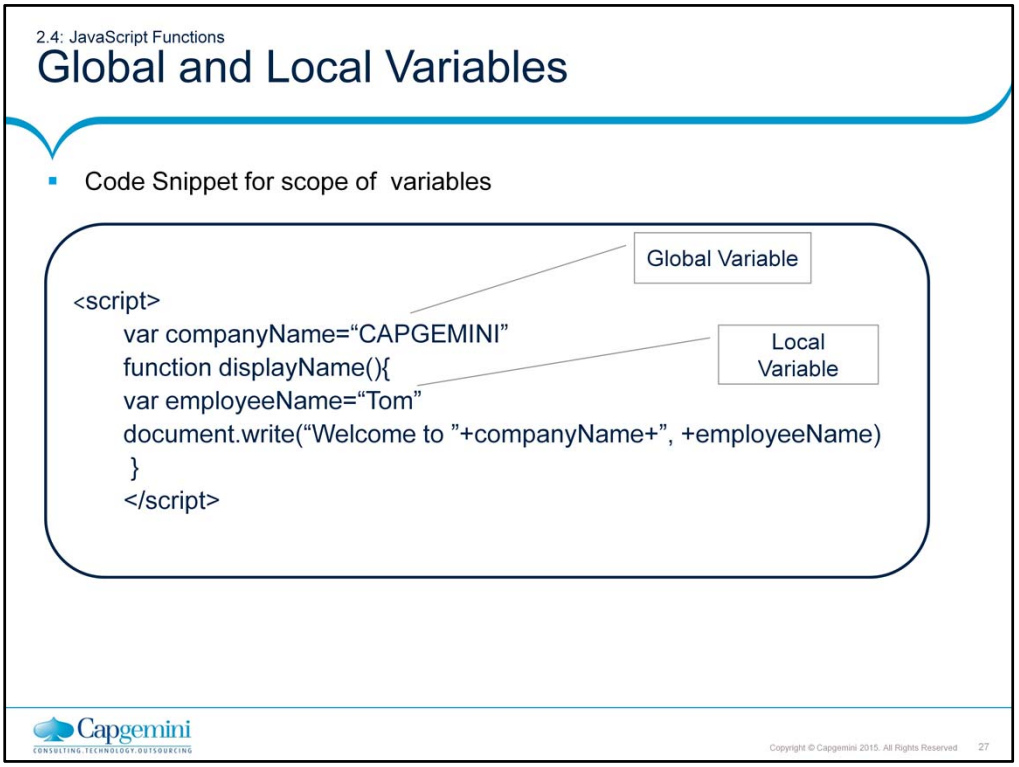
where parseInt parses its first argument, the string str, and attempts to return an integer of the specified radix (base), indicated by the second, optional argument, radix. For example, a radix of ten indicates to convert to a decimal number, eight octal, sixteen hexadecimal, and so on. For radixes above ten, the letters of the alphabet indicate numerals greater than nine. For example, for hexadecimal numbers (base 16), A through F are used.

If parseInt encounters a character that is not a numeral in the specified radix, it ignores it and all succeeding characters and returns the integer value parsed up to that point. If the first character cannot be converted to a number in the specified radix, it returns "NaN." The parseInt function truncates the string to integer values.




Number and String Functions:
The Number and String functions let you convert an object to a number or a string. The syntax of these functions is:
Number(objRef)
String(objRef)
where objRef is an object reference. The following example converts the Date object to a readable string.

```
D = new Date (430054663215)
// The following returns
// "Thu Aug 18 04:37:43 GMT-0700 (Pacific Daylight Time) 1983"
x = String(D)
```



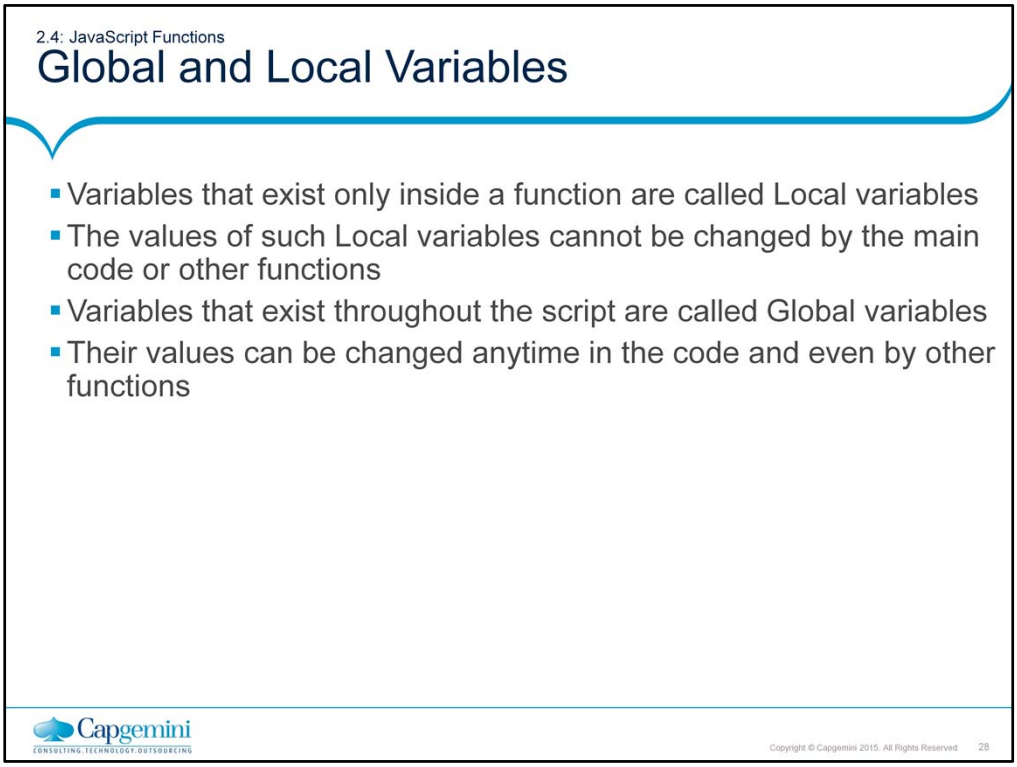

Scope of Variables:
JavaScript supports two variable scopes:

Global variables
Local variables

The local variable applies only within a function and limits the scope of the variable to that function. To declare a local variable, the variable name must be preceded by var, as shown following:

```
var MaxValue=0;
```

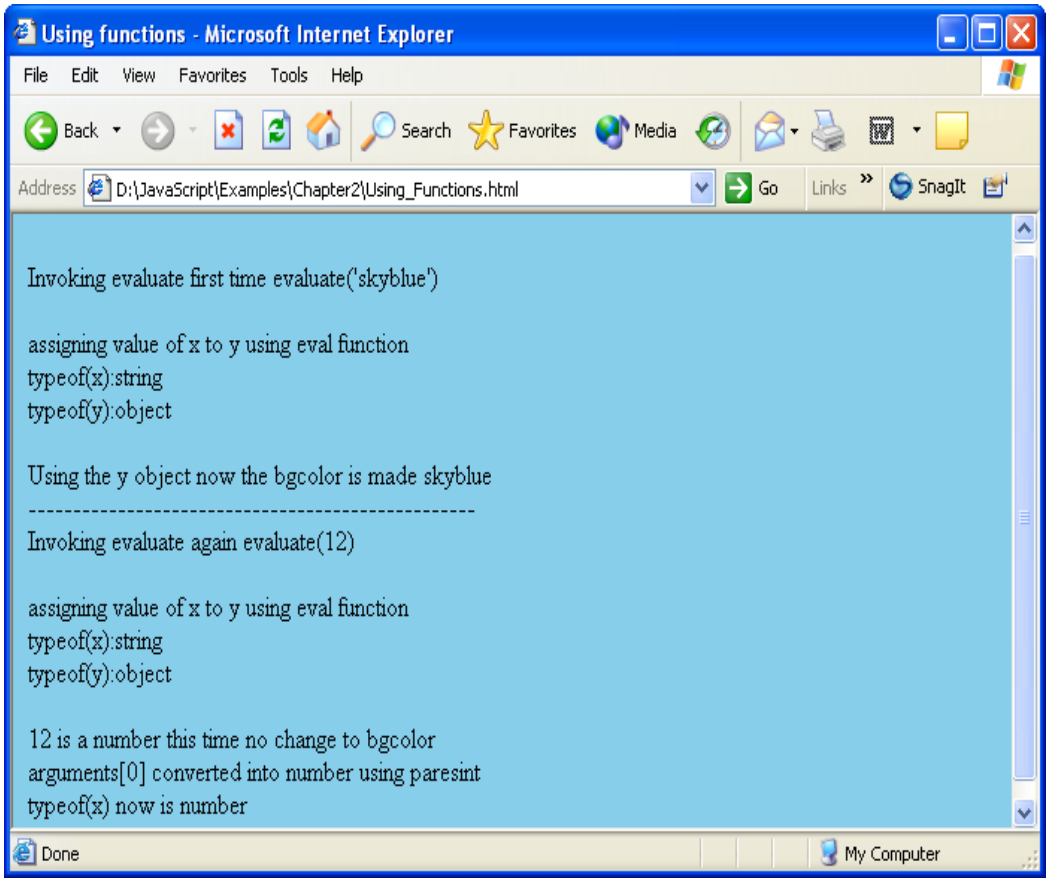
Any variable declaration that is not within a function, is treated as a global variable. The syntax to declare a global variable is the same as that for local variable.




Using Global and Local Variables:
You can choose between local and Global variables by using the following guidelines :
If the value of a variable is meant to be used by any part of the program, both inside and outside functions, the variable should be declared outside any function. This has the effect of making it global and modifiable by any part of the program. The best place to declare global variables is in the <head> block of the HTML document.
If the variable is needed only within a particular function, the variable should be declared inside that function.
If you want the value of a variable to be modified only by the main script of a single function, but you need to use it in another function, pass the variable as an argument to that function. This has the effect of making a copy of the variable and assigning its value to the argument. As the function works and modifies its own copy of the variable, it will not effect the original. Argument variables are automatically declared as local to that function. Even if the argument has the same name as the variable being passed, making changes to it does not effect the variable that was passed. The only exception to this is objects. When an object is passed as an argument, it is passed by reference as opposed to being passed by value. Instead of making a copy of the object, the function uses the original object. Changes made to an object's properties within the function have an effect on the original object.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio
6.0">
<TITLE>Using functions</TITLE>
<script language="Javascript">
<!--
   function evaluate()
    {
       var x;
        var y;
      x = "document“
      document.write("<br>assigning value of x to y using eval
function");
      y = eval(x);
      document.write("<br>typeof(x):");
      document.write(typeof(x)+"<br>");
      document.write("typeof(y):");
      document.write(typeof(y)+"<br>");
        if(isNaN(arguments[0]))
      {
          document.write("<br>Using the y object now the bgcolor is
made                              "+arguments[0] );
           y.bgColor=arguments[0] ;
      }
      else
      {
         document.write("<br>"+arguments[0]+" is a number this time
no                              change to bgcolor");
         x=parseInt(arguments[0]);
         document.write("<br> arguments[0] converted into number
using                          parseInt ");
         document.write("<br>typeof(x) now is ");
         document.write(typeof(x)+"<br>");
      }
   }
-->
</script></HEAD><BODY><script>
document.write("<br>Invoking evaluate first time
evaluate('skyblue')<br>");
evaluate("skyblue");
document.write("<BR>");
for(var i=0;i<50;i++)
document.write("-");
document.write("<br>Invoking evaluate again evaluate(12)<br>");
evaluate(12);
</script>
</BODY>
</HTML>
```

Example 2.5 Demo of creating functions and predefined function (num_string_fun.html)
And it produces the output as:



# Demo

- If_else.html
- Switch_ex.html
- For_ex.html
- Break_con_ex.html
- Fun_ex.html
- Num_string_fun.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31

# Lab

- Lab 2 :
- The JavaScript language

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 32

# Summary

- Data Types & Variables
  - Numbers, Strings, Boolean, and Null
- Operators & Expressions
- Functions
- Predefined Functions
  - eval, isFinite, isNAN, parseInt & parseFloat, Number & String
- Global and Local variables


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33

# Review Question

- Question 1: Which of the following two variable scopes is supported by JavaScript:
  - Global, Local
  - Functional, Non functional
  - Static, Dynamic

- Question 2: The eval function evaluates a string of JavaScript code without reference to a particular object.
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 34

## Review Question: Match the Following

| | |
|---|---|
| 1. Loop statements | 1. isNan |
| 2. Arithmetic operators | 2. +=, -= |
| 3. Predefined function | 3. &&,\|\| |
| 4. Assignment operators | 4. For, While |
| 5. Logical operators | 5. ++, --, %, * |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 35

# Web Basics -JavaScript

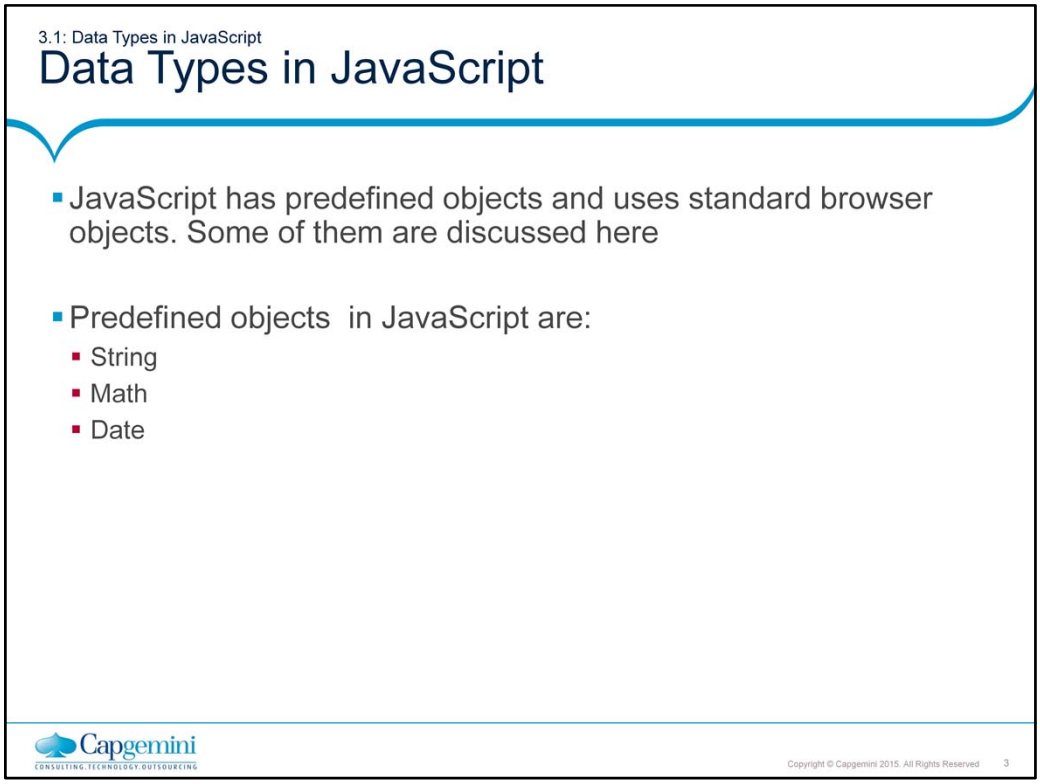
Lesson 3: Working with Predefined Core Objects

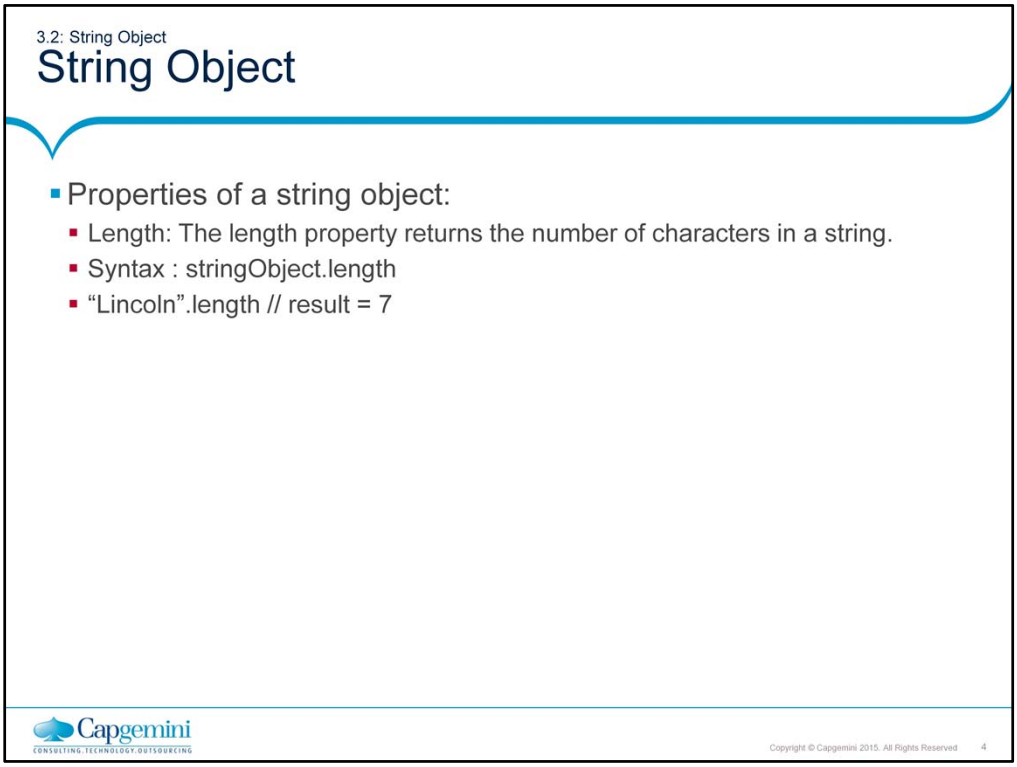
# Lesson Objectives

- Data Types in JavaScript
- String Object
- Math Object
- Date Object

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2


3.1: Data Types in JavaScript
Data Types in JavaScript
JavaScript has predefined objects and uses standard browser objects. Some of them are discussed here
Predefined objects in JavaScript are:
String
Math
Date
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
3




String Objects: Creating a String

A string consists of one or more standard text characters between matching quote marks. JavaScript is forgiving in one regard: You can use single or double quotes, as long as you match two single quotes or two double quotes around a string. JavaScript draws a fine line between a string value and a string object. Both let you use the same methods on their contents, so by and large, you do not have to create a string object (with the new String() constructor) every time you want to assign a string value to a variable. A simple assignment operation (var myString = “fred”) is all you need to create a string value that behaves on the surface very much like a full-fledged string object.

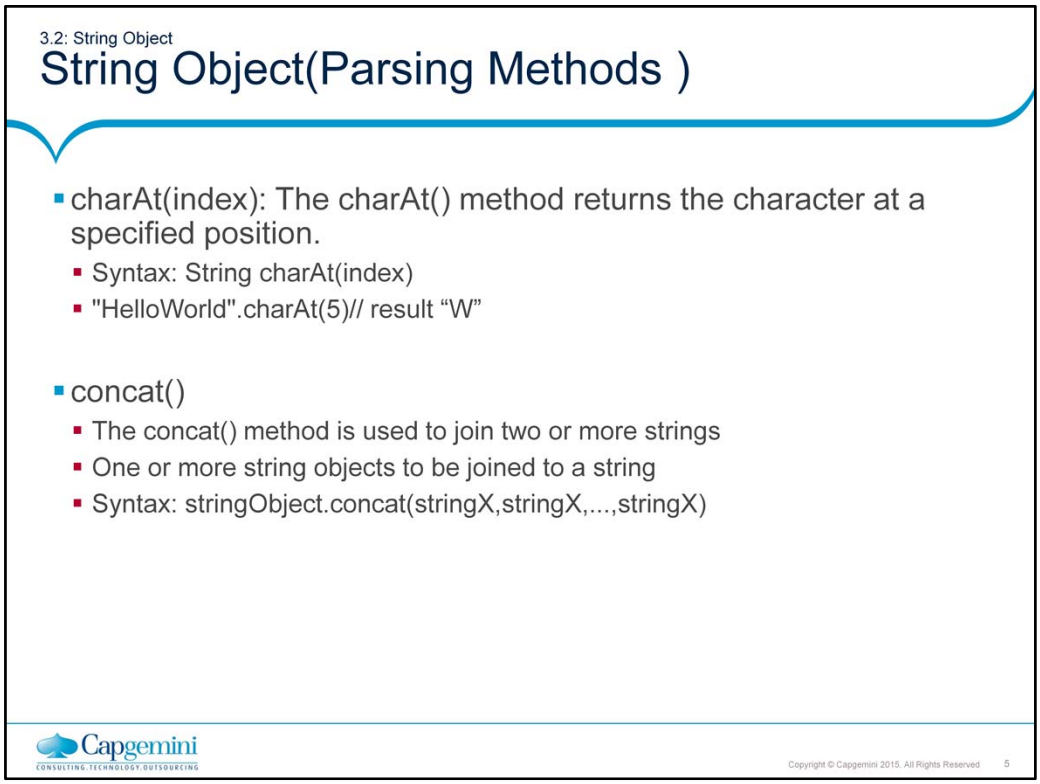
```
var myString = new String(“characters”)
var myString = “fred”
```

Properties of String Objects:

Length:

The most frequently used property of a string is length. To derive the length of a string, extract its property as you would extract the length property of any object. The length value represents an integer count of the number of characters within the string. Spaces and punctuation symbols count as characters. Any backslash special characters embedded in a string count as one character, including such characters as newline and tab.

```
“Four score”.length // result = 10
“One\ntwo”.length // result = 7
“”.length // result = 0
```




**String Objects (Parsing Methods):**

String.chartAt(index)

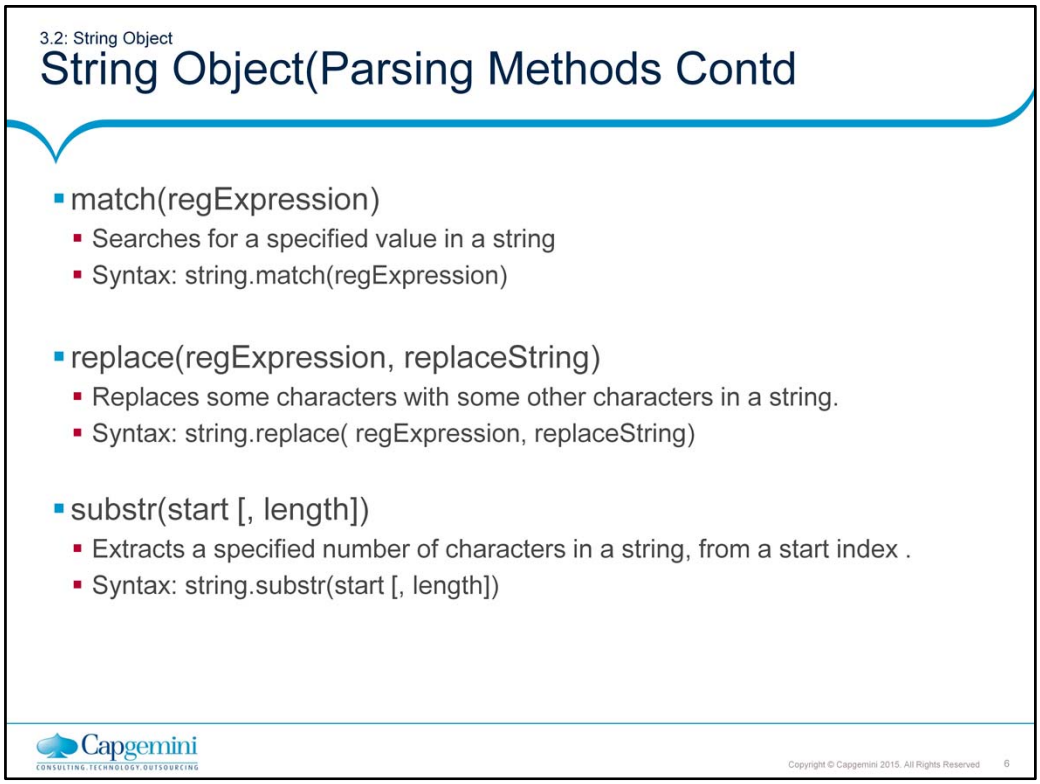
Use the string.charAt() method to extract a single character from a string when you know the position of that character. For this method, you specify an index value in the string as a parameter to the method. The index value of the first character of the string is 0. If your script needs to get a range of characters, use the string.substring() method. It is a common mistake to use string.substring() to extract a character from inside a string, when the string.charAt() method is more efficient.

string.concat( string2)

Returns: Combined string.

“abc”.concat(“def”) // result: “abcdef”

JavaScript’s add-by-value operator (+=) provides a convenient way to concatenate strings. N4 and IE4, however, introduces a string object method that performs the same task. The base string to which more text is appended is the object or value to the left of the period. The string to be appended is the parameter of the method, as the example demonstrates. Like the add-by-value operator, the concat() method doesn’t know about word endings. You are responsible for including the necessary space between words if the two strings require a space between them in the result.




**String Objects (Parsing Methods contd..):**

string.match(regExpression) **:**Returns Array of matching strings.

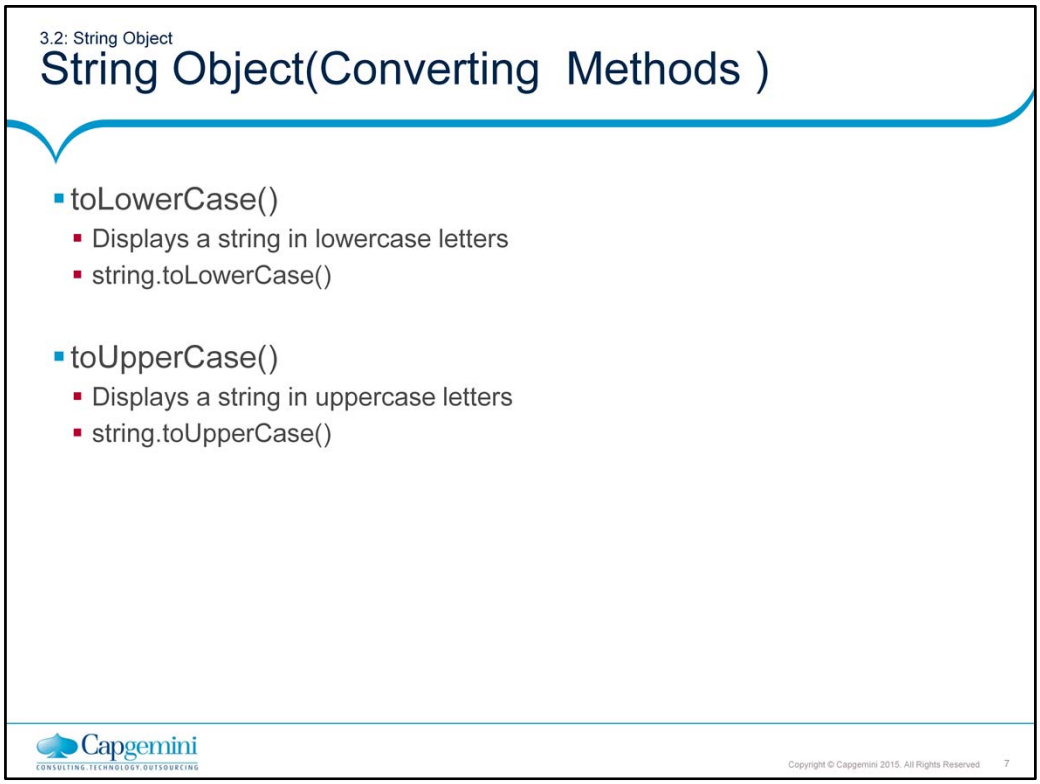
The string.match() method relies on the RegExp (regular expression) object. The parameter must be a regular expression object. This method returns an array value when at least one match turns up; otherwise the returned value is null.

string.replace( regExpression, replaceString): Returns Changed string.

Regular expressions are commonly used to perform search-and-replace operations. JavaScript's string.replace() method provides a simple framework in which to perform this kind of operation on any string.

```
var str = “To be, or not to be: that is the question.”
var regexp = /be/
str.relace(regexp, “exist”)
```

string.substr(start [, length]) :Returns Characters of the string from the first character of the given string to the specified length.

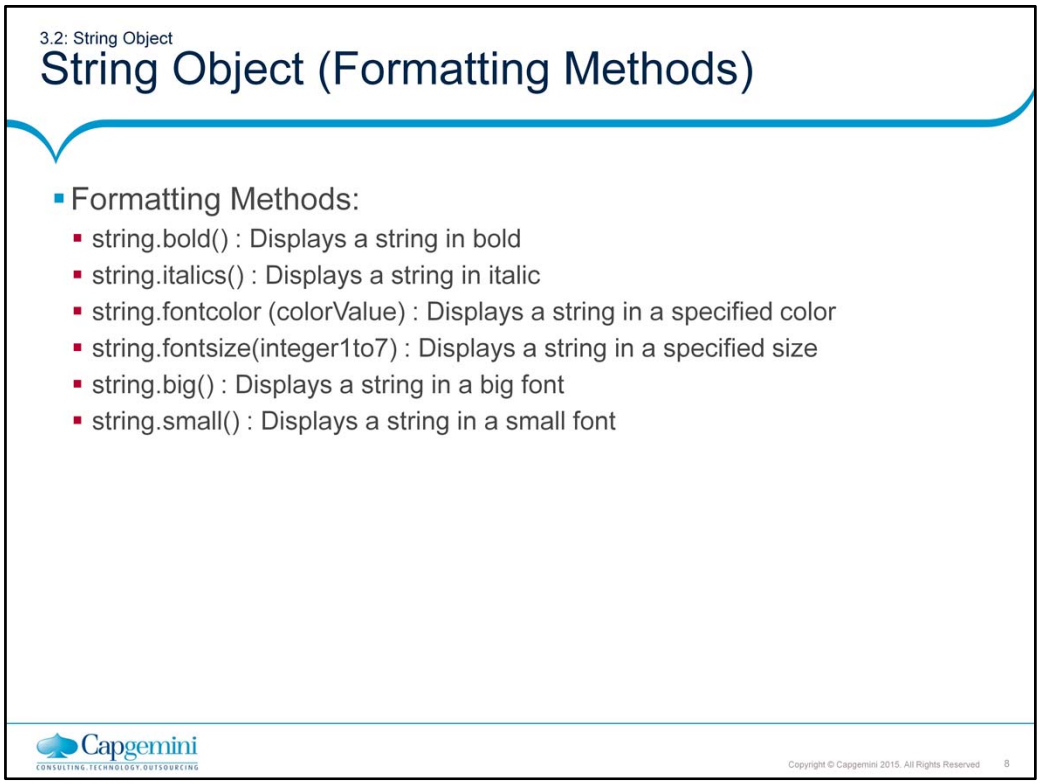



**String Objects (Parsing Methods contd..):**

string.toLowerCase() and string.toUpperCase()

Returns: The string in all lower- or uppercase, depending on which method you invoke.

A great deal of what takes place on the Internet (and in JavaScript) is case-sensitive. URLs on some servers, for instance, are case-sensitive for directory names and filenames. These two methods, the simplest of the string methods, convert any string to either all lowercase or all uppercase. Any mixed-case strings get converted to a uniform case. If you want to compare user input from a field against some coded string without worrying about matching case, you should convert both strings to the same case for the comparison.

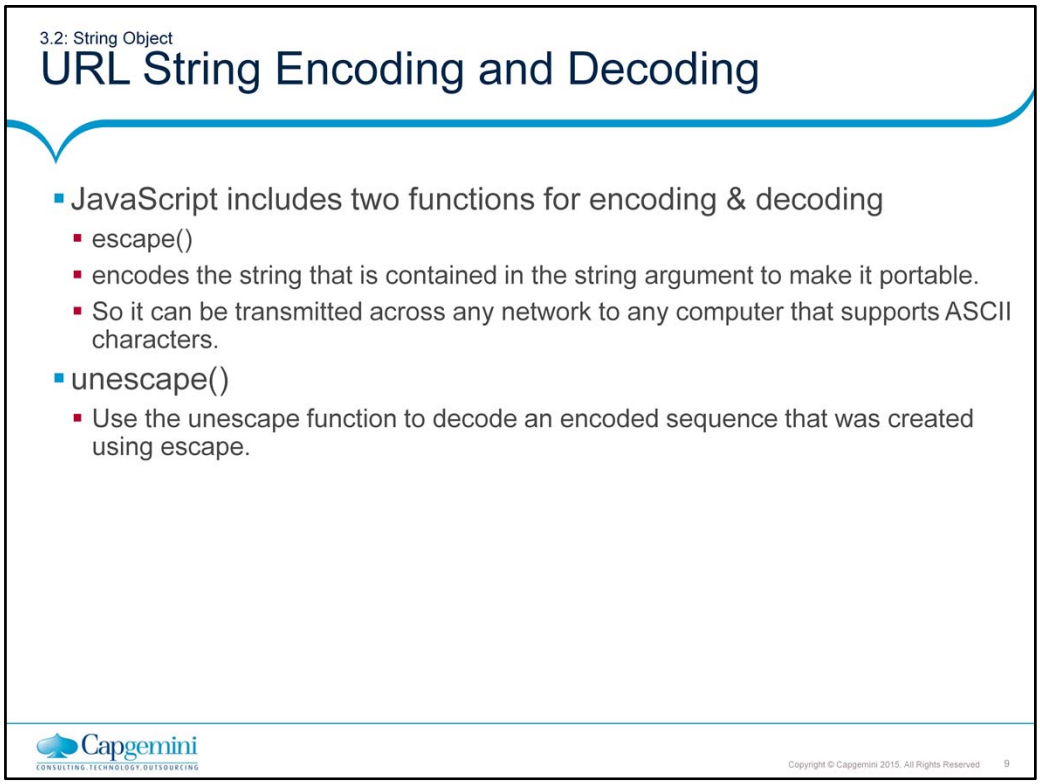



**String Objects (Formatting Methods):**

Now we come to the other group of string object methods, which ease the process of creating the numerous string display characteristics when you use JavaScript to assemble HTML code.

You can still use the standard HTML tags instead of by calling the string methods in your web pages. The choice is up to you. One advantage to the string methods is that they never forget the ending tag of a tag pair.

string.fontsize() and string.fontcolor() also affect the font characteristics of strings displayed in the HTML page. The parameters for these items are pretty straightforward —an integer between 1 and 7 corresponding to the seven browser font sizes and a color value (as either a hexadecimal triplet or color constant name) for the designated text.




**URL String Encoding and Decoding**

escape("Howdy Pardner") // result = "Howdy%20Pardner"
unescape("Howdy%20Pardner") // result = "Howdy Pardner"

When browsers and servers communicate, some nonalphanumeric characters that we take for granted (such as a space) cannot make the journey in their native form. Only a narrower set of letters, numbers, and punctuation is allowed. To accommodate the rest, the characters must be encoded with a special symbol (%)and their hexadecimal ASCII values. For example, the space character is hex 20 (ASCII decimal 32). When encoded, it looks like %20. You may have seen this symbol in browser history lists or URLs.

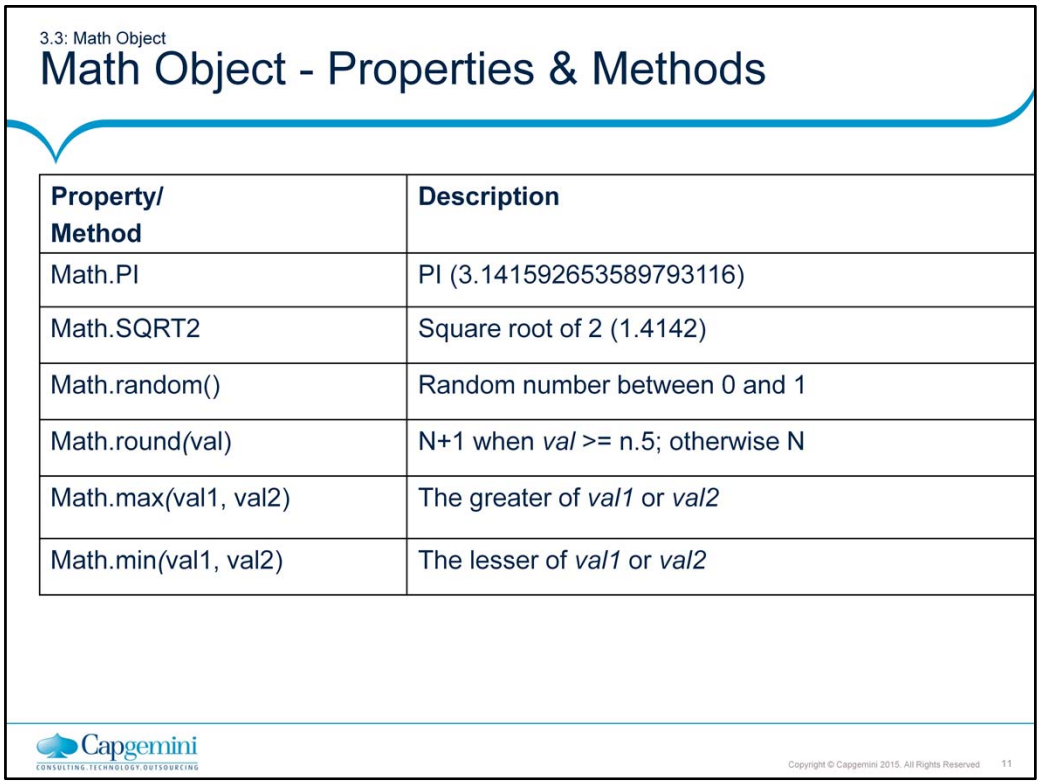
JavaScript includes two functions, escape() and unescape(), that offer instant conversion of whole strings. To convert a plain string to one with these escape codes, use the escape function, as in escape("Howdy Pardner") // result = "Howdy%20Pardner". The unescape() function converts the escape codes into human-readable form.

## Demo

- string_len.html
- string_method.html
- string_style.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10




**Math Object – Properties & Methods:**

The Math object allows you to perform mathematical tasks. All properties and methods can be called without creating the Math object.

You can use them in your regular arithmetic expressions since they return constant values. For example, to obtain the circumference of a circle whose diameter is in variable d, you use the statement shown below.

circumference = d * Math.PI

The Math.random() method returns a floating-point value between 0 and 1. If you are designing a script to act like a card game, you need random integers between 1 and 52; for dice, the range is 1 to 6 per die. To generate a random integer between zero and any top value, use the first formula shown where n is the top number.

To generate random numbers between a different range use the second formula where m is the lowest possible integer value of the range and n equals the top number of the range minus m. In other words n+m should add up to the highest number of the range you want. For the dice game, use the third formula for each throw of the die.

Math.round(Math.random() * n)

Math.round(Math.random() * n) + m

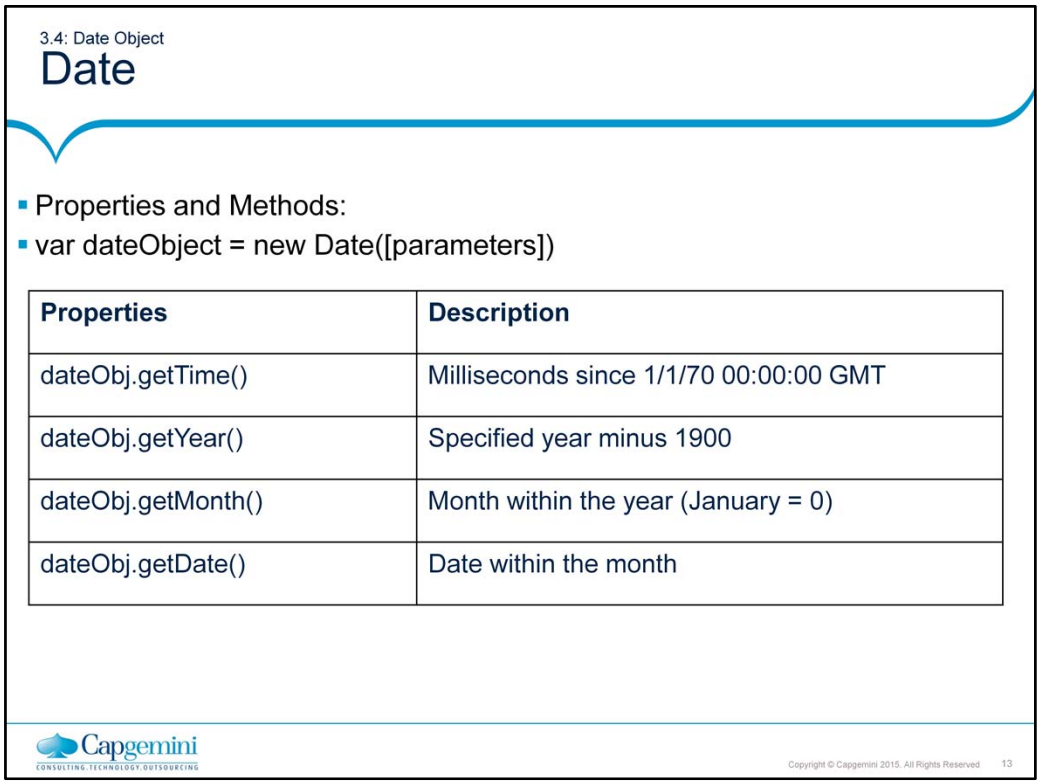
newDieValue = Math.round(Math.random() * 5) + 1

## Demo

- Rand_fun.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

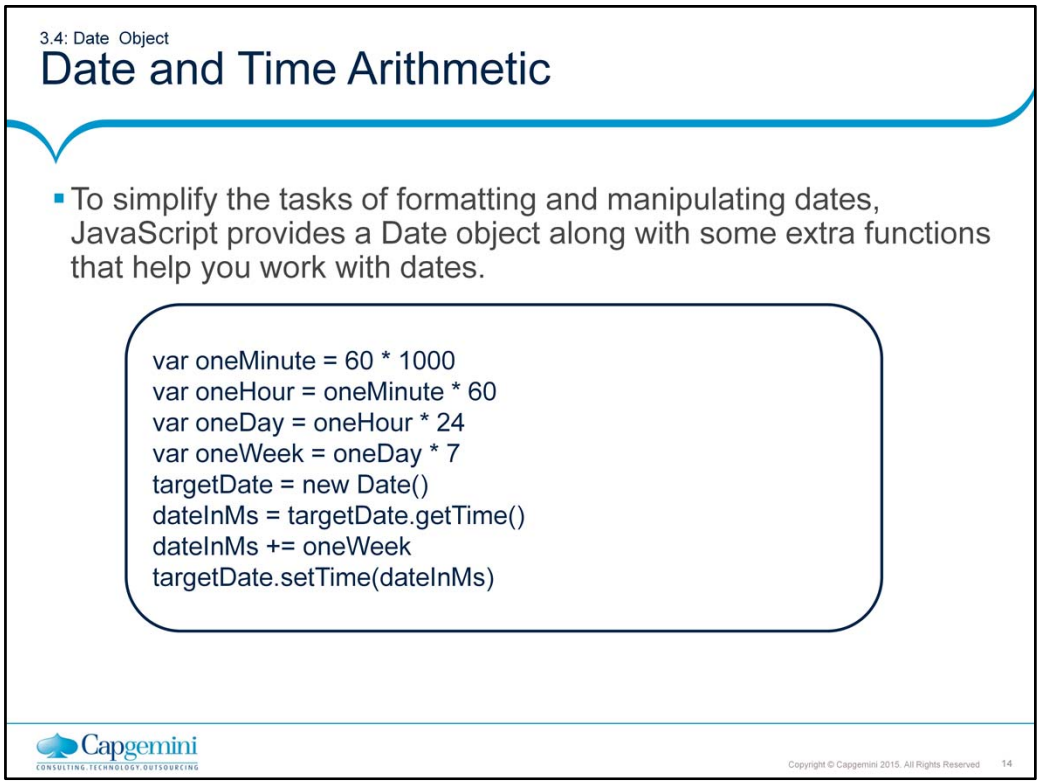



**Date Object: Creating a Date object**

```
var dateObject = new Date([parameters])
new Date("Month dd, yyyy hh:mm:ss")
new Date("Month dd, yyyy")
new Date(yy,mm,dd,hh,mm,ss)
new Date(yy,mm,dd)
new Date(milliseconds)
```

The Date object evaluates to an object rather than to some string or numeric value. If you leave the parameters empty, JavaScript takes that to mean you want today's date and the current time to be assigned to that new Date object. To create a Date object for a specific date or time, you have five ways to send values as a parameter to the new Date() constructor function.

The Date object has only a prototype property, which enables you to apply new properties and methods to every Date object created in the current page.




**Date and time arithmetic:**

You may need to perform some math with dates for any number of reasons. Perhaps you need to calculate a date at some fixed number of days or weeks in the future or figure out the number of days between two dates. When calculations of these types are required, remember the *lingua franca* of JavaScript date values: the milliseconds.

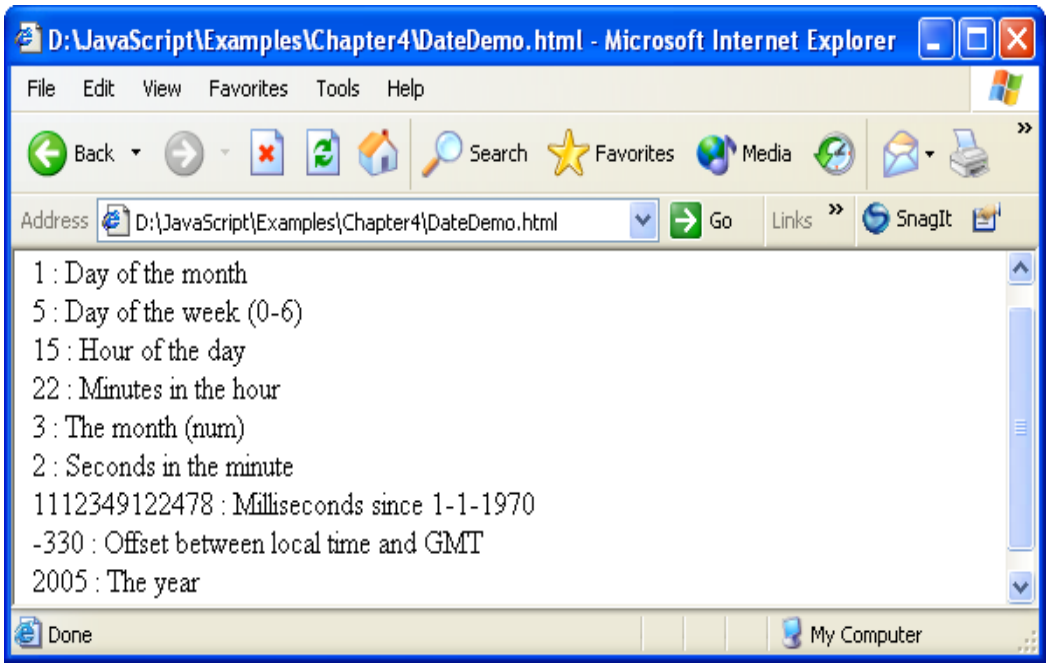
What you may need to do in your date-intensive scripts is establish some variable values representing the number of milliseconds for minutes, hours, days, or weeks, and then use those variables in your calculations. On the slide you can see an example that establishes some practical variable values, building on each other.

With these values established in a script, you can use one to calculate the date one week from today. Following is the complete example.

**Date and time arithmetic (contd..):**

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>
<script>
dateinfo = new Date();
document.write(dateinfo.getDate() + " : Day of the month" + "<br>")

document.write(dateinfo.getDay()              +" : Day of the week  (0-6)" +
"<br>" )
document.write(dateinfo.getHours()+" : Hour of the day " + "<br>")

document.write(dateinfo.getMinutes()+" : Minutes in the hour" + "<br>")
document.write(dateinfo.getMonth()+" : The month              (num)" +
"<br>" )
document.write(dateinfo.getSeconds()+ " : Seconds in the minute " +
"<br>")
document.write(dateinfo.getTime()+" : Milliseconds since 1-1-1970" +
"<br>")
document.write(dateinfo.getTimezoneOffset() +" : Offset between local time
and GMT" + "<br>")
document.write(dateinfo.getYear() +" : The year" + "<br>")
</script>
</BODY>
</HTML>
```

And it produces the output as:



## Demo

- Date_info.html
- DateDifference.html
- Utc_ex.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

Add the notes here.

## Lab

- Lab 3:
  - Working with Predefined Core Objects

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

Add the notes here.

## Summary

- Predefined objects of JavaScript like String, Math and Date
- How to use predefined objects
- How to manipulate their properties and invoke methods


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

## Review Question

- Question 1: Which is the method to extract a single character from a string when you know the position of that character.
  - Option 1: string.charAt()
  - Option 2: string.charAtIndex()
- Question 2: getDate() returns the day within the month.
  - True/False
- Question 3: To convert a plain string to one with these escape codes, use the _____ function

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

# Web Basics-JavaScript

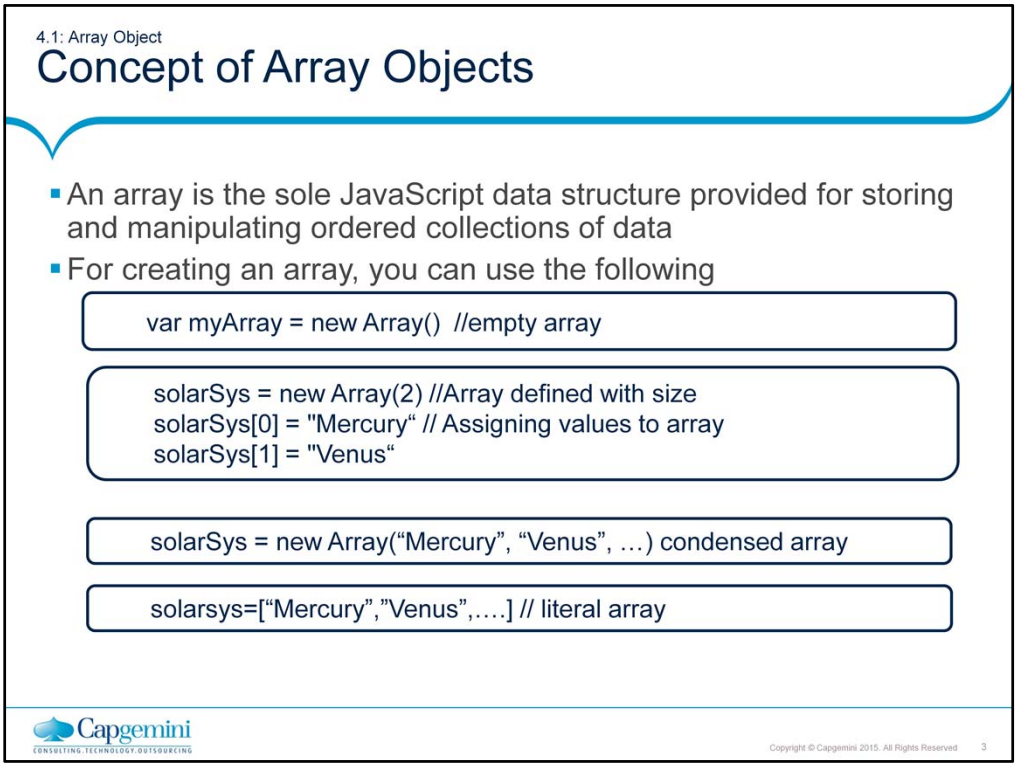
Lesson 4: Working with Arrays

# Lesson Objectives

- In this lesson, you will learn about:
  - Array object
  - Properties and methods of Array objects

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2




Array Objects:

An array is a kind of variable that can hold more than one value at a time.

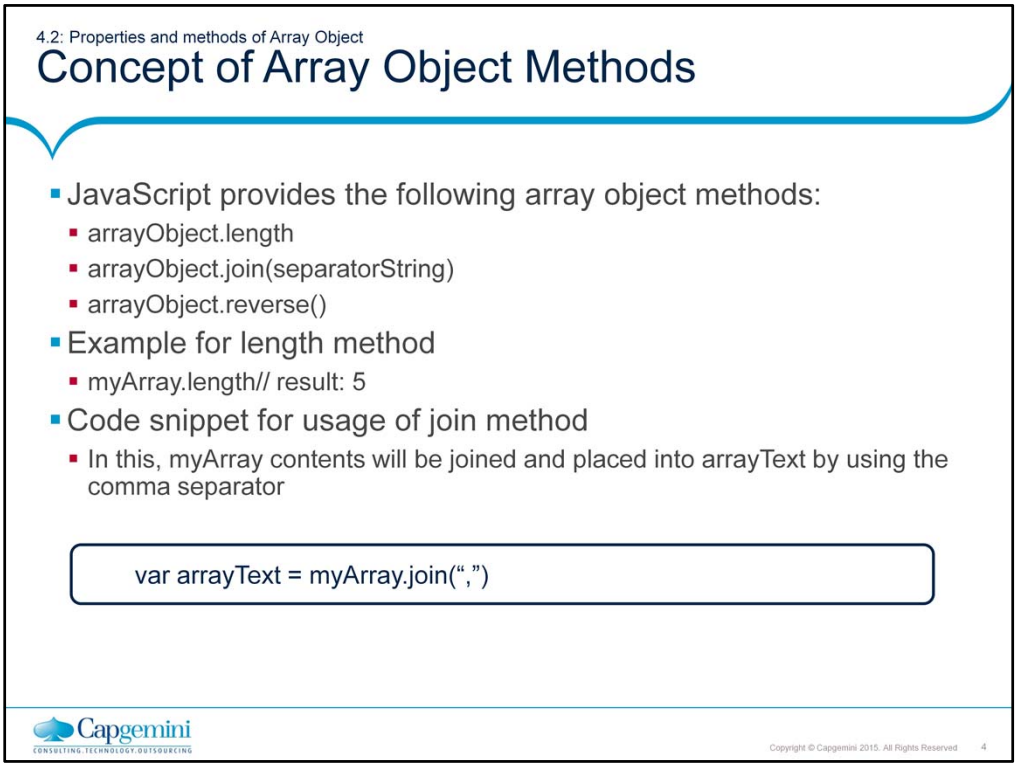
However, unlike some other programming languages, JavaScript's arrays are very forgiving as to the kind of data you store in each cell or entry of the array. This allows, for example, an array of arrays, providing the equivalent of multidimensional arrays customized to the kind of data your application needs.

You can see a few examples listed on the slide for creating an array. We see example of creating an empty array. To limit the size of array you can specify the optional integer value as seen in the second example.

Another way of defining an array is called as condensed array which allows you to combine the array and array elements definitions into one step.

Literal arrays are define by assigning the value in square brackets. To create an array with initial undefined values you can simple enter a comma. For eg

myarray=[“Pune”, , , “Mumbai]




Array Object Methods:

After you have information stored in an array, JavaScript provides several methods to help you manage that data.

arrayObject.join(separatorString)

It returns string of entries from the array delimited by the separatorString value.

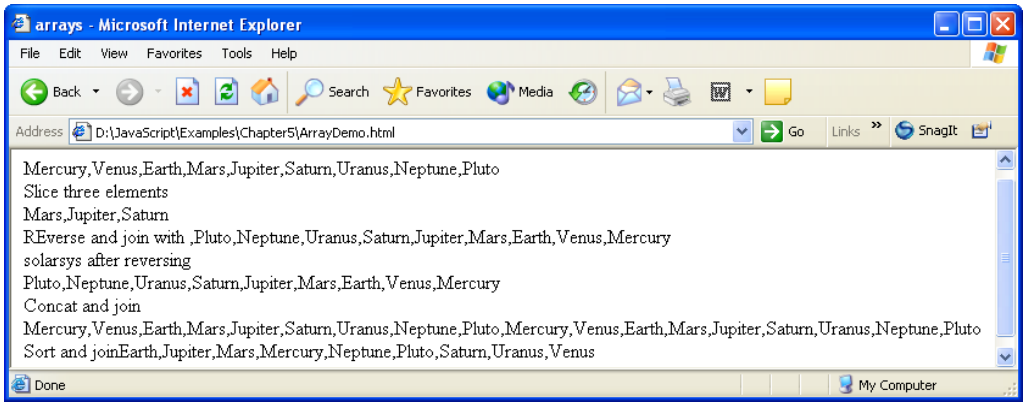
```
var arrayText = myArray.join(“,”)
```

arrayObject.reverse()

It returns array of entries in the op.

The element that was last in the array becomes the o index item in the array. Note that when you do this, you are restructuring the original array, and not copying it.


Demo of Array object (Arr_Demo.html) produces the output as shown below:



# Lab

- Lab 4 :
  - Working with Arrays

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Summary

- An array is a set of variables
- You can create an array object by using new operator
- Array Object properties are length
- Array Object Methods are concat, join, reverse, slice etc


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

## Review Question

- Question 1: The ___ method allows you to join array contents and place it into a text
  - Option 1: array.concat()
  - Option 2: array.join()
- Question 2: An array object automatically has a size property.
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

# Web Basics-JavaScript

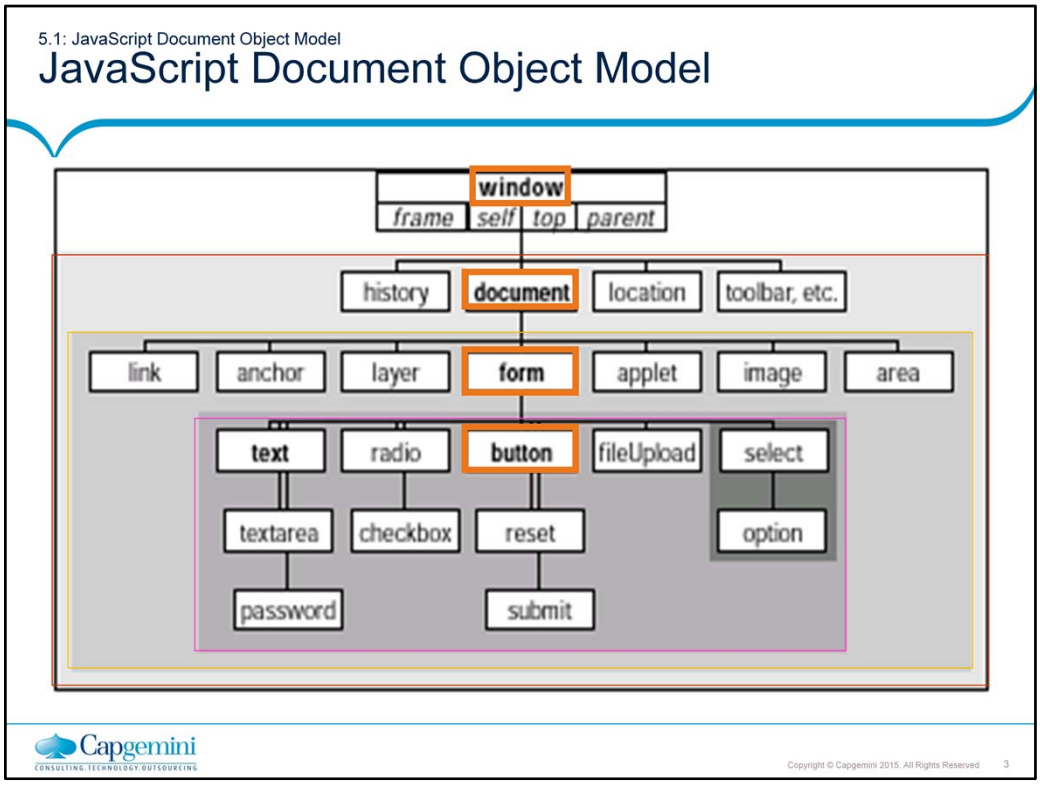
Lesson 5: Document Object Model

# Lesson Objectives

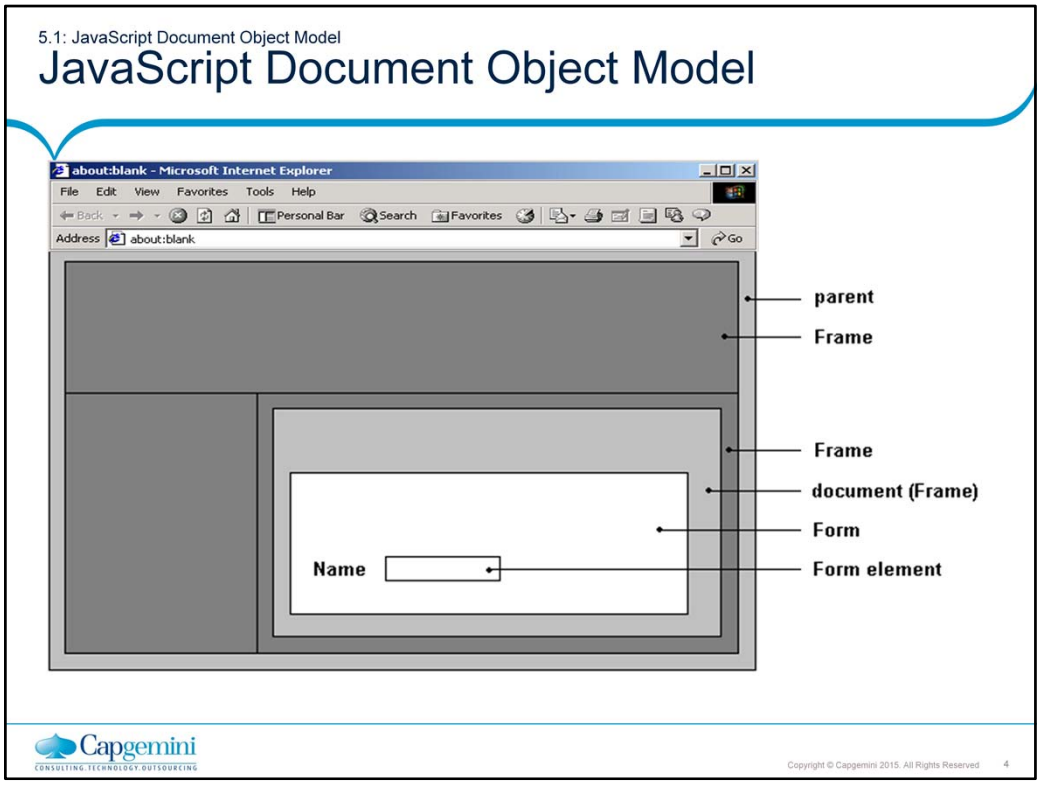
- After completing this module you will be able to:
  - Understand the JavaScript Object Model
  - Understand the *Window* object and Navigator Object
  - Working with Location and History Object

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



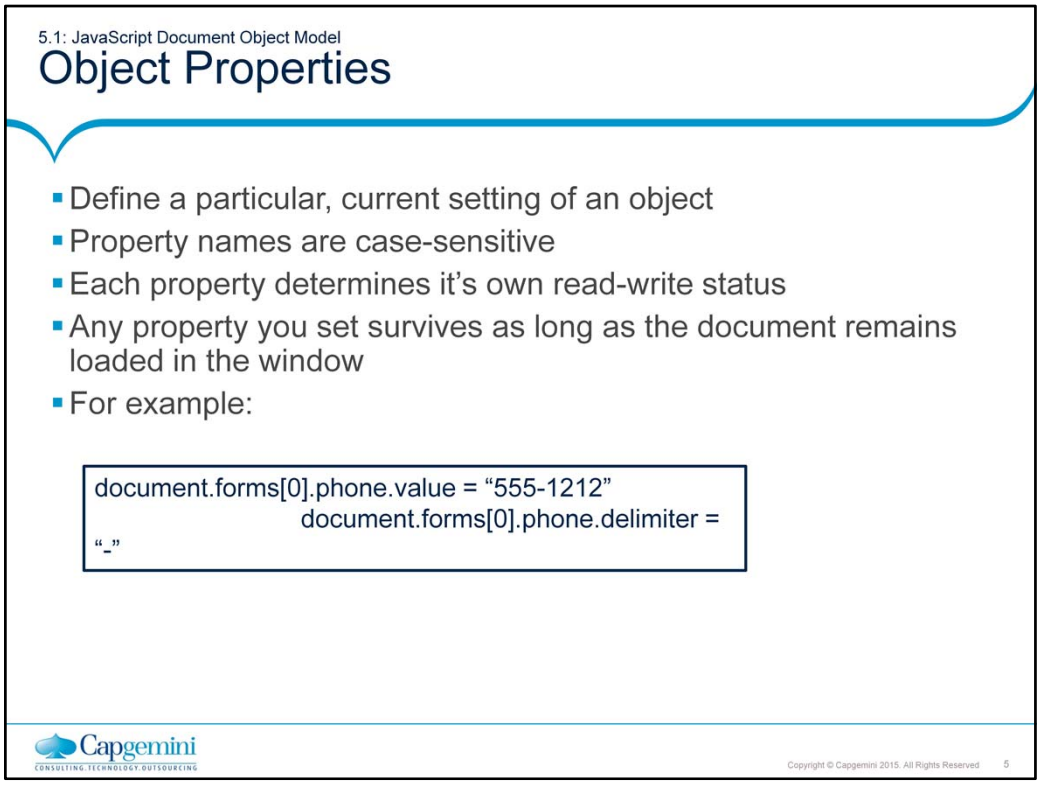

The figure shows the complete JavaScript document object hierarchy as implemented in Netscape Navigator 4. Notice that the window object is the topmost object in the entire scheme. Everything you script in JavaScript is in the browser's window, be it the window itself or a form element. Of all the objects shown in the figure, you are likely to work most with the ones that appear in **boldface**. Objects whose names appear in *italics* are synonyms for the window object, and are used only in some circumstances. Pay attention to the shading of the concentric rectangles. Every object in the same shaded area is at the same level relative to the window object. When a link from an object extends to the next darker shaded rectangle, that object contains all the objects in darker areas. There exists at most one of these links between levels. A window object contains a document object; a document object contains a form object; a form object contains many different kinds of form elements. Study this figure to establish a mental model for the scriptable elements of a Web page. After you script these objects a few times, the object hierarchy will become second nature to you — even if you do not remember every detail ( property, method, and event handler) of every object. At least you know where to look for information.




Creating JavaScript Objects

Most of the objects that a browser creates for you are established when an HTML document loads into the browser. The same kind of HTML code you used to create links, anchors, and input elements tell a JavaScript-enhanced browser to create those objects in memory. The objects are there whether or not your scripts call them into action.

The only visible differences to the HTML code for defining those objects are one or more optional attributes specifically dedicated to JavaScript. By and large, these attributes specify the event you want the user interface element to react to and what JavaScript should do when the user takes that action. If you rely on the document's HTML code to perform the object generation, you spend more time figuring out how to do things with those objects or have them do things for you. Bear in mind that objects are created in their load order, which is why you should put most, if not all, deferred function definitions in the document's Head. If you create a multi-frame environment, a script in one frame cannot communicate with another frame's objects until both frames load.




Object Properties

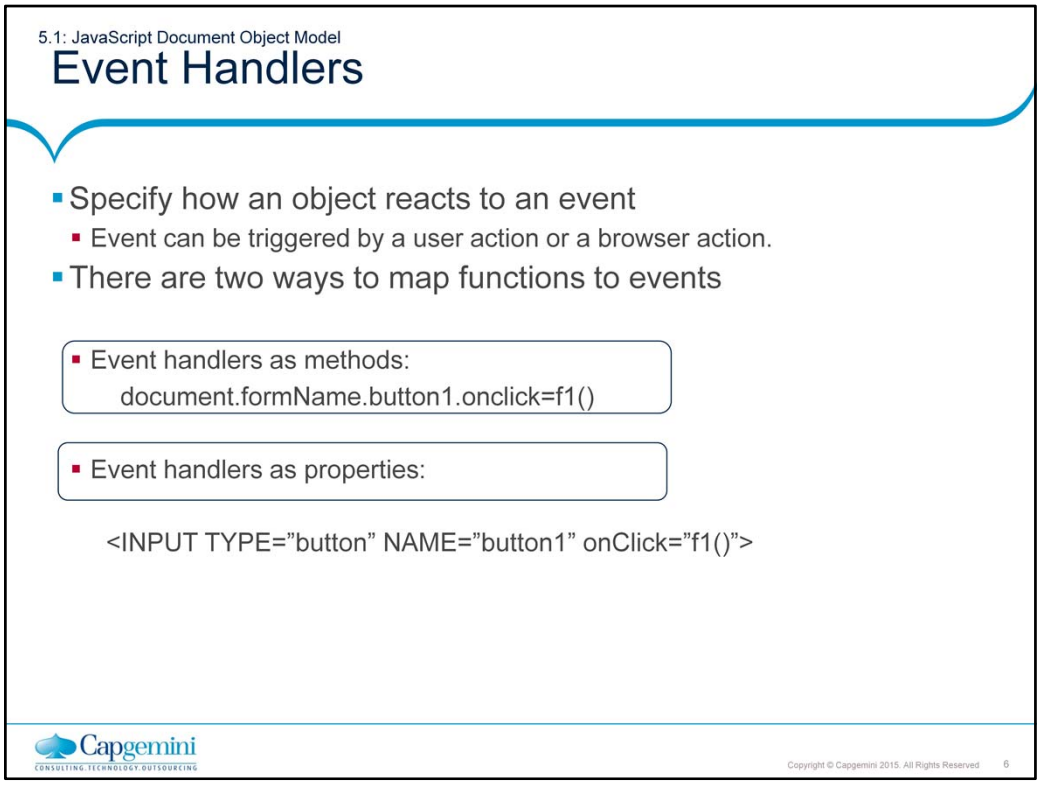
A property generally defines a particular, current setting of an object. The setting may reflect a visible attribute, such as a document's background color. It may also contain information that is not so obvious, such as the form *action* and *method* when it is submitted.

Document objects have most of their properties assigned by attribute settings of HTML tags that generate the objects. Thus, a property may be a string (for example, a name) or a number (for example, a size). A property can also be an array, such as an array of images contained by a document. If the HTML does not include all attributes, the browser usually provides default value for both attributes and corresponding JavaScript properties.

When used in script statements, property names are case-sensitive. Therefore, if you see a property name listed as *bgColor*, you must use it in a script statement with that exact case usage. But when you set an initial value of a property by way of an HTML attribute, the attribute name ( like all of HTML) is not case-sensitive. Thus, **<BODY BGCOLOR="white">** and **<body bgcolor="white">** both set the same property value.

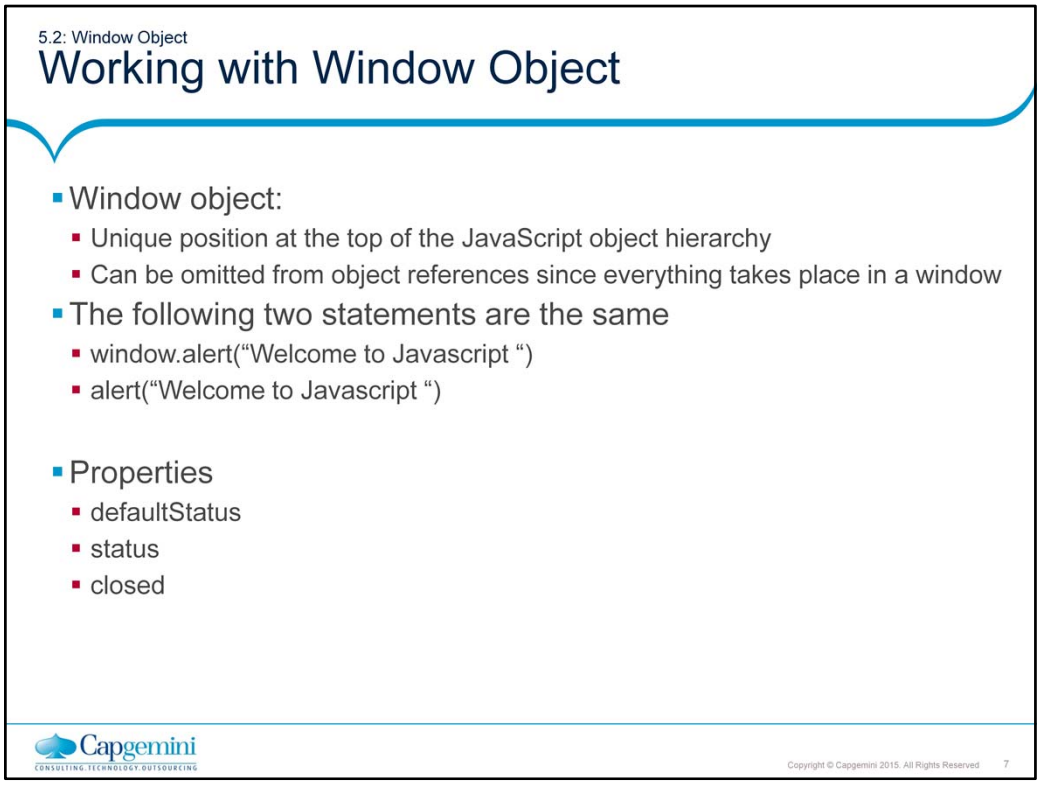
Object Methods

An object's method is a command that a script can give to that object. Some methods return values, but that is not a prerequisite for a method. Also, not every object has methods defined for it. In a majority of cases, invoking a method from a script causes some action to take place. It may be an obvious action, such as resizing a window, or something more subtle, such as processing a mouse click.




Object Event Handlers

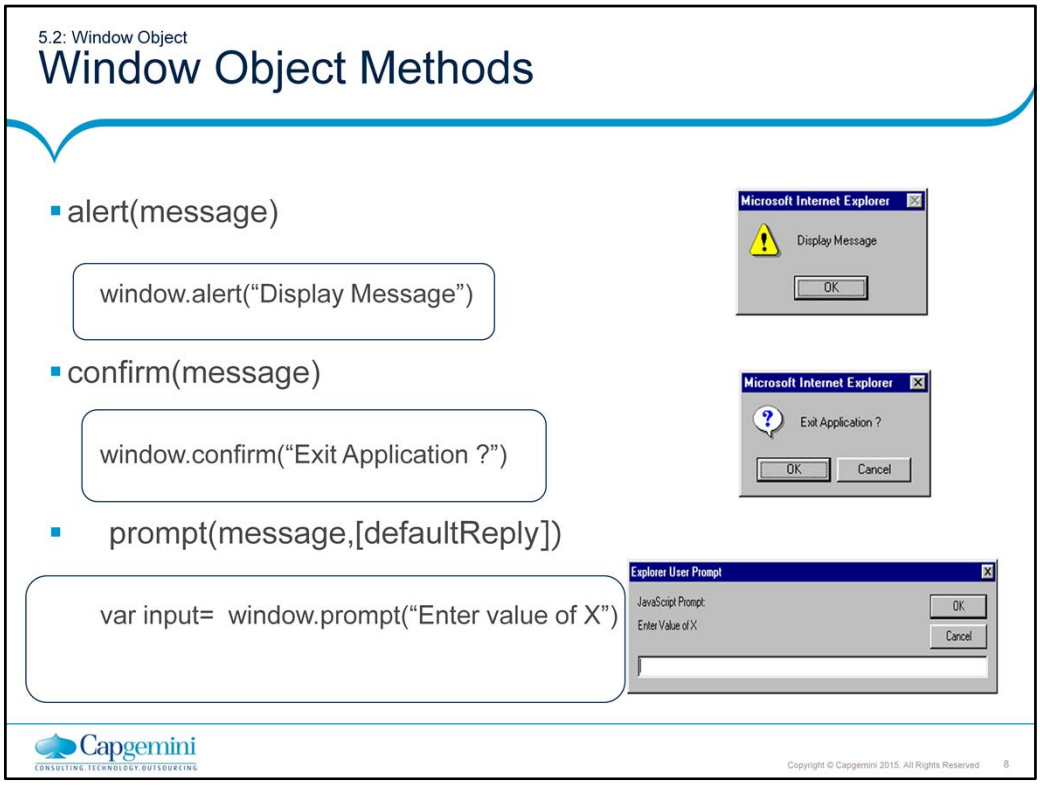
Event handlers specify how an object reacts to an event, whether the event is triggered by a user action (for example, a button click) or a browser action (for example, the completion of a document load). Event Handlers can be specified as methods or they can be specified using attributes in tags.



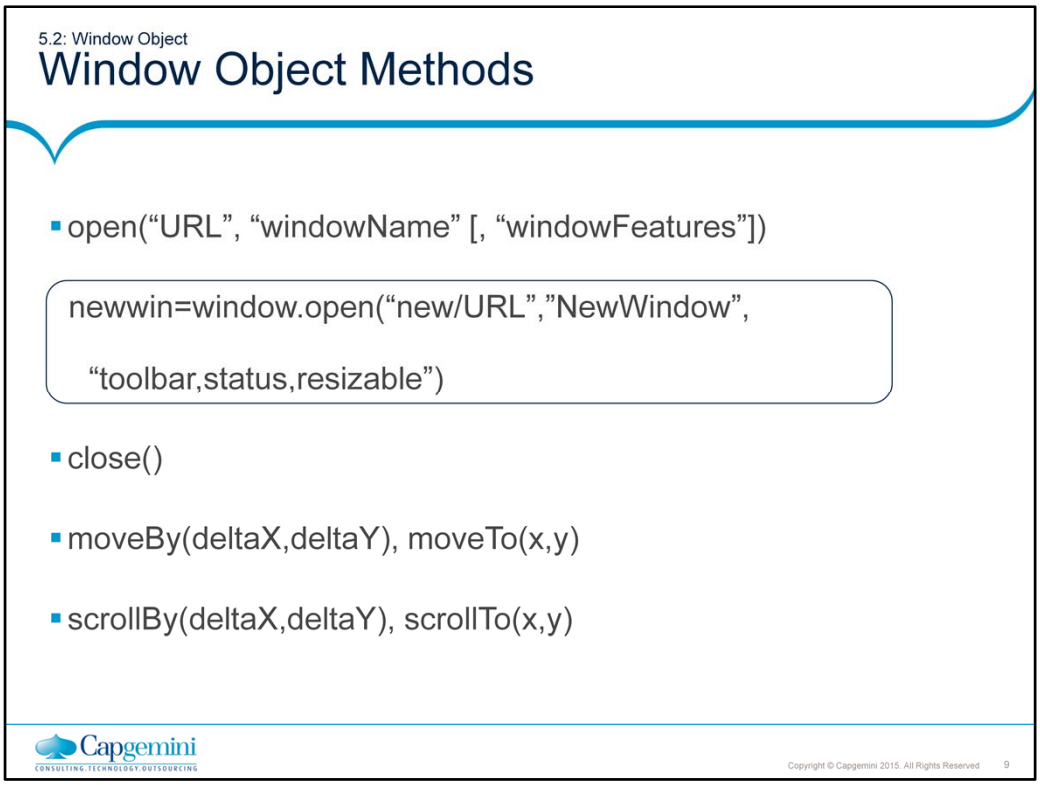

About this Object

The *window* object has the unique position of being at the top of the JavaScript object hierarchy. This exalted location gives it a number of properties and behaviors unlike any other object. Among the list of properties for the window object is one called *self*. This property is synonymous to the window object itself. When you start your browser, it usually opens a window. That window is a valid window object, even if it is blank. This object is also the level at which a script asks the browser to display any of the three styles of the dialog boxes (a plain alert dialog box, an OK-Cancel confirmation dialog box, or a prompt for user text entry).

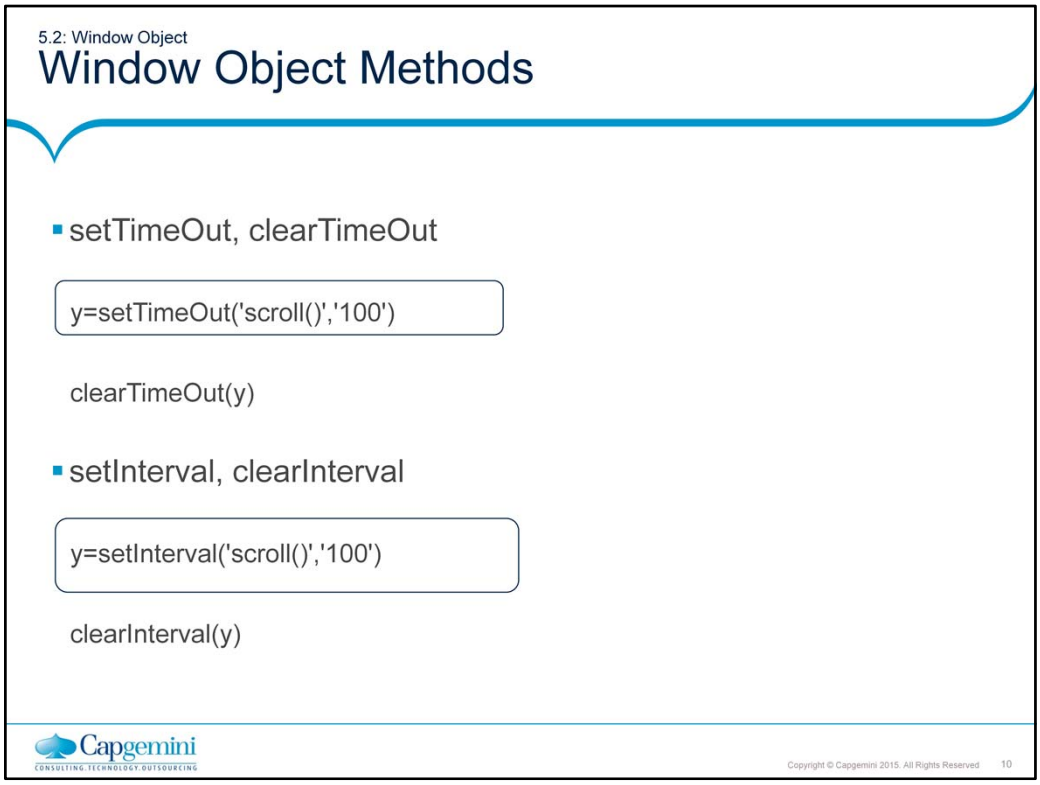
| Property | Description |
|---|---|
| defaultStatus | window.defaultStatus property is normally an empty string, it sets or returns the default text which is in the statusbar of the window |
| Status | This property sets a text value to be displayed in the status bar |
| closed | Returns a boolean value which indicated if the window has been closed or no |



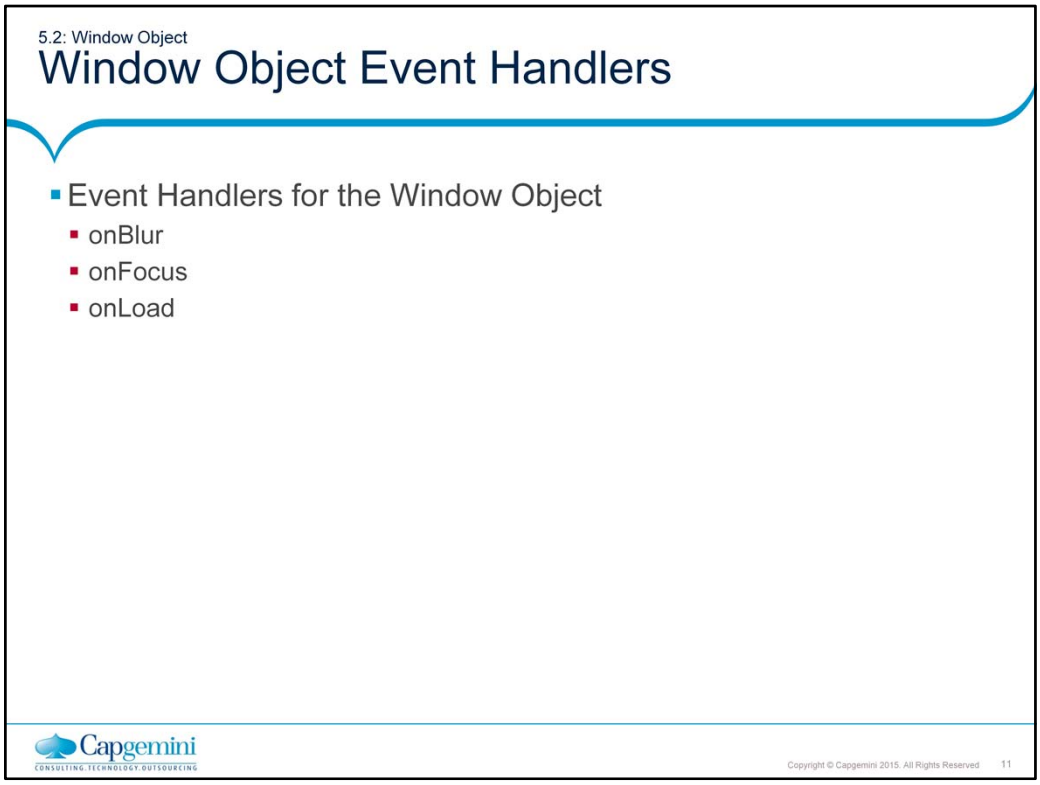

| Method | Description |
|---|---|
| alert(message) | An alert dialog box is a modal window that presents a message to the user with a single OK button to dismiss the dialog box. |
| confirm(message) | A confirm dialog box presents a message in a modal dialog box along with OK and Cancel buttons. Such a dialog box can be used to ask a question of the user, usually prior to a script performing actions that will not be undoable. |
| prompt(message, defaultReply) | The third kind of dialog box that JavaScript can display includes a message from the script author, a field for user entry, and two buttons (OK and Cancel). |




| | |
|---|---|
| open("URL", "windowName" [, "windowFeatures"]) | The *window.open()* method, provides a Web site designer with options for the way a new brwoser window should look on the user's computerscreen. The optional *windowFeatures* parameter is *one strin*g, that comprises a comma-separated list of assignment expressions. Boolean values for true can be either yes, 1, or just the feature name by itself; for false, use a value of no or 0. If you omit any Boolean attributes, they are rendered as false. Therefore, if you want to create a new window that shows only the toolbar and statusbar and is resizable, the method looks like this: window.open("*newUR*L","NewWindow", "toolbar,status,resizable"). |
| close() | The *window.close()* method closes the browser window referenced by the window object. |
| scrollBy(deltaX,deltaY) scrollTo(x,y) | scrollBy(..) method scrolls the content by the specified number of pixels which is relative scroll. scrollTo(..) is an absolute scroll to the specified coordinates |
| moveBy(deltax,delta Y) moveTo(x,y) | moveBy(..) moves the window relative to the current position. moveT(..) moves it to the specified coordinates |




| | |
|---|---|
| setTimeout(" functionOrExpr", msecDelay [, funcarg1, ..., funcargn]) | Javascript holds a statement or function from executing for the desired amount of time. The timeout value is in milliseconds |
| setInterval( "functionOrExpr", msecDelay,language) | Use this method when your script needs to call a function or execute some expression repeatedly with a fixed time delay between calls to that function or expression. The timeinterval is in milliseconds. Optional Language i.e Javascript,vbscript |
| clearInterval (intervalIDnumber) | Use this method to turn off an interval loop action started with the *window.setInterval()* method. The parameter is the ID number returned by the *setInterval()* method. |
| clearTimeout (timeoutIDnumber) | Use the *clearTimeout()* method in concert with the *window.setTimeout()* method when you want your script to cancel a timer that is waiting to run its expression. The parameter for this method is the ID number that the *setTimeout()* method returns when the timer starts ticking. |




Event Handlers

Table 6.3 Window Object Event Handlers

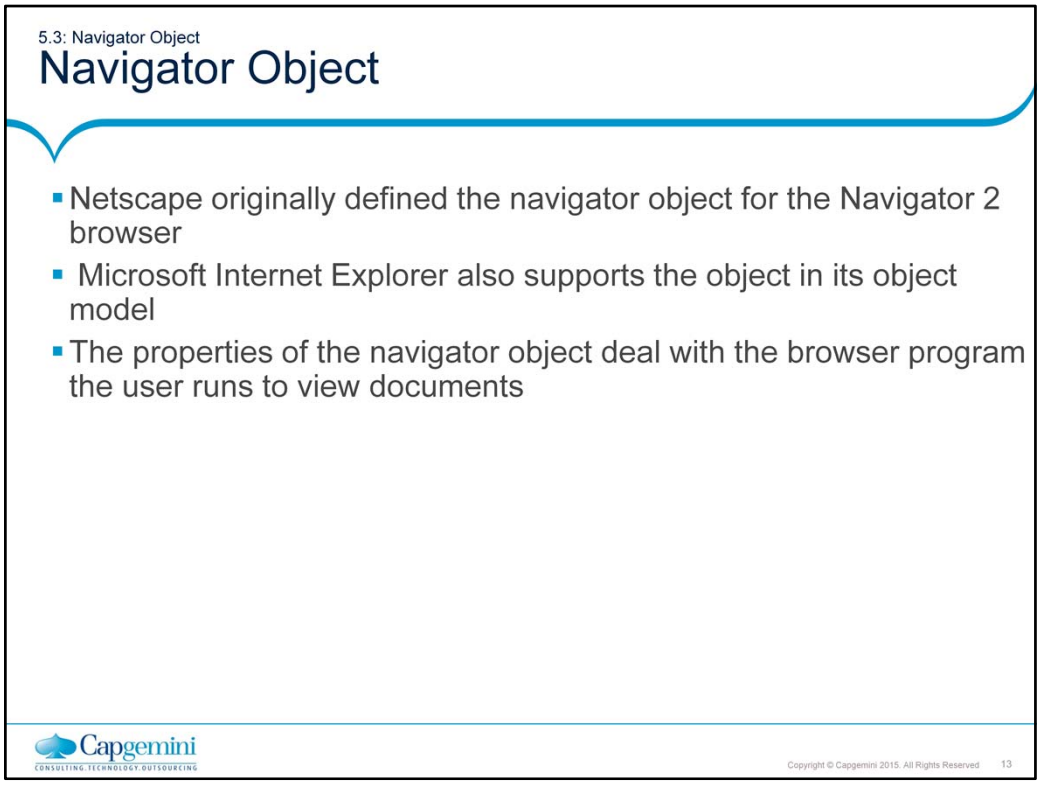
| Event Handler | Description |
|---|---|
| onBlur<br>onFocus | Fired when window or frame has been activated and deactivated respectively. |
| onLoad | The load event is sent to the current window at the end of the document loading process. |

## Demo

- Window_object.html
- setTimeOut_method.html
- Window_ex.html
- setInterval_method.html

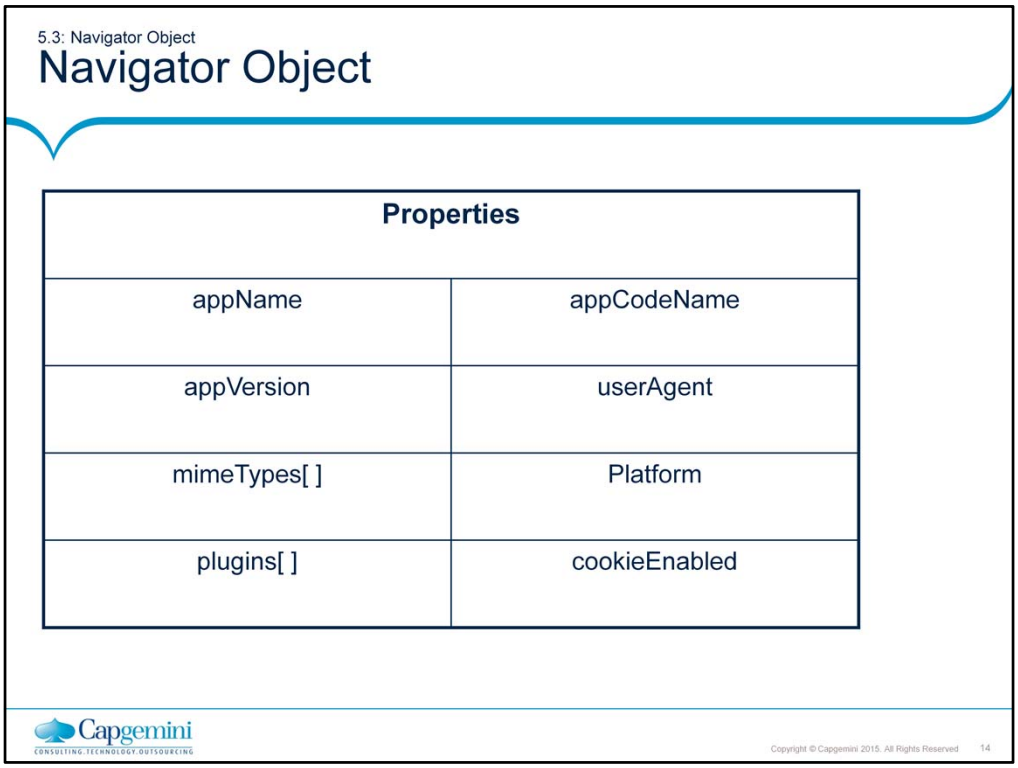

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12




Navigator Object

Netscape originally defined the navigator object for the Navigator 2 browser. Microsoft Internet Explorer also supports the object in its object model. Properties of the navigator object deal with the browser program the user runs to view documents. Properties include those for extracting the version of the browser and the platform of the client running the browser.




| Property | Description |
|---|---|
| appName<br>appCodeName<br>appVersion<br>userAgent<br>Platform<br>cookieEnabled | The *appName* and *appCodeName* properties are simply the official name and the internal code name for the browser. *appVersion* returns version information of the browser and *userAgent* returns the user-agent header sent by the browser to the server. Platform returns for which platform the browser is compiled.<br>cookieEnabled determines if cookies are enabled in the browser |
| plugins[] | Returns an array of plugins available on the client browser. |
| mimeTypes[] | Returns an array of MIME types supported by the browser |

## Demo

- property_bro_ver_name.html
- plugin_object.html
- MIME_Type.html

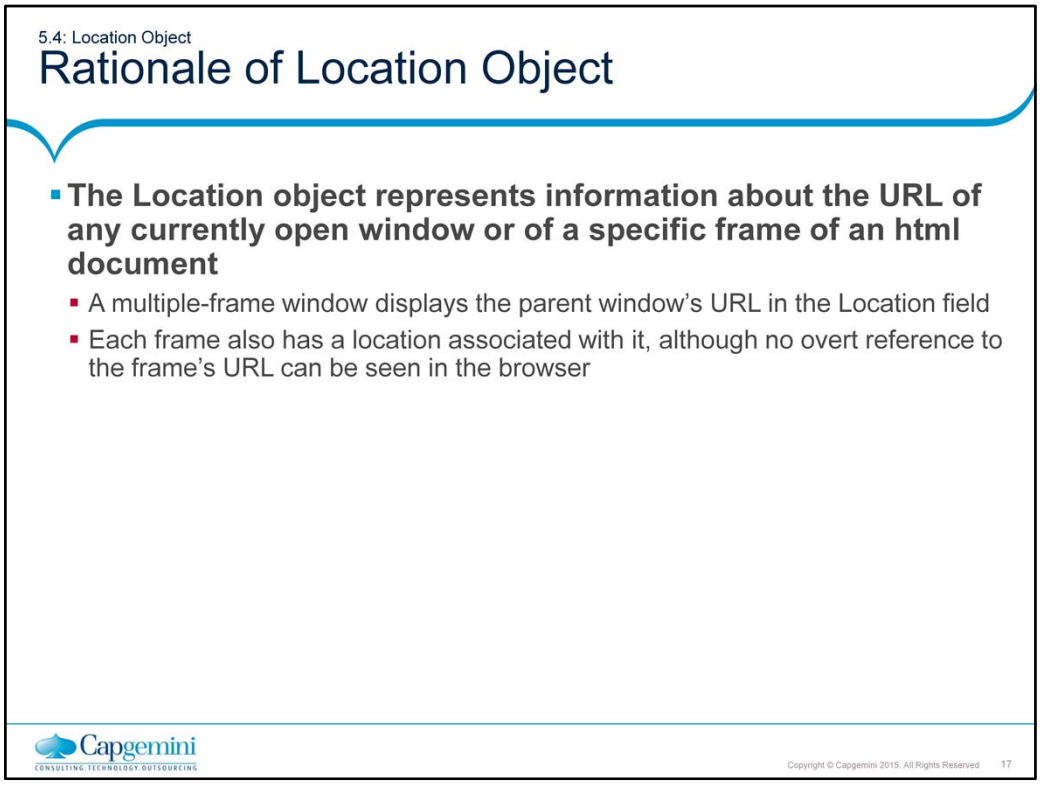

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

# Lab

- Lab 5 :
  - Working with Document Object Model (DOM)




Working with Location Object:

To get URL information about a document located in another frame, the reference to the location object must include the window frame reference.

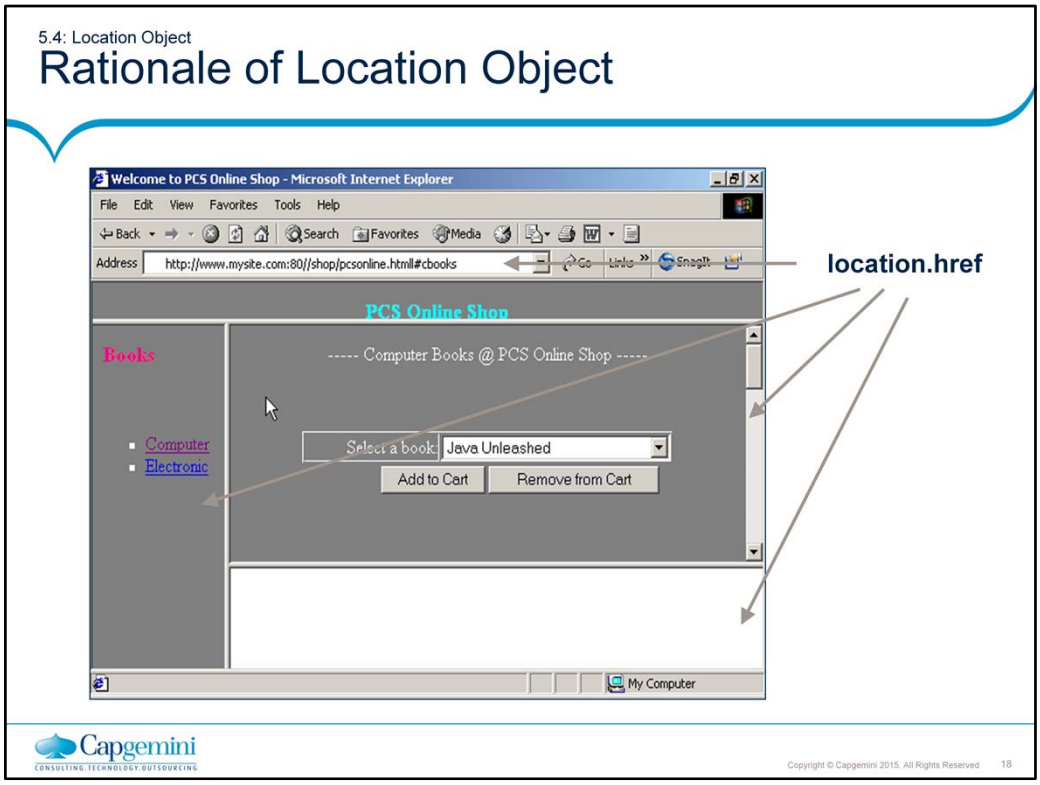
Most properties of a location object deal with network-oriented information.

This information includes various data about the physical location of the document on the network, including the host server, the protocol being used, and other components of the URL

The **window.location** object can be handy when a script needs to extract information about the URL, perhaps to obtain a base reference on which to build URLs for other documents to be fetched as the result of user action.

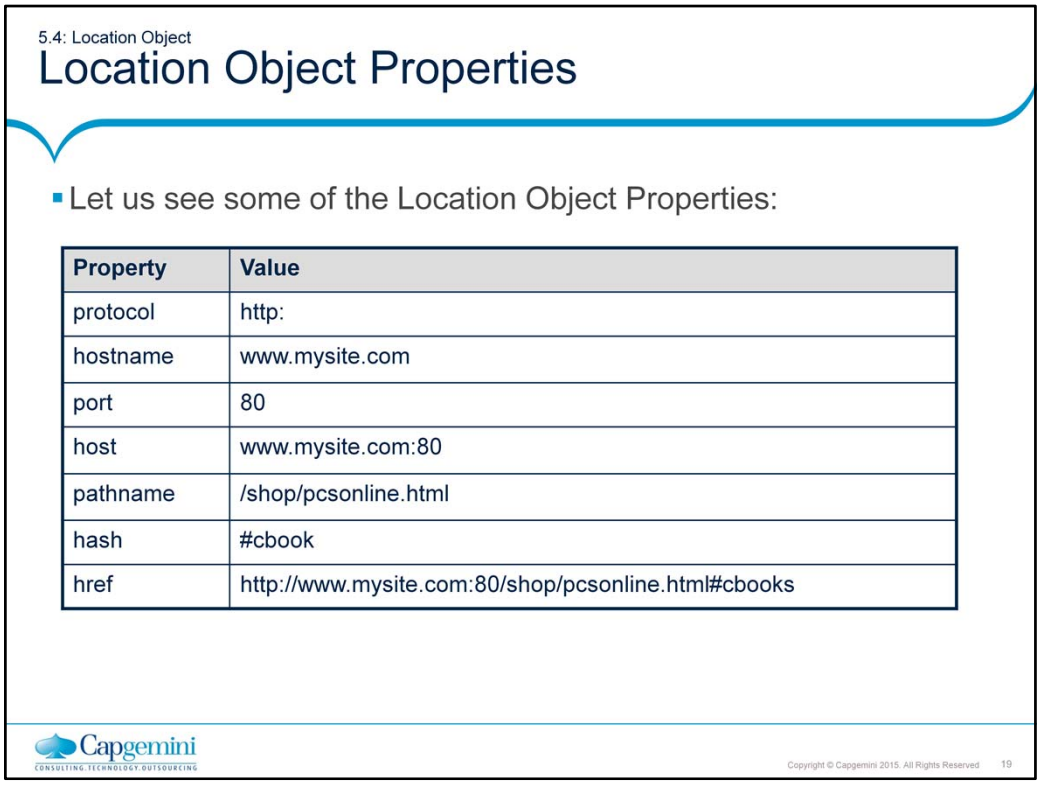
Setting the value of some location properties is the preferred way to control the document that gets loaded into a window or frame.

Location object properties and methods



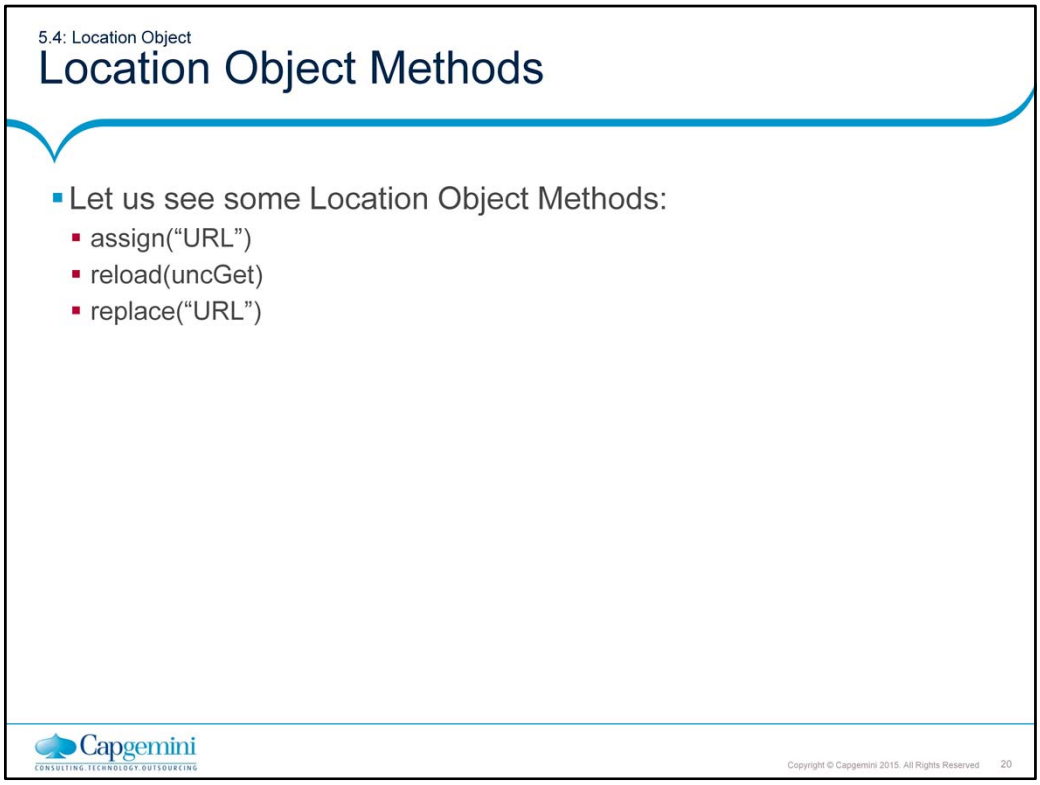

Working with Location Object:

- Given a complete URL for a typical WWW page, the **window.location** object assigns property names to various segments of the URL as shown in the above slide.



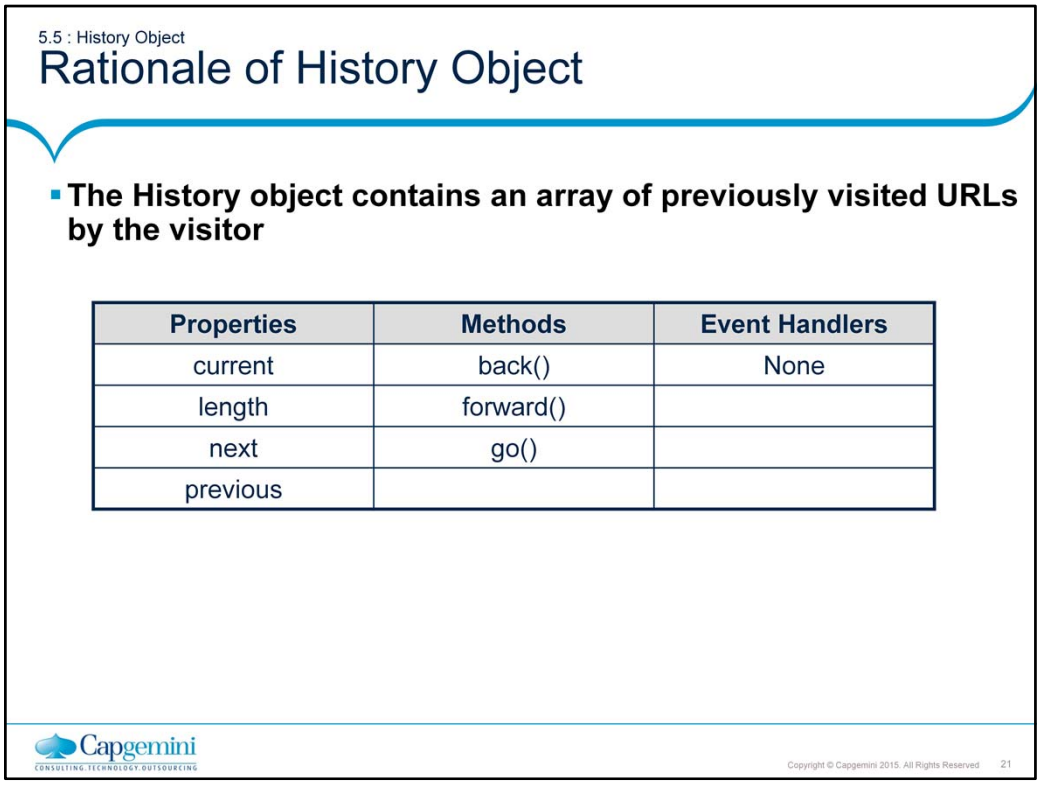

Working with Location Object:
Let us discuss some of the location object properties.

| Property | Description |
|---|---|
| protocol | The first component of any URL is the protocol being used for the particular type of communication. For eg: http,ftp,mailTo |
| hostname | The hostname of a typical URL is the name of the server on the network that stores the document you're viewing in the browser. |
| port | It retrieves the port number of the URL. |
| host | The property describes both the hostname and port of a URL. |
| pathname | The pathname component of a URL consists of the directory structure relative to the server's *root* volume. |
| hash | The hash property returns the anchor portion of a URL, including the hash symbol(#). |
| href | The location.href property supplies a string of the entire URL of the specified window object. |
| search | It accesses the query string of the URL. |




Working with Location Object:
Location Object Methods:
Let us discuss some Location Object Methods:

| Property | Description |
|---|---|
| assign("URL") | Just as you navigate to another page by assigning a new URL to the location object or location.href property, there also exists a method, location.assign(), that does the same task. |
| reload(uncGet) | The reload() method performs what is known as a *conditional-GET,* which means that the file is retrieved from the server or the browser's cache according to the cache preferences in the browser. If your page must perform an *unconditional-GET* to retrieve continually updated server or CGI-based data, then add a true parameter to the reload() method |
| replace("URL") | In a complex Web site, you may have pages that you do not want to appear in the user's history list. You cannot prevent a document from appearing in the history list (visible in the Go menu) while the user is looking at that page. However, you can instruct the browser to load another document into that window and replace the current history entry with the entry for the new document. |




Working with History Object:
As a user surfs the Web, the browser maintains a list of URLs for the most recent stops. This list is represented in JavaScript by the history object.

| Property | Description |
|---|---|
| current<br>next<br>previous | It retrieves the current, next and previous URLs in the history list. |
| length | It retrieves the number of items in the history list. |
| **Methods** | **Description** |
| back()<br>forward() | It navigates backward and forward in the browser history list. |
| go(relativeNumber\|<br>“URLstring”) | It uses the history.go() method for navigating to a specific index or URL in the history list.<br>This “go” command only accepts items that already exist in the history listing, so you cannot use it in place of setting the window.location object to a brand-new URL. |

## Demo on Working with History Object

- main.html
- Next.html
- Next2.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

Some additional egs:-

History_back.html
History_forward.html
History_go.html
History_Property.html
Location_Assign_method.html
Location_property.html
Location_reload_method.html
Location_replace_method.html

## Lab

- Lab 6 :
- Working with Location Object.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23

## Summary

- Document Object Model is a interface that allows programs and scripts to dynamically access and update content, structure and style of documents
- Window object is the topmost object in the entire scheme. It has properties, methods and event handlers
- The history object has an array of history items having details of the URL's visited from within that window
- The Location object contains information about the current URL


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 24

## Review Questions

- Question 1: Closed property returns _____ if the window object is closed either by a script or by the user.
  - Option 1: 1
  - Option 2: True
  - Option 3: 0
- Question 2: An alert dialog box is a modal window that presents a message for users with a single OK button to dismiss it.
  - True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 25

## Review Questions (Contd..)

- Question 4: The _____ and appCodeName properties are simply the official name and the internal code name for the browser application.
  - Option 1: Appname
  - Option 2: appName
  - Option 3: applname

- Question 5: The _____ property supplies a string of the entire URL of the specified window object.
  - Option 1: location.href
  - Option 2: hostname
  - Option 3: hash
- Question 6: The _____ property describes both the hostname and port of a URL.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 26

# Web Basics-JavaScript

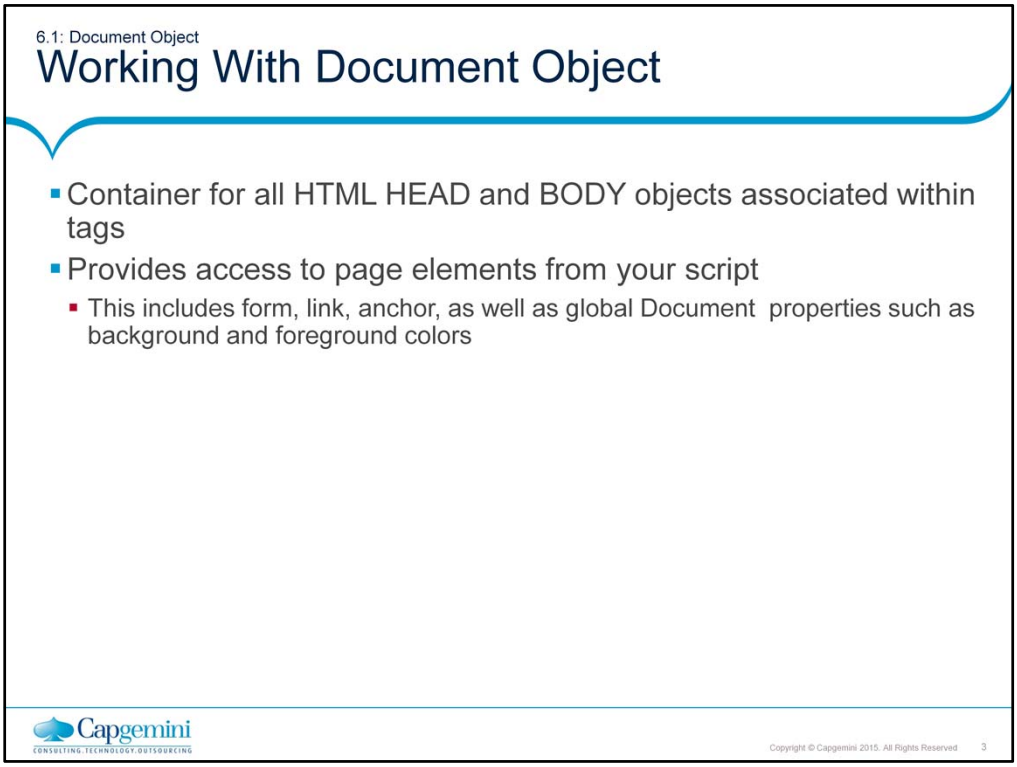
Lesson 6: Working With Document Object

# Lesson Objectives

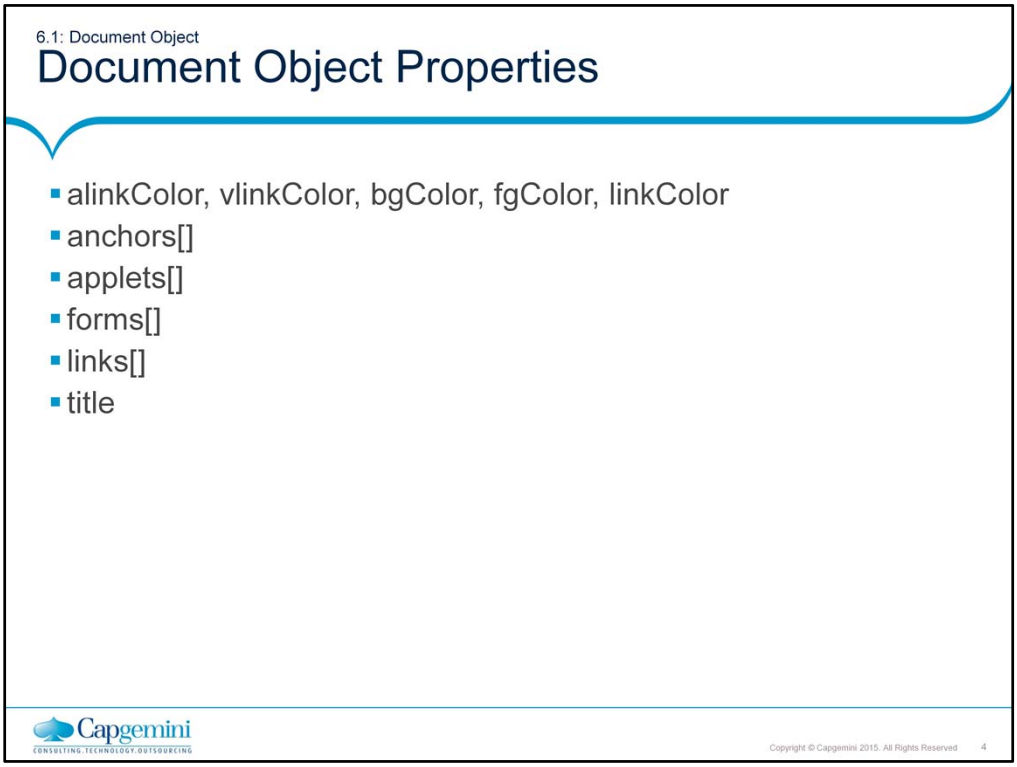
- To understand the following topics:
  - Document Object and its properties and methods
  - Cookies object

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

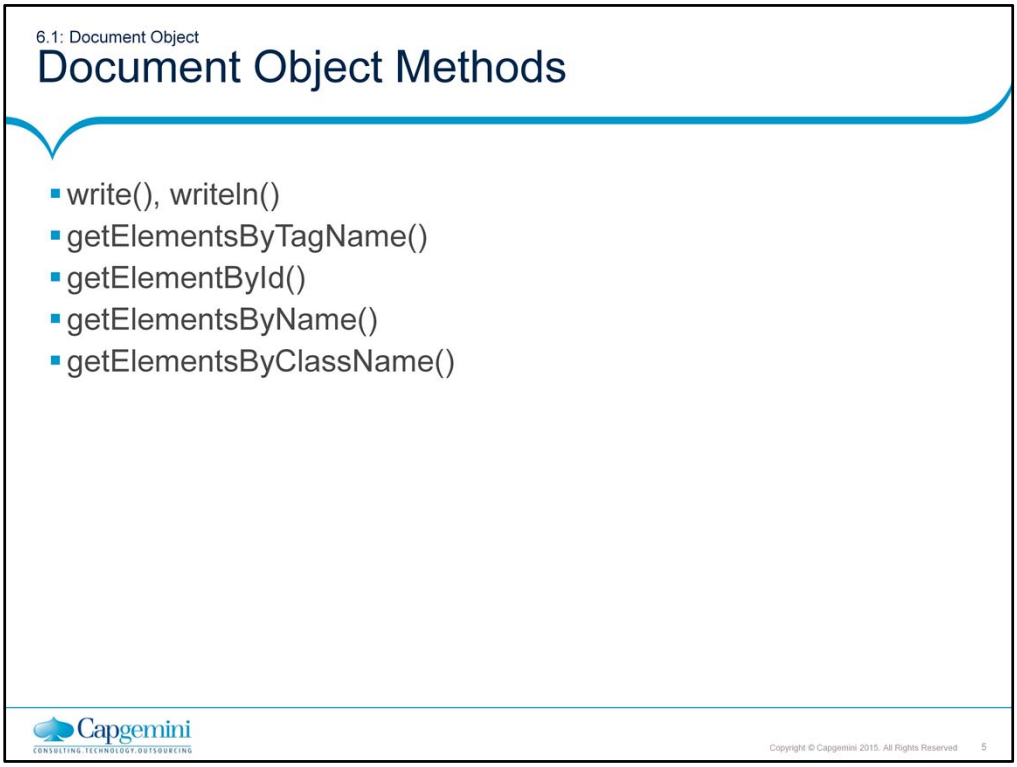
 2




Document object is part of the Window object. It is used to access all elements in a page. It provides access to the elements in an HTML page from within the script. This includes the properties of every form, link and anchor (and, where applicable, any sub-elements), as well as global document properties such as background and foreground colors.




| Property | Description |
|---|---|
| alinkColor<br>vlinkColor<br>bgColor<br>fgColor<br>linkColor | Get and set the properties of document – activated link, visited link, background color, foreground color (text) and hyperlink color. |
| anchors[],<br>forms[],<br>links[] | These properties retrieve array of values respectively as present in the document object |
| title | Gets the title of the document which occurs between the TITLE tags. |



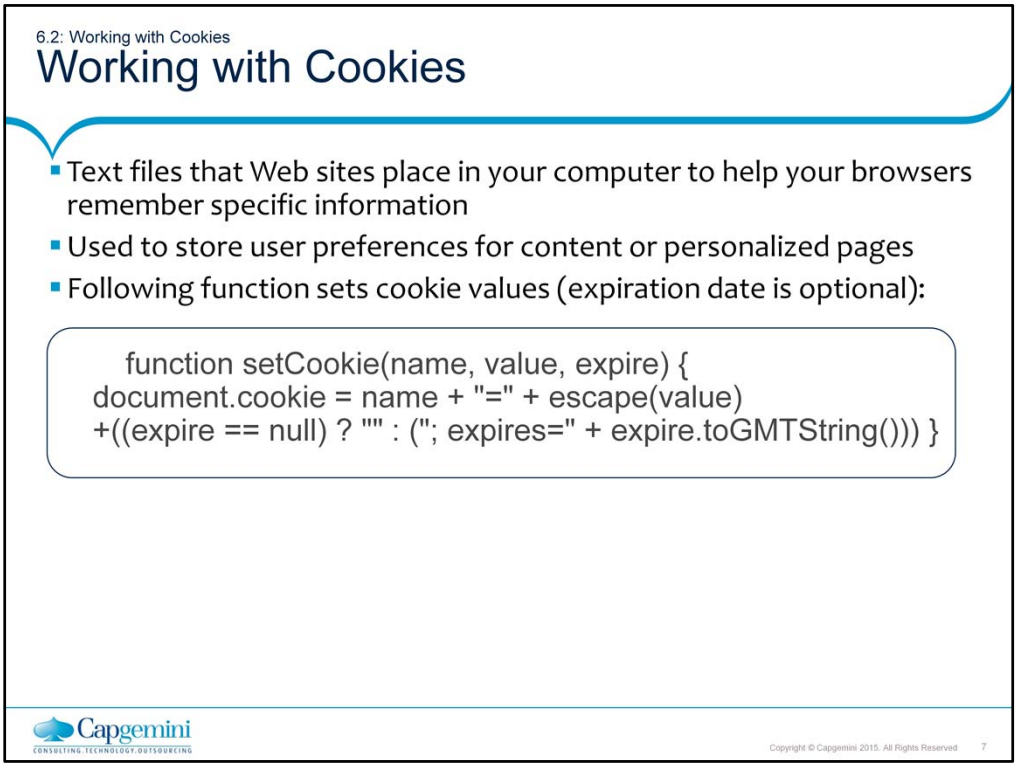

| Property | Description |
|---|---|
| write("string1", …)<br>writeln("string1", ..) | Both of these methods send text to a document for display in its window. The only difference between the two methods is that *document.writeln()* appends a carriage return to the end of the string it sends to the document (you must still write a <BR> to insert a line break). |
| getElementById("#para1") | This method locates the element whose id has been passed. The text within this element can then be accessed using properties innerHTML or innerText |
| getElementsByTagName("p") | This method locates all the elements which match the tagname passed. Each element of this type of tag can then be accessed in an array like manner |
| getElementsByName() | This method locates all the elements which match the name passed. Same name to many elements is usually given for radio buttons. |
| getElementsByClass() | This method locates all the elements which match the class name passed. |

## Demo

- Link_Anchor_object.html
- Meta_information.html
- locate_element_by_id.html
- locate_elements_by_tagname.html
- locate_elements_by_name.html
- locate_element_by_class_name.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6




Using Cookies

Cookies are a mechanism for storing persistent data on the client in a file called cookies.txt. Because HyperText Transport Protocol (HTTP) is a stateless protocol, cookies provide a way to maintain information between client requests. This section discusses basic uses of cookies and illustrates with a simple example.

Each cookie is a small item of information with an optional expiration date and is added to the cookie file in the following format:

name=value;expires=expDate;

Name is the name of the datum being stored, and value is its value. If name and value contain any semicolon, comma, or blank (space) characters, you must use the escape and unescape functions to encode and decode them respectively.

expDate is the expiration date, in GMT date format:

Wdy, DD-Mon-YY HH:MM:SS GMT

Although it is slightly different from this format, the date string returned by the Date method toGMTString can be used to set cookie expiration dates.

The expiration date is an optional parameter indicating how long to maintain the cookie. If expDate is not specified, the cookie expires when the user exits the current browser session. Browser maintains and retrieves a cookie only if its expiration date has not yet passed.

Limitations

Cookies have these limitations:

300 total cookies in the cookie file.

4 Kbytes per cookie, for the sum of both the cookie's name and value.

20 cookies per server or domain (completely specified hosts and domains are treated as separate entities and have a 20-cookie limitation for each, not combined).

Cookies can be associated with one or more directories. If your files are all in one directory, then you need not worry about this. If your files are in multiple directories, you may need to use an additional path parameter for each cookie.

Using Cookies with JavaScript

The *document.cookie* property is a string that contains all names and values of Navigator cookies. Use this property to work with cookies in JavaScript.

## Demo

- Ch9_ex1.cookie.html
- Demo_Cookies.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

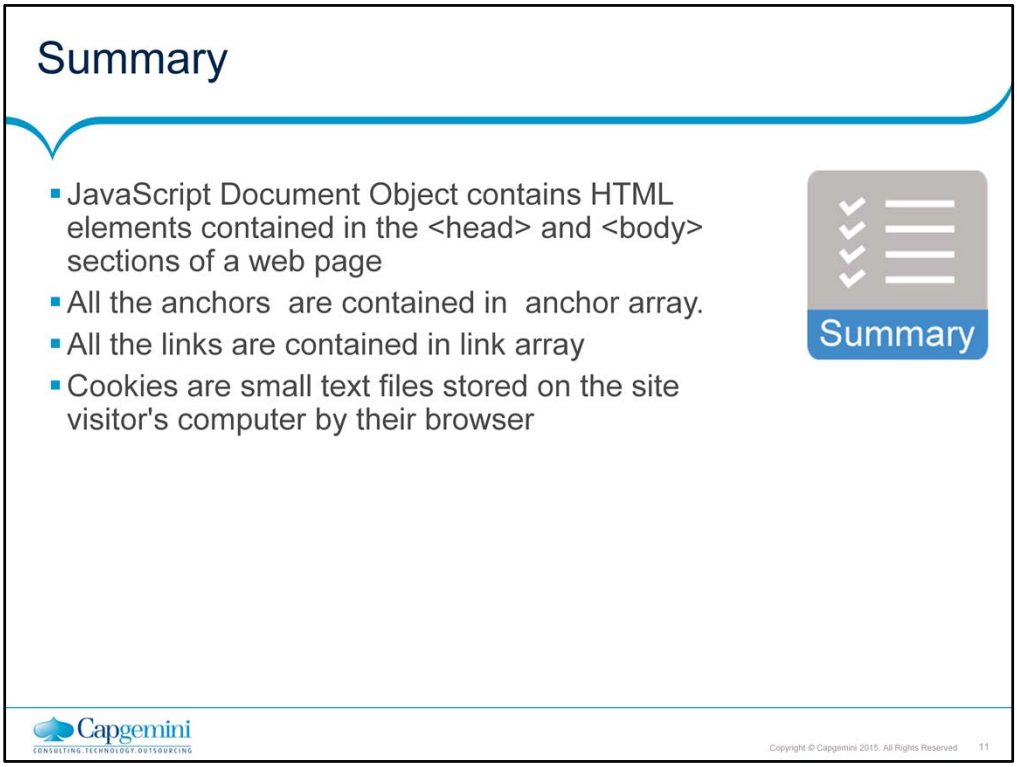
 9

# Lab

- Lab 7 :
- Working with Document object.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10




Summary

In this chapter, you understood:

- DOM structure
- How to work with Document Object
- How to work with cookies

## Review Questions

- Question 1: The _____ is the container for all HTML HEAD and BODY objects.
  - Option 1: Document
  - Option 2: Object
  - Option 3: Container

- Question 2: _____ property in document object retrieves an indexed array of anchors in a document.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

# Web Basics-JavaScript

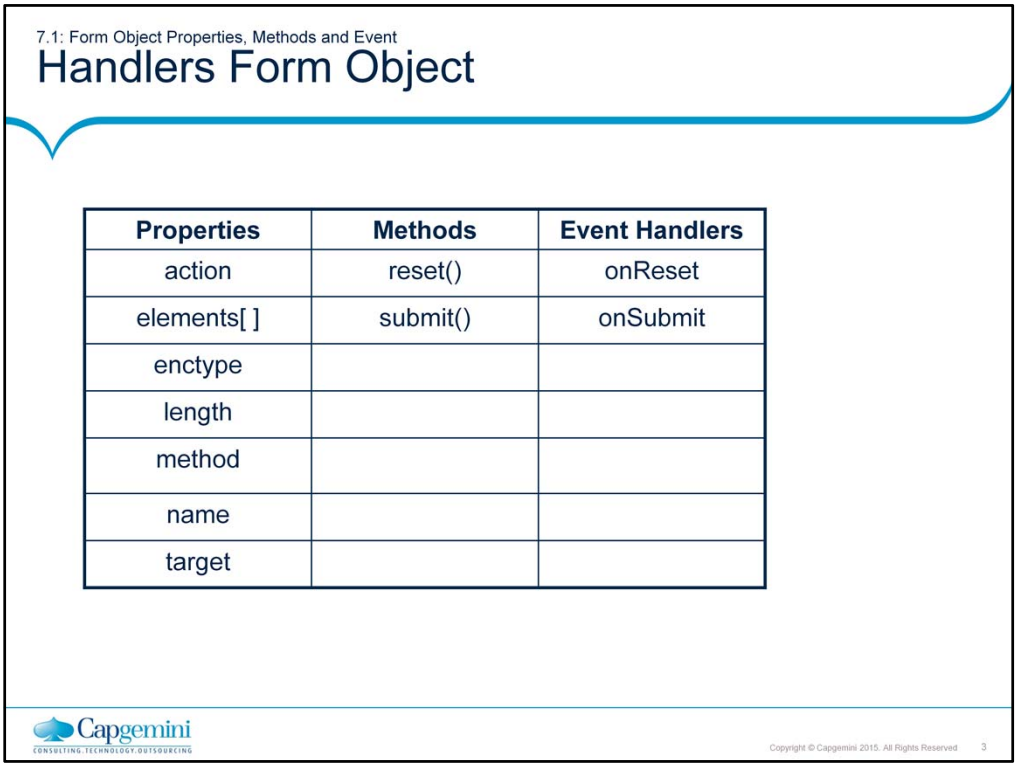
Lesson 7: Working with Form Object

# Lesson Objectives

- To understand the following topics:
  - Form Object Properties, Methods & Event Handlers
  - Text-Related Objects
  - Button Objects
  - Check Box and Radio Objects
  - Select Objects

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

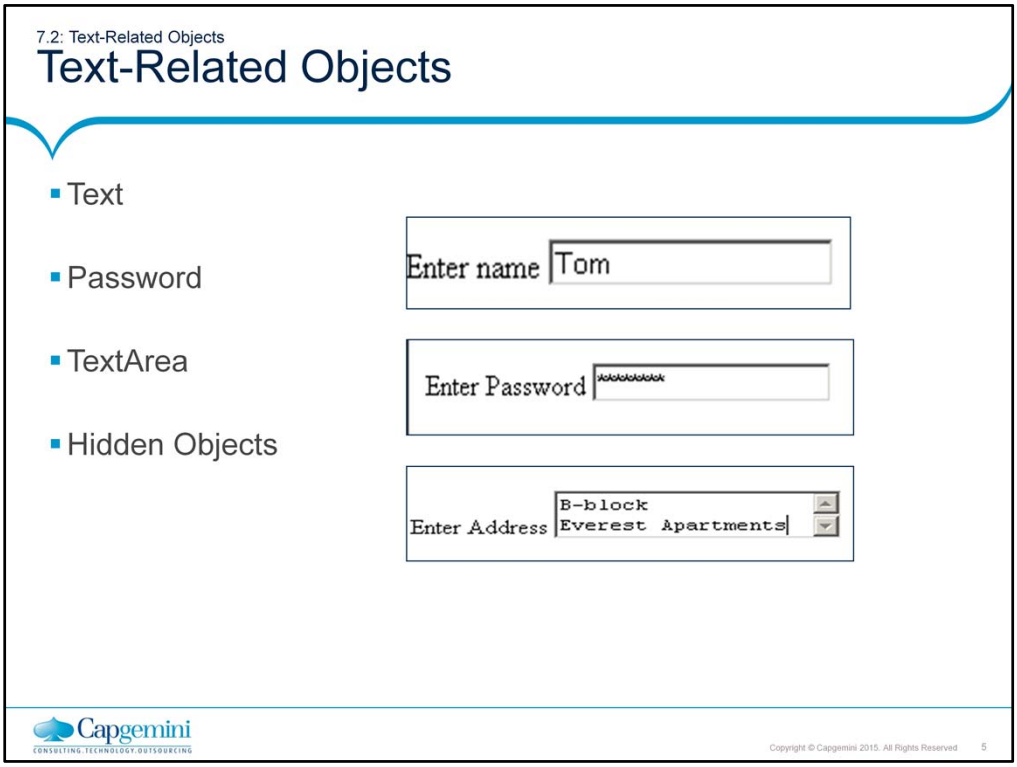



Working with Form Objects: Form Object Properties:
A form element provides the only way that users can enter textual information or make a selection from a predetermined set of choices, whether those choices appear in the form of an on/off checkbox, one of a set of mutually exclusive radio buttons, or a selection from a list.

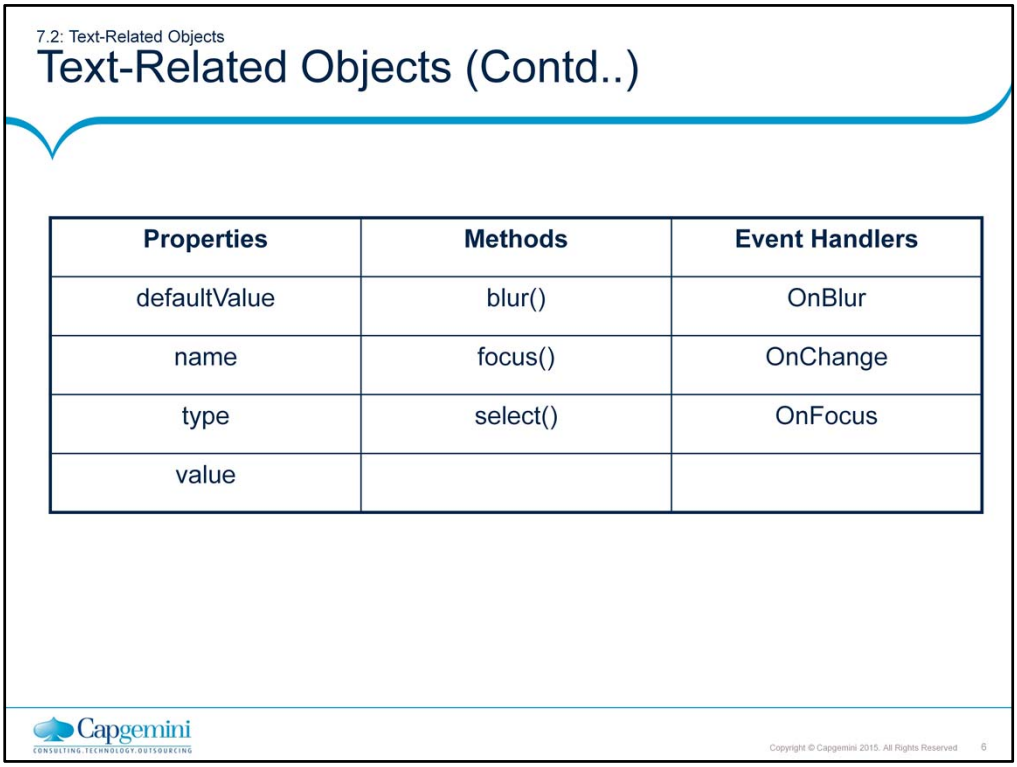
| Property/ Method/ Events | Description |
|---|---|
| action | This property is the same as the value you assign to the ACTION attribute of a <FORM> tag. The value is typically a URL on the server where queries or postings are sent for submission. |
| elements[] | Returns an array of elements. It includes all the user interface elements defined for a form: text fields, buttons, radio buttons, checkboxes, selection lists, and more. |
| encoding | You can define a form to alert a server that the data being submitted is in a MIME type. This property reflects the setting of the ENCTYPE attribute in the form definition. The default value is an empty string. |
| method | A form's method property is either the GET or POST values assigned to the METHOD attribute in a <FORM> tag. |

| | |
|---|---|
| name | Assigning a name to a form via the NAME attribute is optional but highly recommended when your scripts need to reference a form or its elements. This attribute's value is retrievable as the name property of a form. |
| target | The purpose of the TARGET attribute of a <FORM> definition is to enable you to specify where the output from the server's query should be displayed. The value of the target property is the name of the window or frame. |
| reset() | If you want to clear the form i.e return the form elements to its default settings using script control, you must do so by invoking the reset() method for the form. |
| submit() | Invoking this method is almost the same as a user clicking a form's Submit button |
| onReset | Immediately before a Reset button returns a form to its default settings, JavaScript sends a reset event to the form. By including an onReset event handler in the form definition, you can trap that event before the reset takes place. |
| onSubmit | When you define an onSubmit handler as an attribute of a <FORM> definition, JavaScript sends the submit event to the form just before it dashes off the data to the server. Therefore, any script or function that is the parameter of the onSubmit attribute executes before the data is actually submitted. Note that this event handler fires only in response to a genuine Submit-style button, and not from a form.submit() method. |

Table 9.1 Form object properties, methods and event handlers




Text-Related Objects:
Text Objects : The text object is the primary medium for capturing user-entered text.
Password Object: A password-style field looks like a text object, but when the user types something into the field, only asterisks or bullets (depending on your operating system) appears in the field.
Textarea Object: A textarea object closely resembles a text object, except for attributes that define its physical appearance on the page.
Hidden object: A hidden object is a simple string holder within a form object whose contents are not visible to the user of your Web page. With no methods or event handlers, the hidden object's value to your scripting is as a delivery vehicle for strings that your scripts need for reference values or other hard-wired data.

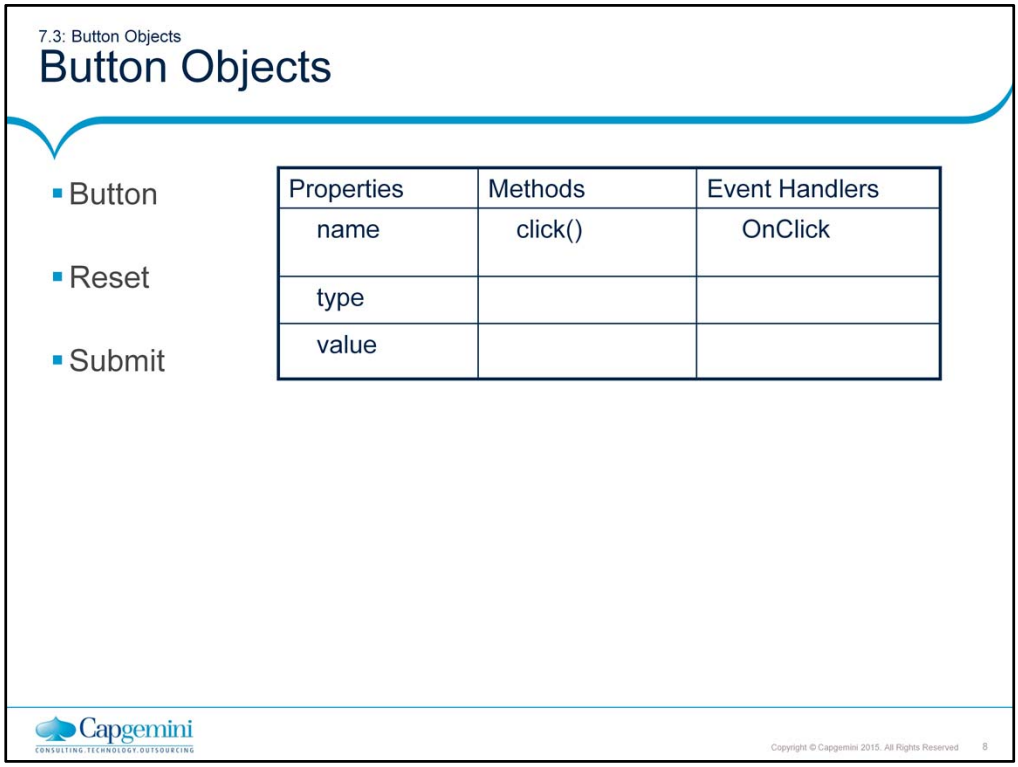



The properties, methods and event handlers are same for text object, text area and Password. For hidden object the properties are same but no methods and event handlers are associated with this object.

| Property/ Events/ Methods | Description |
|---|---|
| defaultValue | Specifies or returns a defaultValue for a text related objects. |
| name | This property can be used to reference the text object in the script. |
| type | Returns the type of text related object |
| value | A reference to an object's value property returns the string currently displayed in the field. |

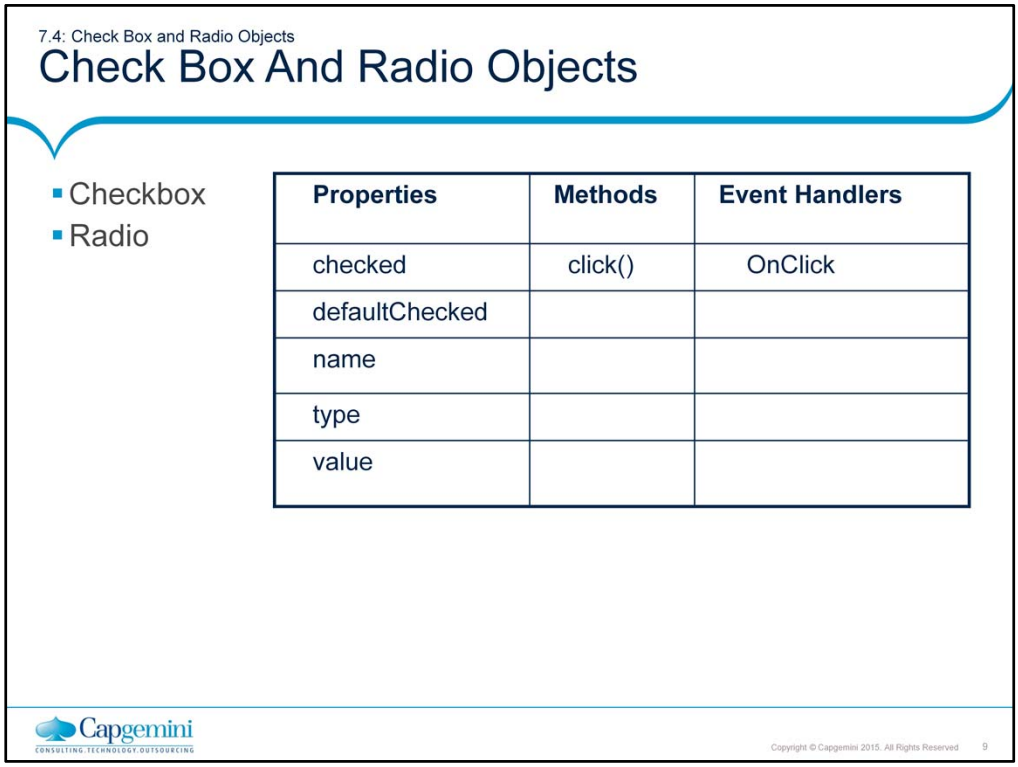
| | |
|---|---|
| blur() | blur() deselects whatever may be selected in the field, and the text insertion pointer leaves the field. The pointer does not proceed to the next field in tabbing order, as it does if you perform a blur by tabbing out of the field manually. |
| focus() | For a text object, having focus means that the text insertion pointer is flashing in that text object's field. The cursor usually appears at the beginning of the text. To prepare a field for entry to remove the existing text, use both the focus() and select() methods. |
| select() | Selecting a field under script control means selecting all text within the text object. |
| onBlur<br>onFocus | The onBlur event is fired when a text field loses focus because user has clicked somewhere outside the text field. The onFocus event is fired when the user clicks inside the text field. |
| onChange | This event is fired when the user changes the value in the text field. |

Refer to Appendix for more event handlers




Button Objects: Button, Submit and Reset

| Property | Description |
|---|---|
| name | You may need to retrieve this property in a general-purpose function handler called by multiple buttons in a document. The function can test for a button name and perform the necessary statements for that button. |
| type | The precise value of the type property echoes the setting of the TYPE attribute of the <INPUT> tag that defined the object: button; submit; or reset. |
| value | A button's visible label is determined by the VALUE property. |
| click() | A button's click() method should replicate, via scripting, the human action of clicking that button. |
| onClick | Virtually all button action takes place in response to the onClick event handler. A click is defined as a press and release of the mouse button while the screen pointer rests atop the button. |

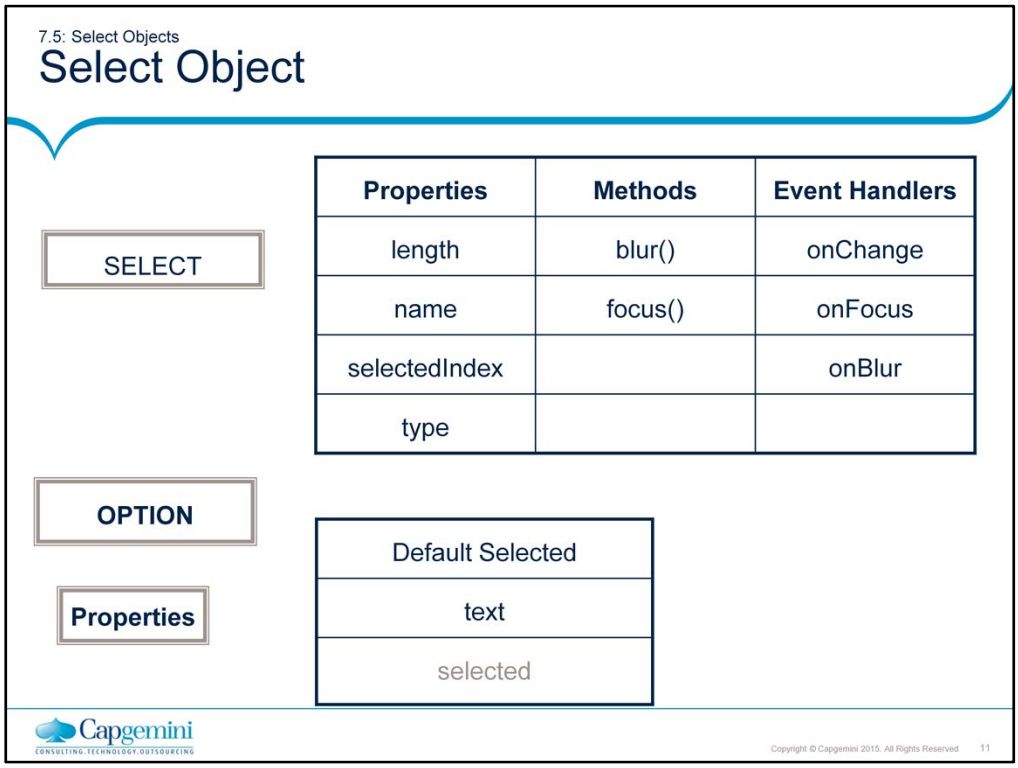



Checkbox object:

| Property/ Events/ Methods | Description |
|---|---|
| checked | The simplest property of a checkbox gets or lets you set whether or not a checkbox is checked. The value is true for a checked box and false for an unchecked box.<br>Only one radio button in a group can be highlighted checked) at a time. That one button's checked property is set to true, whereas all others in the group are set to false. |
| defaultChecked | If you add the CHECKED attribute to the <INPUT> definition for a checkbox or radio button, the defaultChecked property for that object is true; otherwise, false. |
| name | The name property allows user to access name for the checkbox or radio button through script. |
| type | Use the type property to help you identify a checkbox object or a radio button object from an unknown group of form elements. |

| | |
|---|---|
| value | A checkbox or radio button object's value property is a string of any text you want to associate with. Either you can set or retrieve the value |
| click() | The intention of the click() method is to enact, via script, the physical act of checking a checkbox or selecting a radio button |
| onClick | The onClick event of checkboxes or radiobuttons should be handled when through script you need to handle a specific task |

Table 9.4 Checkbox object properties, methods and event handlers




| Property/ Methods/ Events | Description |
|---|---|
| length | Returns the number of items available in the list. A select object with three choices in it has a length property of 3. |
| Name | A select object's name property is the string you assign to the object by way of its NAME attribute in the object's <SELECT> tag which can be |
| selectedIndex | When a user clicks on a choice in a selection list, the selectedIndex property changes to a number corresponding to that item in the list. |
| type | Use the type property to help you identify a select object from an unknown group of form elements. |
| blur()<br>focus() | Your scripts can bring focus to a select object by invoking the object's focus() method. To remove focus from an object, invoke its blur() method. These methods work identically with their counterparts in the text object. |
| onChange | As a user clicks on a new choice in a select object, the object receives a change event that can be captured by the onChange event handler. |

| | |
|---|---|
| options[index]. defaultSelected | If your select object definition includes one option whose SELECTED attribute is included, that option's defaultSelected property is set to true. The defaultSelected property for all other options is false. |
| options[index]. selected | To determine which option a user has selected from a list than looping through all options and examining the selected property this property can be used. |
| options[index]. text | The text property of an option is the text of the item as it appears in the list. |

Refer to Appendix for some more properties

Using ‘this’ keyword
The ‘this’ keyword can be used to reference the object which called the function. It can be used within a function scope or global scope and it receives a different value in each scope. Depending on which object has called the function the value of ‘this’ will differ. The ‘this’ keyword always points to the object that is calling a particular method.
Consider the example given below:

The ‘this’ keyword is used in the showColor() function of an object. In this context, this is equal to car, making this code functionality equivalent to the following code snippet

```
var oCar = new Object;
oCar.color = “red”;
oCar.showColor = function () {
    alert(this.color); //outputs “red”
};
```

So the reason for using ‘this’ is you never know what kind of variable names you will use to instantiate an object. By using ‘this’ you are sure to invoke the correct function with the correct value. Also it allows you to use the same function any number of times. To understand this consider the following code:

```
var oCar = new Object;
oCar.color = “red”;
oCar.showColor = function () {
    alert(oCar.color); //outputs “red”
};
```

In the above snippet both oCar1 and oCar2 refer to the same function. The function gives the output according to the object which called the function.

```
function showColor() {
        alert(this.color);
}
var oCar1 = new Object;
oCar1.color = “red”;
oCar1.showColor = showColor;
var oCar2 = new Object;
oCar2.color = “blue”;
oCar2.showColor = showColor;
oCar1.showColor(); //outputs “red”
oCar2.showColor(); //outputs “blue”
```

## Demo

- Form_Object.html
- Select_option.html
- Element_array.html
- Enctype.html
- Hidden_value.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

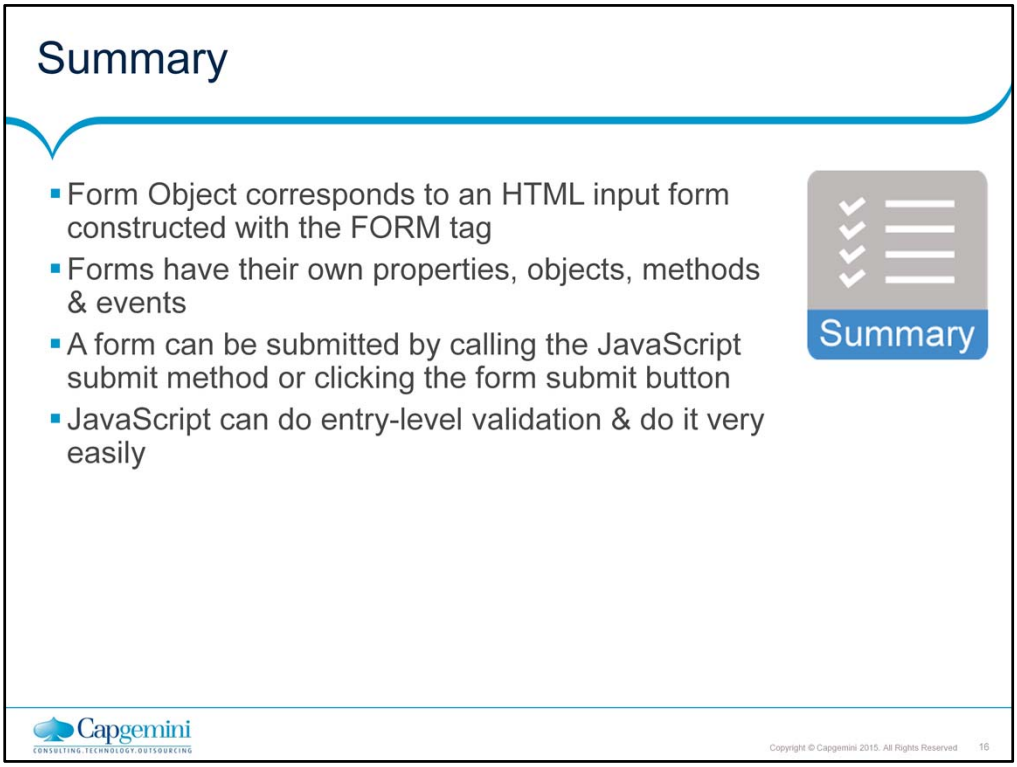
 

# Lab

- Lab 8:
- Working with Form Object

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15




Summary
This module provided an understanding of:
Form object and its components.
How to create form objects.
How to handle events.
How to validate data.
How to submit a form.

# Review Questions

- Question 1: A form's _____ property is either the GET or POST values assigned to the METHOD attribute in a <FORM> definition.
  - Option 1: Method
  - Option 2: Class
  - Option 3: Object
- Question 2: The intention of the click() method is to enact, via a script, the physical act of clicking a radio button.
  - True / False
- Question 3: A button's _____method should replicate, via scripting, the human action of clicking that button.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Web Basics-JavaScript

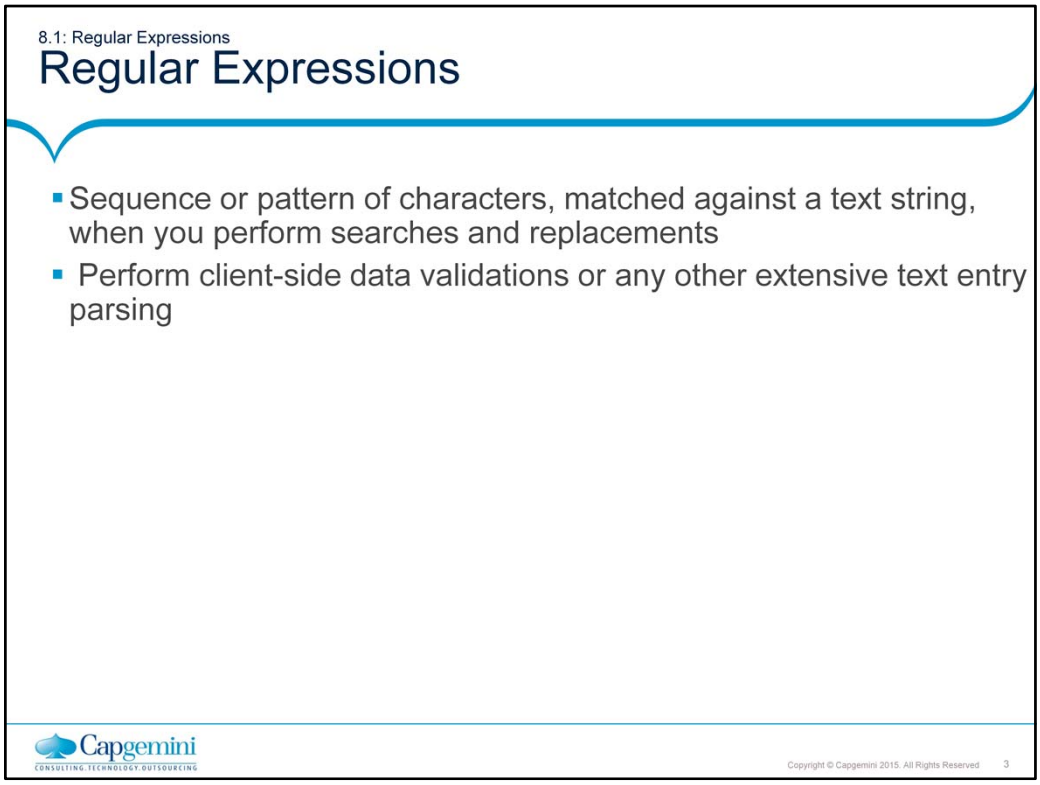
Lesson 8.Working With Regular Expressions

## Lesson Objectives

- After completing this lesson, you will be able to:
  - Use regular expressions
  - Search text using simple patterns and special characters
- Work with RegExp objects

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2




Working with Regular Expressions

If your scripts perform client-side data validations or any other extensive text entry parsing, then you can use regular expressions, rather than cobbling together comparatively complex JavaScript functions to perform the same tasks.

JavaScript treats regular expressions as objects and distinguishes between them and the *RegExp* constructor.

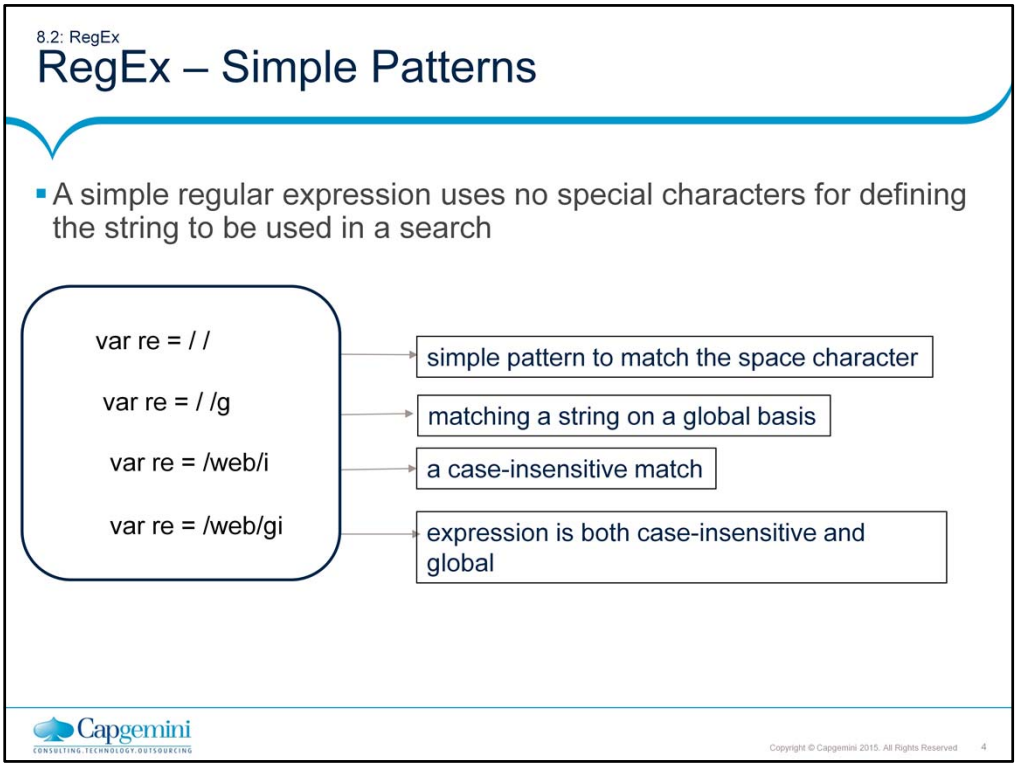
To cover the depth of the regular expression syntax, we need to study the following:

Simple expressions

Range of special characters used to define specifications for search strings

Introduction to the usage of parentheses in the language:

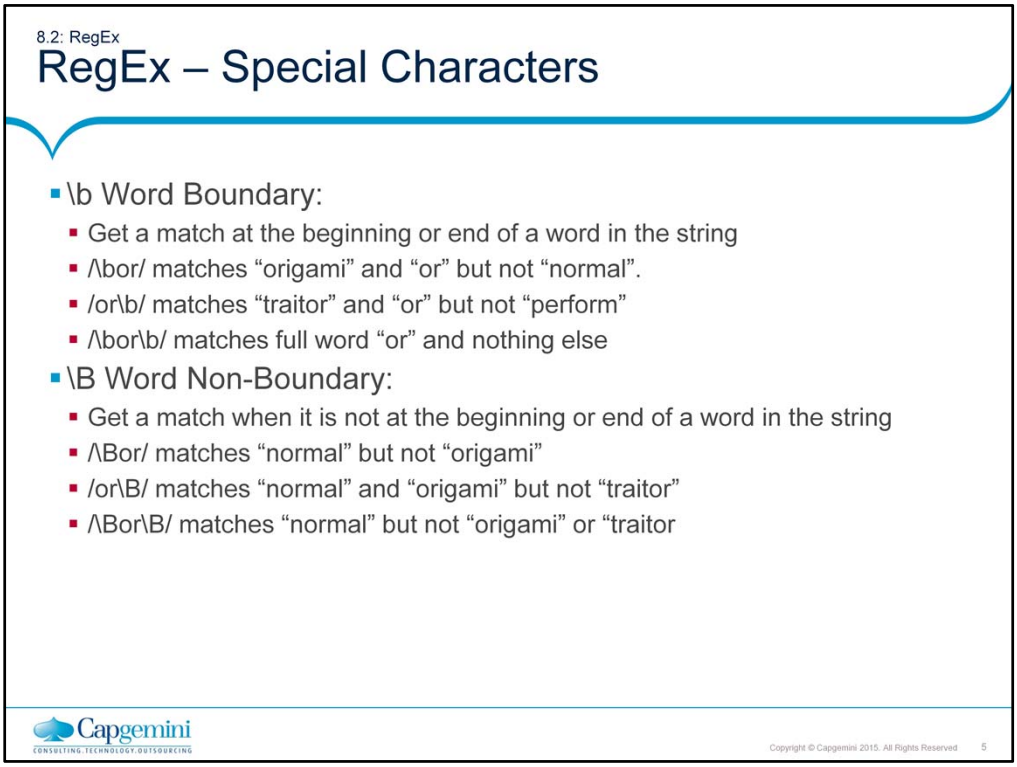
- Group expressions to influence calculation precedence
- Temporarily store intermediate results of more complex expressions for use in reconstructing strings after their dissection by the regular expression.




Simple Patterns
A simple regular expression uses no special characters to define the string to use in a search. Therefore, if you wish to replace every space in a string with an underscore character, the simple pattern to match the space character is: var re = / /

A space appears between the regular expression start-end forward slashes. The problem with this expression, however, is that it knows only how to find a single instance of a space in a long string. Regular expressions can be instructed to apply the matching string on a global basis by appending the g modifier: var re = / /g

Regular expression matching — like a lot of other aspects of JavaScript — is case-sensitive. But you can override this behavior by using one other modifier that lets you specify a case-insensitive match. Therefore, the following expression, var re = /web/I, finds a match for “web,” “Web,” or any combination of upper and lowercase letters in the word. You can combine the two modifiers together at the end of a regular expression. For example, the following expression is both case-insensitive and global in scope: var re = /web/gi




Special Characters

The regular expression in JavaScript borrows most of its vocabulary from the Perl regular expression. In a few instances, JavaScript offers alternatives to simplify the syntax, and accepts their Perl version for developers with experience in that technology.

8.2: RegEx

# RegEx – Special Characters (Contd.)

- \d Numeral:
  - Find any single digit 0 through 9
    - /\d\d\d/ matches “212” and “415” but not “B17”
- \D Non-numeral:
  - Find any non-digit
    - /\D\D\D/ matches “ABC” but not “212” or “B17”
- \s Single White Space:
  - Find any single space character
    - /over\sbite/ matches “over bite” but not “overbite” or “over  bite”

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

8.2: RegEx

## RegEx – Special Characters (Contd.)

- \S Single Non-White Space:
  - /over\Sbite/ matches “over-bite” but not “overbite” or “over bite”
- \w Letter, Numeral, or Underscore:
  - /A\w/ matches “A1” and “AA” but not “A+”
- \W Not letter, Numeral, or Underscore:
  - /A\W/ matches “A+” but not “A1” and “AA”

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

8.2: RegEx

## RegEx – Special Characters (Contd.)

- "." Any Character Except Newline:
  - /…/ matches "ABC", "1+3", "A 3" or any 3 characters
- […] Character Set:
  - Finds any character in the specified character set
    - /[AN]BC/ matches "ABC" and "NBC"
- [^…] Negated Character Set:
  - Find any character not in the specified character set
    - /[^AN]BC/ matches "BBC" and "CBC" but not "ABC" or "NBC"

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

8.2: RegEx

## RegEx – Counting Metacharacters

- "*" - Zero or More Times:
  - /Ja*vaScript/ matches "JvaScript", "JavaScript", and "JaaavaScript" but not "JovaScript"
- "?" - Zero or One Time:
  - /Ja?vaScript/ matches "JvaScript" or "JavaScript" but not "JaaavaScript"
- "+" - One or More Times:
  - /Ja+vaScript/ matches "JavaScript" or "JaavaScript" but not "JvaScript"

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

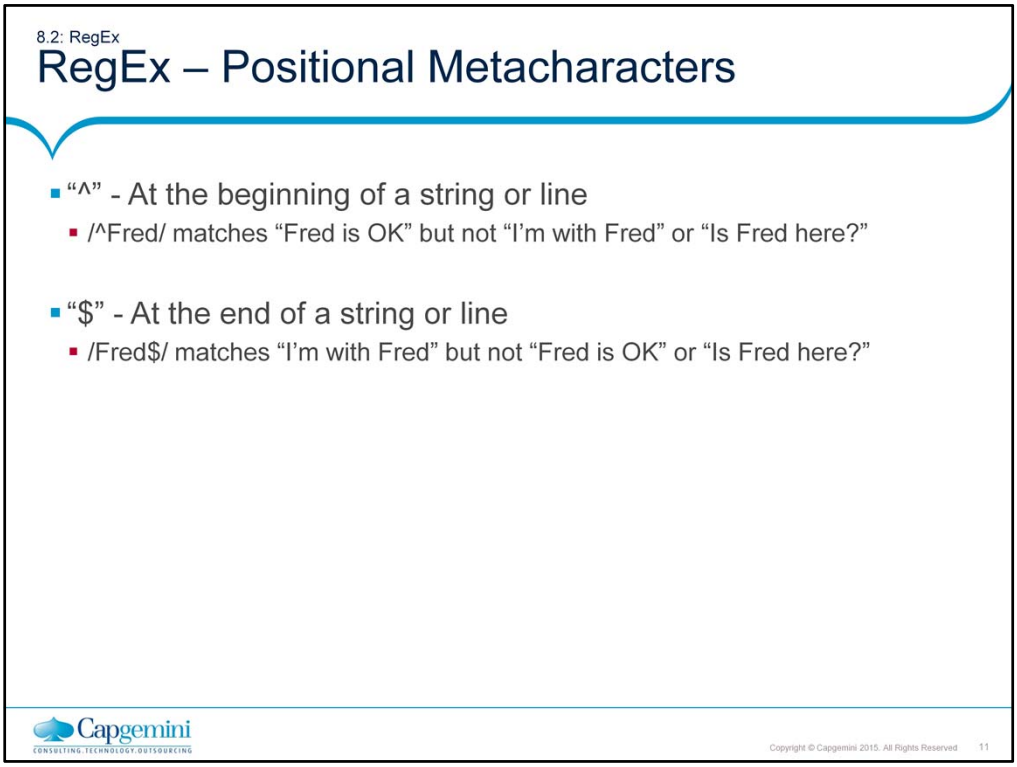
8.2: RegEx

## RegEx – Counting Metacharacters (Contd.)

- {n} - Exactly n Times:
  - /Ja{2}vaScript/ matches “JaavaScript” but not “JvaScript” or “JavaScript”
- {n,} - N or More Times:
  - /Ja{2,}vaScript/ matches “JaavaScript” or “JaaavaScript” but not “JavaScript”
- {n,m} - At Least n, At Most m Times:
  - /Ja{2,3}vaScript/ matches “JaavaScript” or “JaaavaScript” but not “JavaScript”

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10




Every metacharacter applies to the character immediately preceding it in the regular expression. Preceding characters might also be matching *metacharacters*. For example, a match occurs for the following expression if the string contains two digits separated by one or more vowels:

```
/\d[aeiouy]+\d/
```

The last major contribution of metacharacters is to help regular expressions search a particular position in a string. Following table shows positional metacharacters:

For example, you might want to make sure that a match for a roman numeral is found only when it is at the start of a line, rather than when it is used inline somewhere else. If the document contains roman numerals in an outline, you can match all the top-level items that are flush left with the document with a regular expression like the following:

```
/^[IVXMDCL]+\./
```

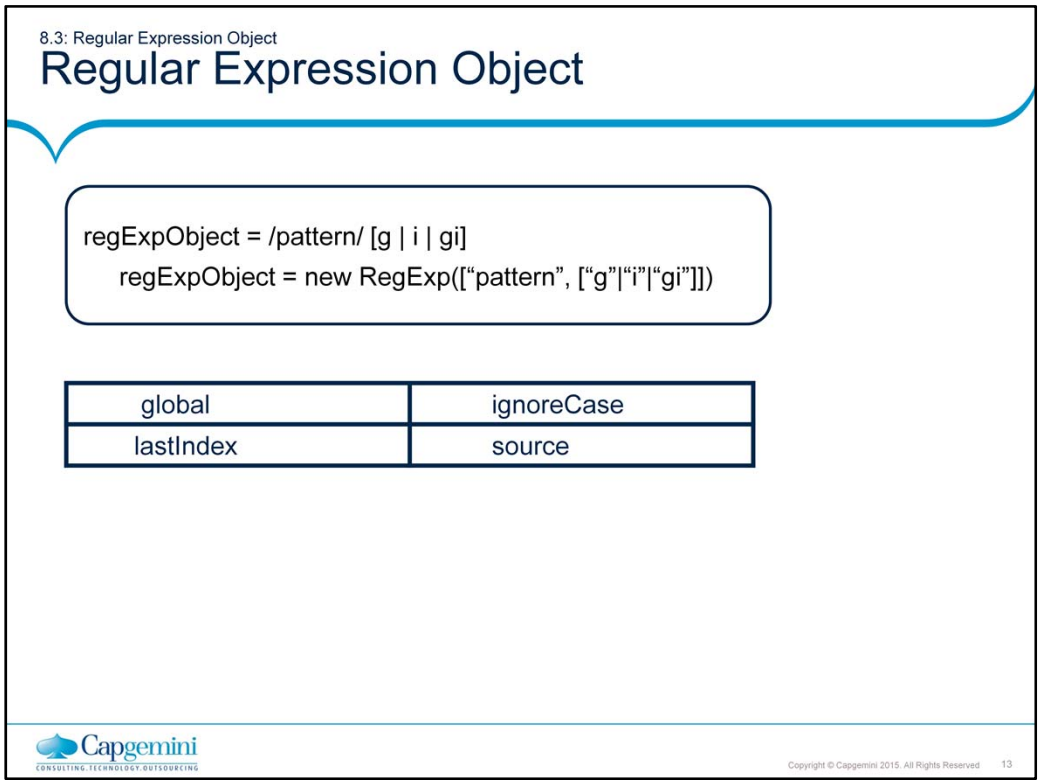
This expression matches any combination of roman numeral characters followed by a period (the period is a special character in regular expressions, as shown in Table 10-1, so you have to escape it to offer it as a character), provided the roman numeral is at the beginning of a line and has no tabs or spaces before it. There would also not be a match in a line that contains, say, the phrase “see Part IV” as the roman numeral is not at the beginning of the line.

### Grouping and Backreferencing

Regular expressions obey most of the JavaScript operator precedence laws with regards to grouping by parentheses and the logical *OR* operator. One difference is that the regular expression’s OR operator is a *single-pipe* character (|) rather than JavaScript’s double-pipe character.

Parentheses have additional powers that go beyond influencing the precedence of calculation. Any set of parentheses (matched pair of *left* and *right* parenthesis) stores the results of a found match of the expression within them.

Parentheses can be nested inside one another. Storage is accomplished automatically, with data stored in an indexed array accessible to your scripts and to your regular expressions (although through different syntax). Access to these storage bins is known as *backreferencing*, because a regular expression can point backward to the result of an expression component earlier in the overall expression. These stored subcomponents come in handy for replace operations, as demonstrated later in this chapter.




**Regular Expression Object**

Each regular expression object contains its own pattern and other properties. To decide which object creation style to use depends on the way the regular expression is to be used in your scripts.

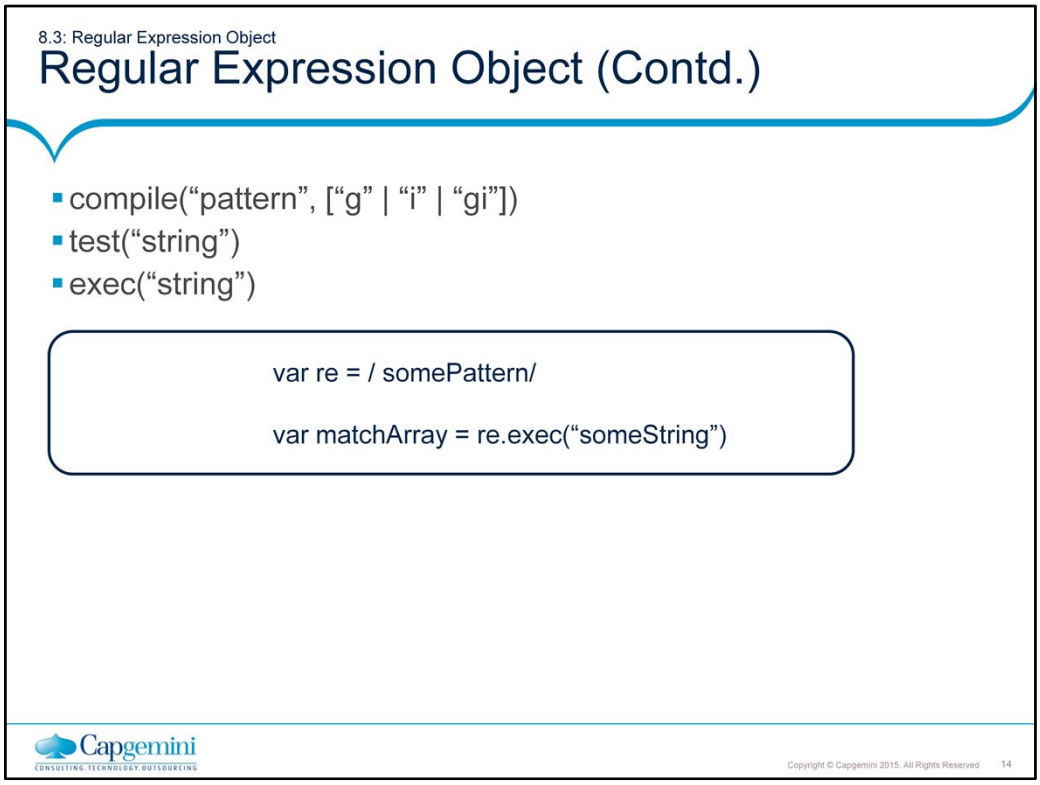
Syntax

Regular expression creation:

```
regularExpressionObject = / pattern/ [g | i | gi]
regularExpressionObject = new RegExp([“ pattern”, [“g”
| “i” | “gi”]])
```

Access Regular Expression Properties or Methods:

```
regularExpressionObject.property | method([ parameters])
```




Properties & Methods of Regular Expression object:
global : Specifies if the modifier “g” is set
ignoreCase : Specified if the modifier “I” is set
lastIndex : Specifies the index position from where to start the next match.
source **:** The source property is simply the string representation of the regular expression used to define the object. This property is read-only.

compile(“ pattern”, [“g” | “i” | “gi”])
Use the *compile()* method to compile on the fly, a regular expression whose content changes continually during script execution. Other regular expression creation statements (literal notation and the new *RegExp()* constructor passing a regular expression) automatically compile their expressions.

Methods

exec(“ string”)

Returns a matched array object or null. The *exec()* method examines the string passed as its parameter for at least one match of the specification defined for the regular expression object. The behavior of this method is similar to that of the *string.match()* method (although the *match()* method is more powerful in completing global matches). Typically, a call to the *exec()* method is made immediately after creating a regular expression object. Refer the following code:

```
var re = / somePattern/
var matchArray = re.exec(“ someString”)
```

Much happens as a result of the *exec()* method. Properties of both the regular expression object and window’s *RegExp* object are updated based on the success of the match. The method also returns an object that conveys additional data about the operation.

test(“string”)

This method returns Boolean  The most efficient way to find out if a regular expression has a match in a string is to use the *test()* method. Returned values are true if a match exists and false if not. In case you need more information, a companion method, *string.search(),* returns the starting index value of the matching string.

Refer to Appendix for additional properties.

## Demo

- Test_compiler.html
- DemoRegExp.html


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

Add the notes here.

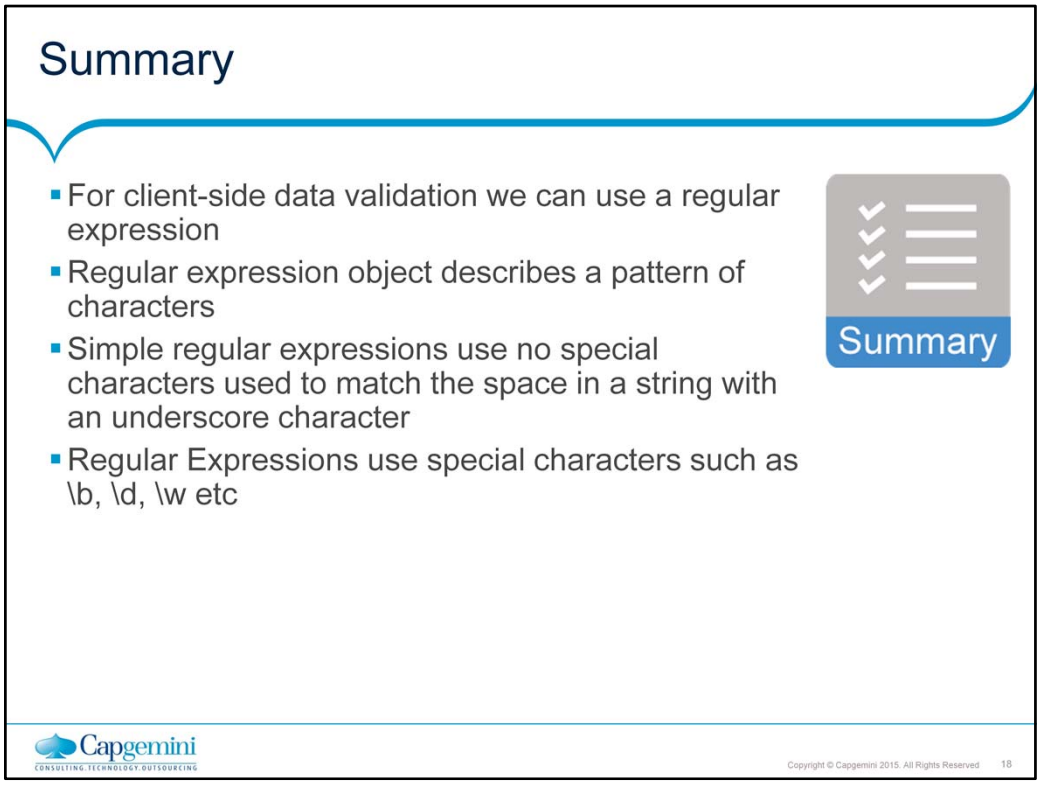
## Lab

- Lab Exercise 9:
  - Regular Expressions in JavaScript

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

Add the notes here.




From this chapter, you know how to:

Use Regular Expressions
Search using Simple patterns
Search using Special characters
Work with RegExp Objects

## Review Questions

- Question 1: The _____ property is the main string against which a regular expression is compared in search of a match.
  - Option 1: RegExp.input
  - Option 2: RegExp.inp
  - Option 3: RegExpr.input
- Question 2: Index property indicates the index counter of the main string to be searched against the current regular expression object.
  - True / False
- Question 3: Use the _____ method to compile on the fly a regular expression whose content changes continually during the execution of a script.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

## Match the Following

| | |
|---|---|
| 1 . \b | a. Word non-boundary |
| 2. \B | b. Word boundary |
| 3 . \d | c. Numeral |
| 4 . \s | d. Single non-white space |
| 5 . \S | e. Single white space |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 20

# Web Basics - JavaScript

# Lab Book

## Document Revision History

| Date | Revision No. | Author | Summary of Changes |
|---|---|---|---|
| 12-May-2009 | 1.0 | Pradnya Jagtap | Content Creation |
| 05-Oct-2009 | 1.1 | CLS Team | Review |
| 24-Jun-2010 | 2.0 | Anu Mitra | Refinements |
| 03-May-2011 | 3.0 | Karthik M/Anu Mitra | Integration Refinements |
| 21-Apr-2015 | 4.0 | Rathnajothi P | Revamp/Refinement |
| 15-May-2015 | 5.0 | Kavita D Arora | Integration Refinements |

## Table of Contents



## Getting Started

**Overview**

These Lab book is a guided tour for Learning JavaScript. It contains solved examples and To Do assignments. Follow the steps provided in the solved examples and then work out the 'to do' Assignments given.

**Setup Checklist for JavaScript**

Here is what is expected on your machine in order for the lab to work.

**Minimum System Requirements**

- Hardware: Networked PCs with minimum 64 MB RAM and 60 MB HDD.
- Software: Window based Operating System having the latest version of Internet Explorer (IE) or Netscape Navigator installed.

**Please ensure that the following is done:**

- A text editor like Notepad, Eclipse Luna or Visual Studio 2008 is installed.

**Instructions**

- For coding standards refer Appendix – A.
- All Lab assignments should follow the coding standards.
- Create a directory by your name in drive <drive> for JavaScript assignments.
- In this directory, create subdirectory javascript_assgn.
- For each lab create directory as lab<lab number>.

**Learning More (Bibliography if applicable)**

- Beginning JavaScript by Paul Wilton
- JavaScript: The Definitive Guide by David Flanagan
- JavaScript Application Cookbook by Jerry Bradenbaugh

## Lab 1:Basics Concepts of JavaScript

| | |
|---|---|
| **Goals** | • Learn to embed script tags in different parts of the HTML document. |
| **Time** | 120 minutes |

**1.1: Create a page to display "Welcome to JavaScript".**

**Solution:**

**Step 1:** Complete the following code and save it as prob1.html

```
<!DOCTYPE html>
<html>
<head>
<title> Welcome to JavaScript</title>
</head>
<body>
<script>
 document.write("Welcome to JavaScript - The Scripting Language")
</script>
</body>
</html>
```

**Example 1: Lab 1: Prob1.html**

**Step 2:** Start the editor to be used.

**Step 3:** Write the JavaScript program.

**Step 4:** Save the file with extension .html or htm.

**Step 5:** Select **Start → Programs → Internet Explorer.**

Alternatively select **Start → Programs → Netscape Navigator.**

**Step 6:** In the Internet Explorer, select **File → Open → Browse**, and select the file you have just saved.

**Step 7:** Click **OK** in the browser pop-up window.

**Step 8:** Verify that you get the output as shown in the figure given below.


**Figure 1: Welcome to JavaScript**

**Note:** Follow the above steps (3 - 8) for every Lab problem for verifying the output. You can also use other text editors like editplus, WordPad, MS Visual Interdev (if installed) to create your **html** and **.js** pages.

**1.2: Create prob2.html to display Formatted Hello World by using JavaScript by embedding Hello World in <H1> tag.**

**Solution:**

**Step 1:** Create **prob2.html** page to complete the following code and save in lab1 directory.

```
<html>
<head>
<title>Displaying Formatted Text using JavaScript</title>
</head>
<body>

<script>

//TODO:  Display hello world embedded in h1 tag with align attribute value right

</script>

</body>
</html>
```

**Example 2: Lab 1: Prob2.html**

**Step 2:** Open **prob2.html** page in the browser, and verify that you get the same output as required.

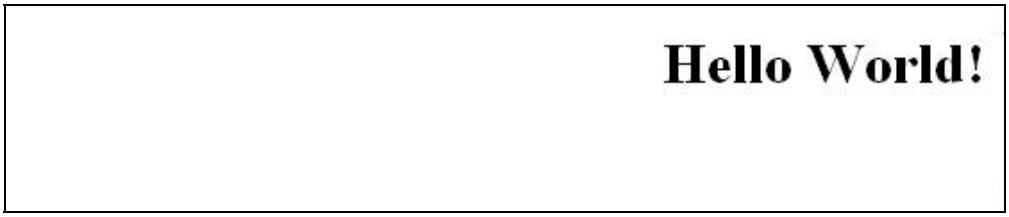



**Figure 2: Formatting Text in JavaScript**

### 1.3: Create page to show use of external JavaScript

**Solution:**

**Step 1:** Create **Prob3.html** to complete the following code and save it in lab1 directory.

```
<html>
<head><title>Using External Script file in HTML Document</title>

<script src="HelloWorld.js">
</script>

</head>
<body>
<hr>
<p>The actual script is in external script file called "HelloWorld.js"</p>



<script>
//TODO: Insert the code here to invoke the function sayHello() in the file HelloWorld.js
</script>

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

**Example 3: Lab 1: Prob3.html**

**Step 2:** Create a file **HelloWorld.js** which should have a function **sayHello()** that returns a string "Hello World".

```
function sayHello()
{
//TODO:return the string “Hello World“
}
```

**Example 4: Lab 1: HelloWorld.js**

**Step 3:** Open **prob3.html** page in the browser, and verify that you get the same output as required.

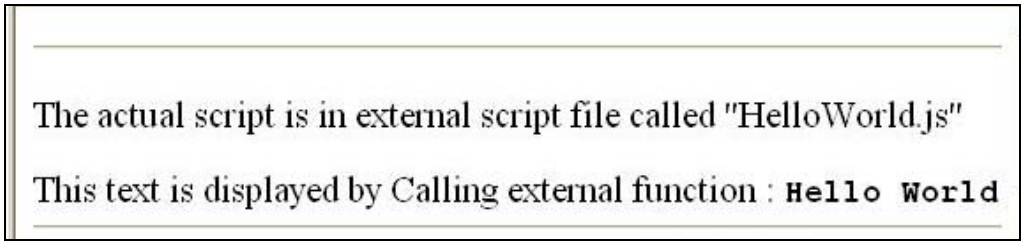



**Figure 3: Using external JavaScript File**

**Step 4:** Create **Prob4.html** page and complete the following code and save it in lab1 directory.

```
<html>
<head><title>Embedding Script tag in HTML Document</title>

<script>

//TODO:use write method in document object to display the desired output

</script>
<hr>
<script src="Hello.js">
</script>

</head>
<body>

<script>
//TODO:  use write method in document object to display desired the output
</script>

<hr>
```

```
<p><code>The actual script is in external script file called "Hello.js"</code></p>

<script>
//TODO:  Insert your code here to call the function dispHello() from the Hello.js file
</script>

<hr>

</body>
</html>
```

**Figure 4: Lab 1: Prob4.html**

**Step 5:** Create a file **Hello.js** which should have a function **dispHello()** that returns a string "Hello World".

```
function dispHello()
{
//TODO:return the string "Hello World"
}
```

**Example 5: Lab 1: Hello.js**

**Step 6:** Open **prob4.html** page in the browser, and verify that you get the same output as required.

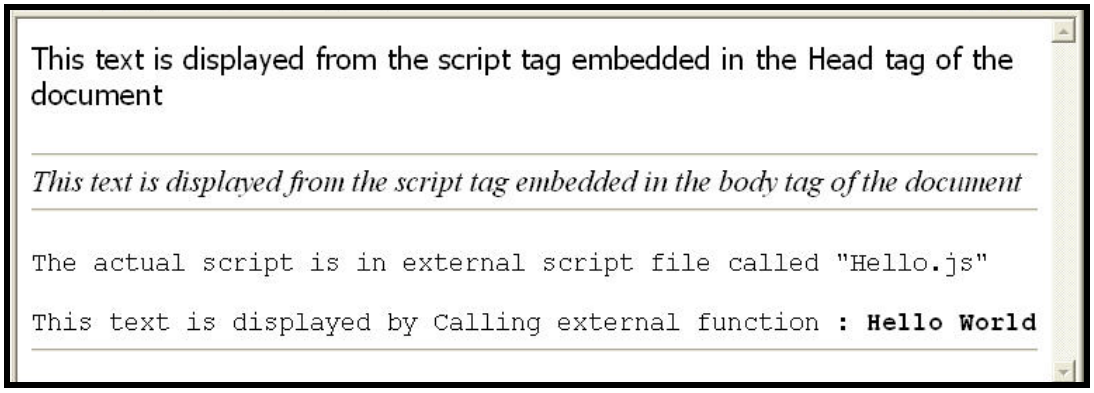



**Figure 5: Embedding Script tags in HTML document**

**1.4: Using Variable in many Script tags**

**Solution:**

**Step 1:** Create **Prob5.html** page, and complete the following code and save it in lab1 directory.

```
<html>
<head><title>Embedding Script tag in HTML Document</title>

<script>
/*
TODO:define variable headVar and initialize it to some integer value and display the value as shown in the Fig 6
*/
</script>

<hr>
</head>
<body>

<script>
/*
TODO:define variable bodyVar and initialize it to some integer value and display the value as shown in the Fig 6
*/
</script>

<hr>
<script src="common.js">
</script>

<script>
/*
TODO:  Invoke the method addNos(headVar,bodyVar)  defined in common.js file and pass the two variables headVar and bodyVar defined in the head and the body script tag and display the added result as shown in the Fig 6
*/
</script>
```

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

**Example 6: Lab 1: Prob5.html**

**Step 2:** Create a file **common.js** which has a function **addNos()** that adds two numbers and returns the addition of two numbers.

```
var msg;
msg="<p><code>The actual script is in external script file called common.js</code></p>";

function addNos(headVar,bodyVar)
{

//TODO: display the contents of the variable "msg"

//TODO: display the addition of two numbers

}
```

**Example 7: Lab 1: common.js**

**Step 3:** Open prob5.html page in the browser, and verify that you get the same output as required.

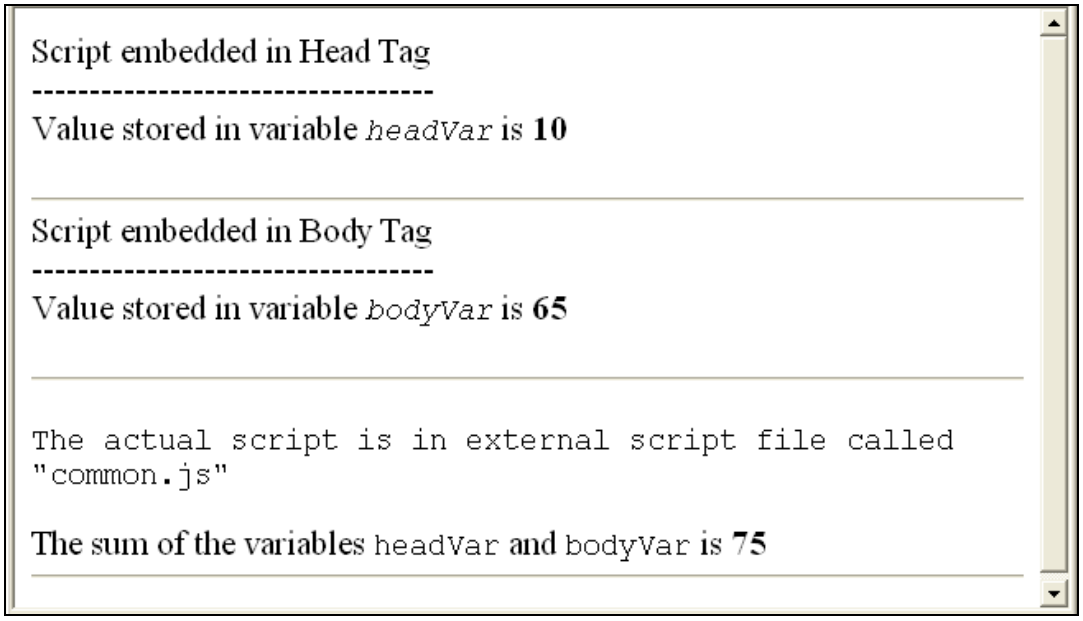



**Figure 6: Using Variable in many Script tags**

## Lab 2:The JavaScript Language

| | |
|---|---|
| **Goals** | • Learn to use looping structures and operators in JavaScript. |
| **Time** | 20 minutes |

### 2.1: For loop in JavaScript

Create a web page containing a heading “Layout is here” followed by a horizontal rule and a table with a single row as shown in the figure given below.

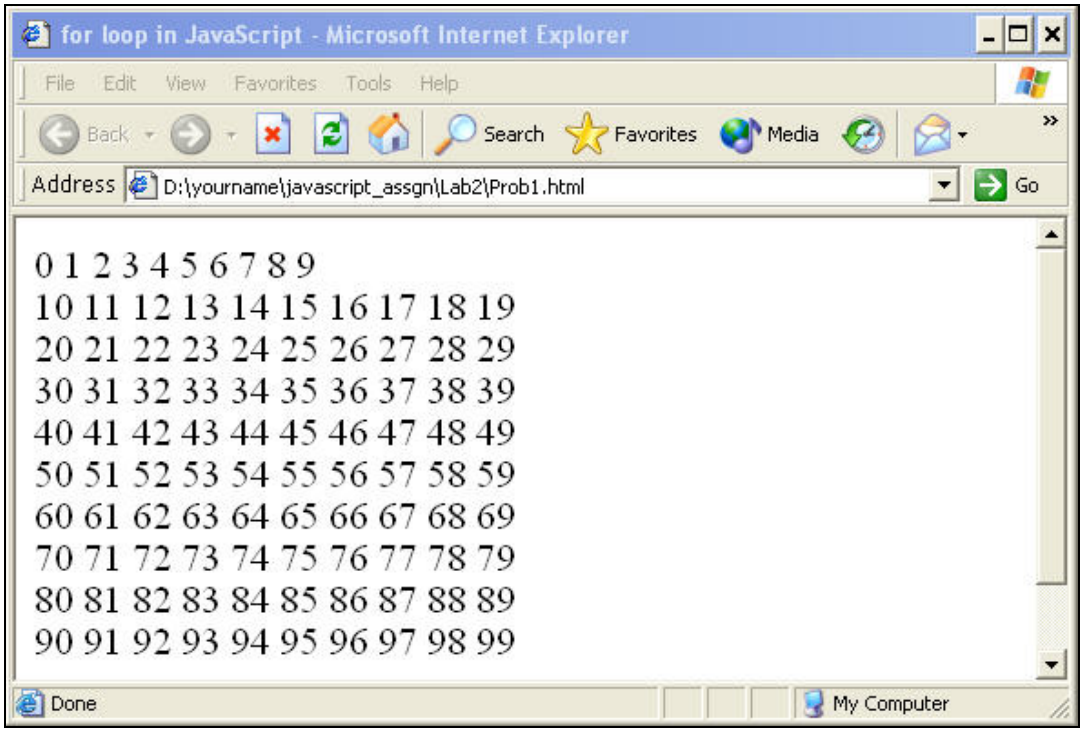



**Figure 7: For loop in JavaScript**

After completing the loop, the variable used, that is **“I”**, should be equal to **100**.

**Solution:**

**Step 1:** Write the code and save it as **Prob1.html** in lab2 directory.

**Step 2:** Open **prob1.html** page in the browser, and verify that you get the same output as required.

**Step 3:** Create **prob1_dowhile.html** and **prob1_whiledo.html** page using **do…while** and **while…do** control statements respectively to display similar output as shown in the figure given above.

**2.2: Create a web page to calculate the Compound Interest using the formula given below:**

**Compund Interest** $\left[ P * \left( 1 + \frac{r}{100} \right)^{n} \right] - P$

**Where:**

**p = Principal,**

**r = Rate of Interest,**

**n = period in years**

**The values used in the example in the following figure are as follows:**

**P = 1000, n = 1, r = 10**

```
-----------------------------------------------------
*******Calculate Compund Interest**********
-----------------------------------------------------

Prinicipal              -       1000 rs
Rate of Interest        -       10%
Period                  -       1 yr
Comp Interest           -       100
```

**Figure 8: Operators and Arithmetic Expression**

**Solution:**

**Step 1:** Write the code, and save it in lab2 directory.

**Step 2:** Open page in the browser, and verify that you get the same output as required.

## Lab 3:Working with Predefined core objects

| | |
|---|---|
| **Goals** | Understand Date, String Object<br><br>Learn to use Date and String objects in HTML pages |
| **Time** | 45 minutes |

### 3.1: Displaying Date using Date Object

Create a web page **Prob1.html**. In this web page, create a **date** object and use the **getXXXX** functions of the date object to display today's date in the format as shown below in the figure and also greet the user depending on the time the user visits the page. The message to be displayed is given in the following table. The time column shows the current date hour value.

| Time | Msg to be displayed |
|---|---|
| < 12 | Good Morning |
| >= 12 and <= 17 | Good Afternoon |
| > 17 | Good Evening |


**Figure 9: Displaying Date using Date Object**

**Solution:**

**Step 1:** Write the Code, and save it as Prob1.html in lab4 directory.

**Step 2:** Open prob1.html page in the browser, and verify that you get the same output as required.

### 3.2: Using indexOf function of String object

Create a web page **prob2.html**, which uses the **indexOf** method of string object and displays the index number of the substring searched for within the string.

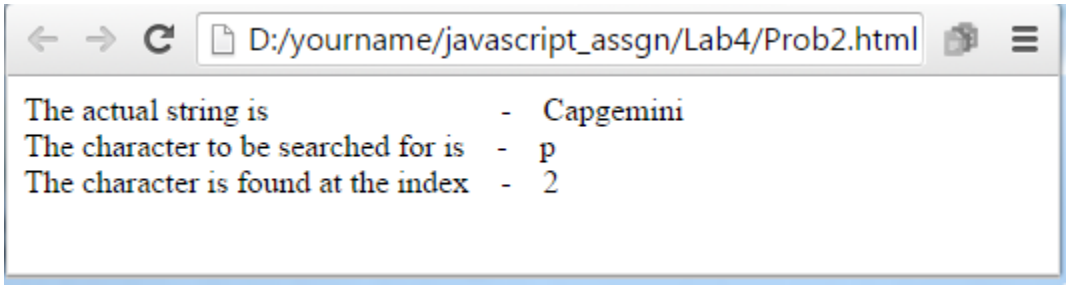



**Figure 10: Using indexOf method of String object**

**Solution:**

**Step 1:** Write the Code and save it as **Prob2.html**.

**Step 2:** Open **prob2.html** page in the browser, and verify that you get the same output as required.

### 3.3: Using various String methods

Write **prob3.html** page by completing the following code that demonstrates some of the methods of the String objects like match, **substr**, **lowerCase**, and **upperCase** to produce the output as shown in the figure given below:

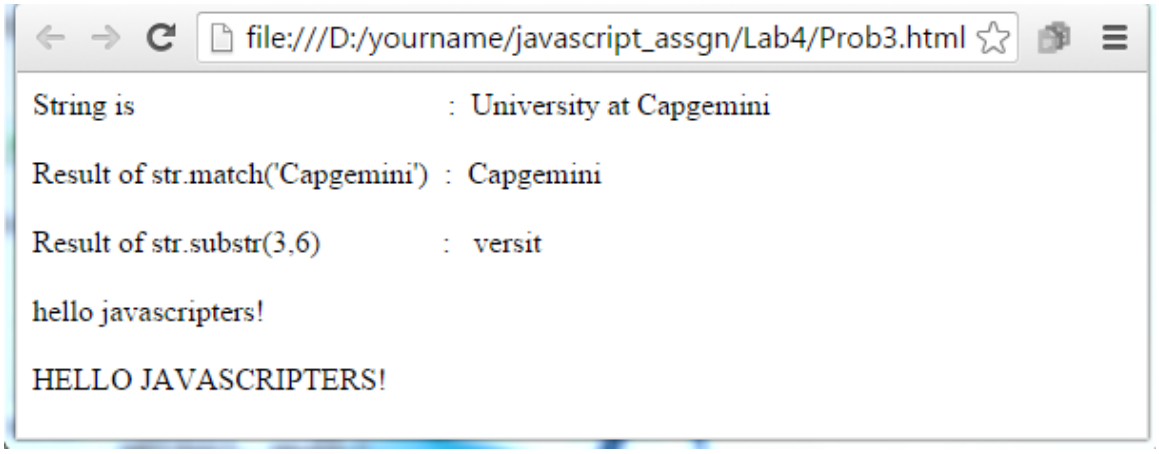


**Figure 11: Using various String methods**

**Solution:**

**Step 1:** Write the Code and save it as **Prob3.html**.

**Step 2:** Open **prob3.html** page in the browser, and verify that you get the same output as required.

## Lab 4:Working with Arrays

| | |
|---|---|
| **Goals** | Work with Array Object |
| **Time** | 10 minutes |

**4 .1: Using Array to display values**

Create a **prob1.html** web page containing script. In this script, declare an array of 6 employee names and display it in the browser as shown below:


**Figure 12: Using Array to display values**

**Solution:**

**Step 1:** Write the Code, and save it as **Prob1.html**.

**Step 2:** Open **prob1.html** page in the browser, and verify that you get the same output as required.

## Lab 5:Working with Document Object Model(DOM)

| | |
|---|---|
| **Goals** | Understand Window Object<br>Dynamically create windows<br>Handle window events |
| **Time** | 90 minutes |

**5 .1: Window object**

Create a **prob1.html** web page which has the following items as shown in the figure given below:

- a form that accepts window parameters width, height, title, left and top parameters from text field, and
- two buttons with the labels **New Window** and **Reset** to the web page

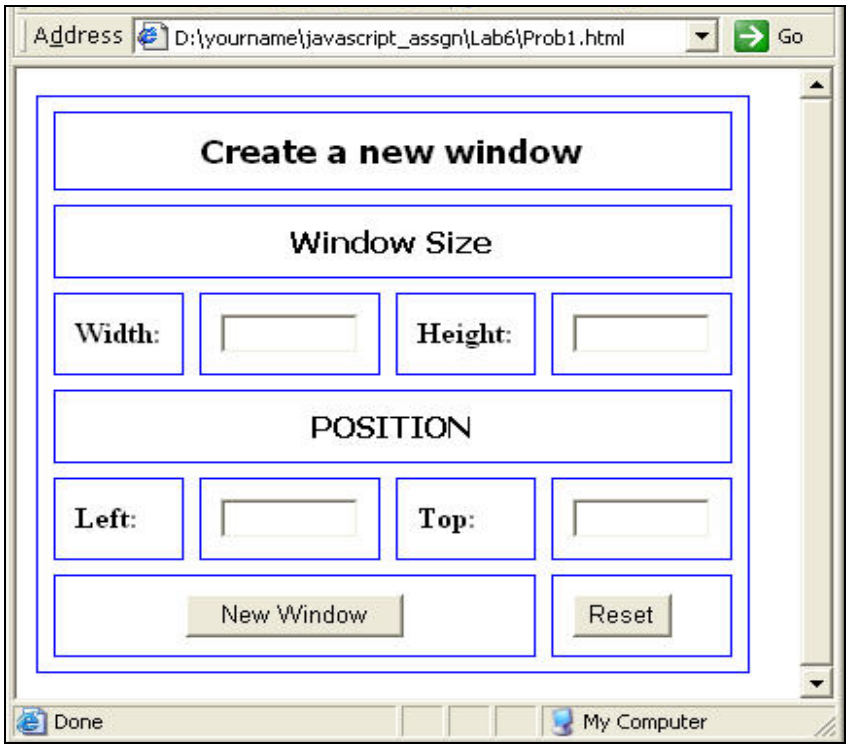



**Figure 13: Interface to accept window coordinates**

If **Reset** button is clicked, then clear all text fields. If **New Window** button is clicked, then open a new window with specifications entered in the text fields as shown in the figure given below.

**Note:** By default, the new window opens at the top left corner of the screen.

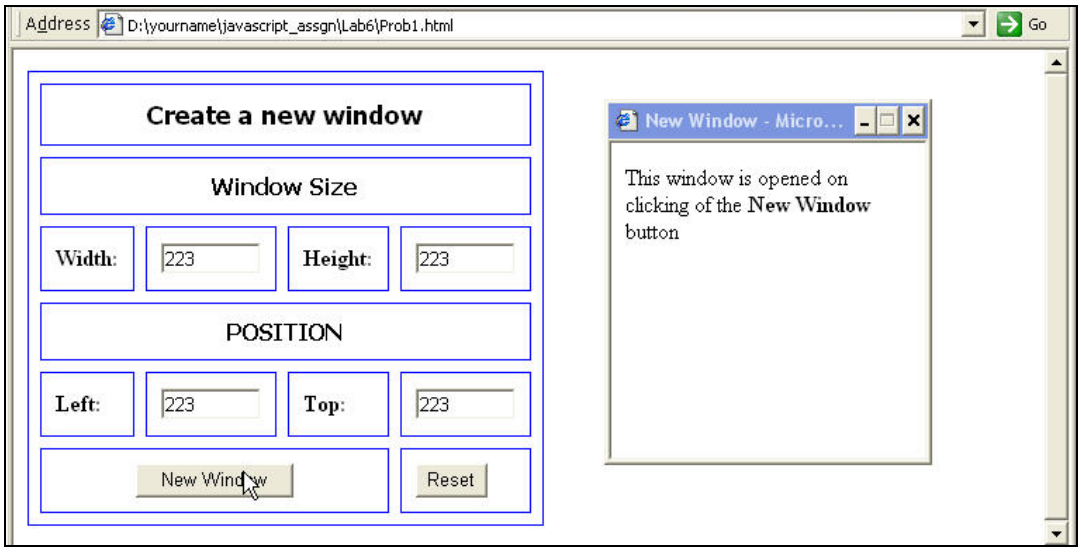



**Figure 14: Opening a window**

**Solution:**

**Step 1:** Complete the following Code and save it as **Prob1.html.**

```
<html>
<head>
<title> window example </title>
</head>
<script>
function nwindow()
{
/*TODO:  get the height, width, left and top from the form object and pass the values to open method of window along with the name of the html file to be opened in the new window.*/
}
</script>
<body >
<form id="frmlab">
<table border="1" cellspacing="8" cellpadding="10" bordercolor="blue">

// Create Table as shown in fig 6.2
</table>
```

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

**Example 8: Lab 5: Prob1.html**

**Step 2:** Open **prob1.html** page in the browser, and verify that you get the same output as required.

**Step 3:** Open **prob2.html** page in the browser, and verify that you get the same output as required.

## Lab 6:Working with Location Object

| | |
|---|---|
| **Goals** | Understand and use Location Object. |
| **Time** | 20 minutes |

### 6.1: Location Object

Create a web page which will display the properties **href**, **protocol**, and the **pathname** of the location object of your current file.

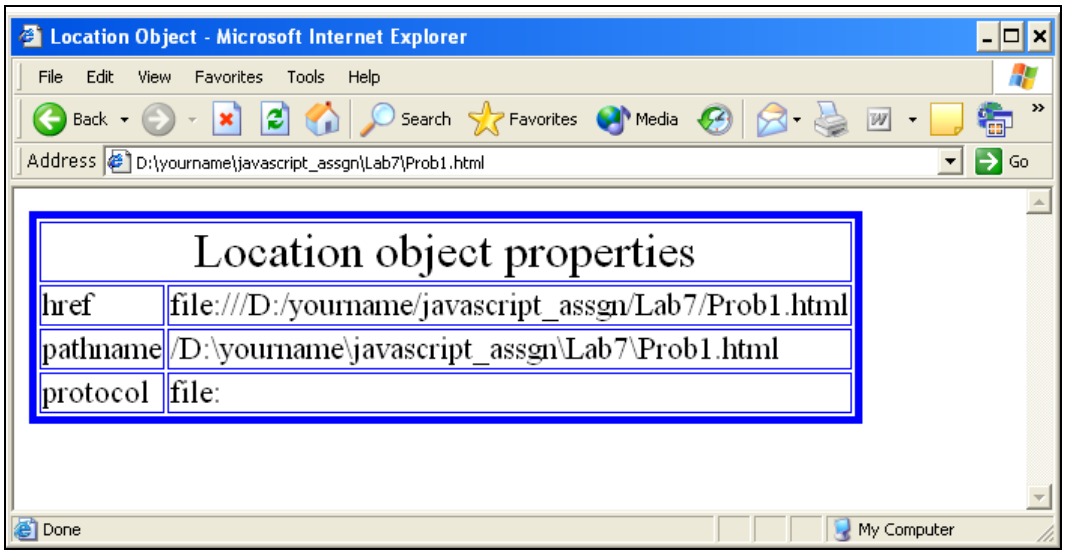



**Figure 15: Location Object Properties**

**Solution:**

**Step 1:** Write the code and save it as **Prob1.html**.

**Step 2:** Open **prob1.html** page in the browser, and verify that you get the same output as required.

## Lab 7:Working with Document Object

| | |
|---|---|
| **Goals** | Understand Document Object |
| **Time** | 120 minutes |

### 7.1: Working with Documents

Create a **prob1.html** web page which displays products available as shown in the following figure. The product details comprise Product Name, Product description, and its price.

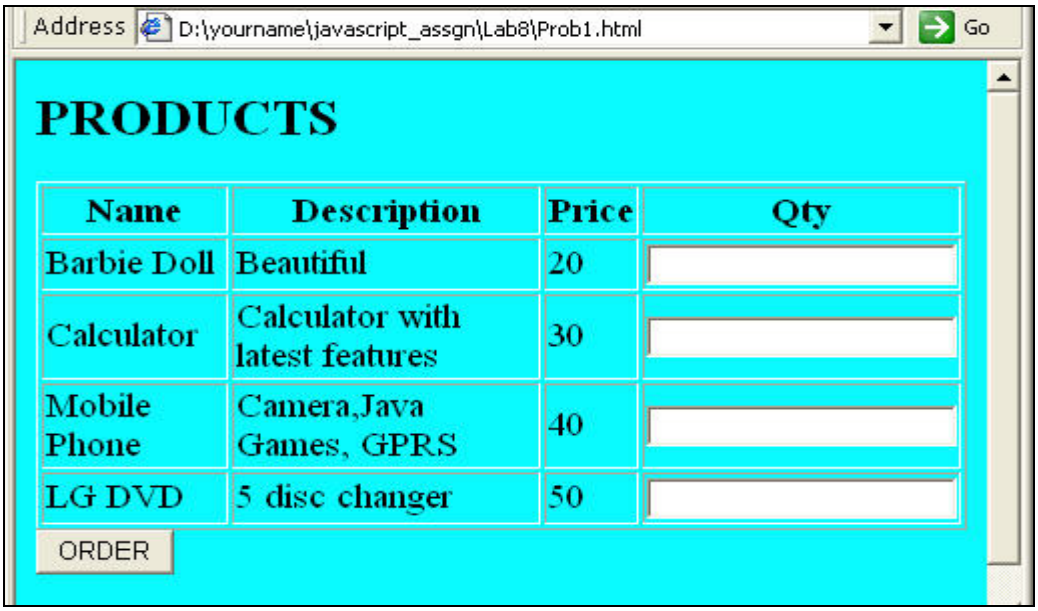



**Figure 16: Displaying Products**

Users can place orders specifying the quantity of each product. If the user does not enter quantity in any of the text fields, then an error message should be displayed as shown in the figure given below:

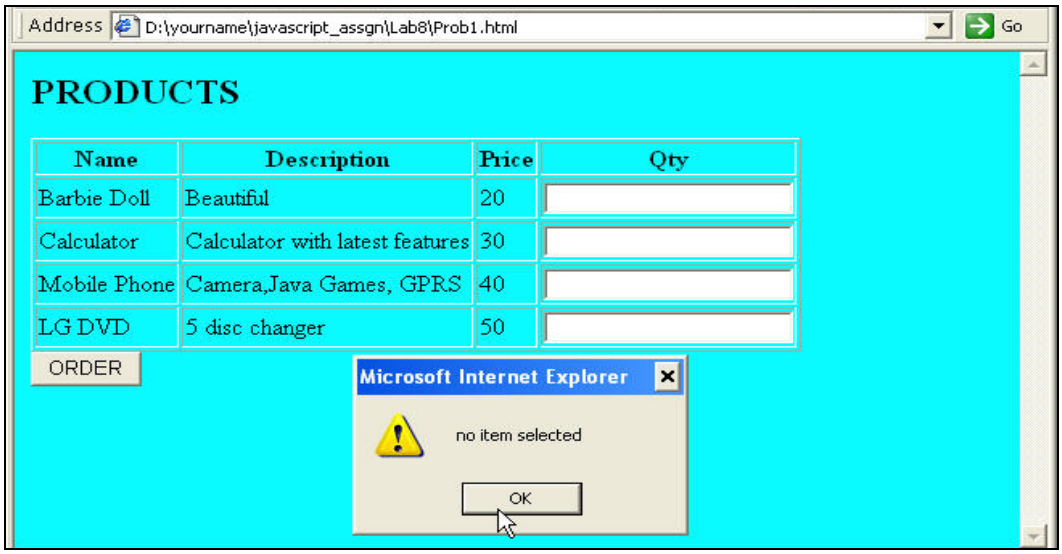



**Figure 17: Validating Products**

When the user clicks the **Order** button, the invoice for the current products transaction showing the product name, quantity ordered, price and total amount is displayed in a new window as shown in the figure given below:

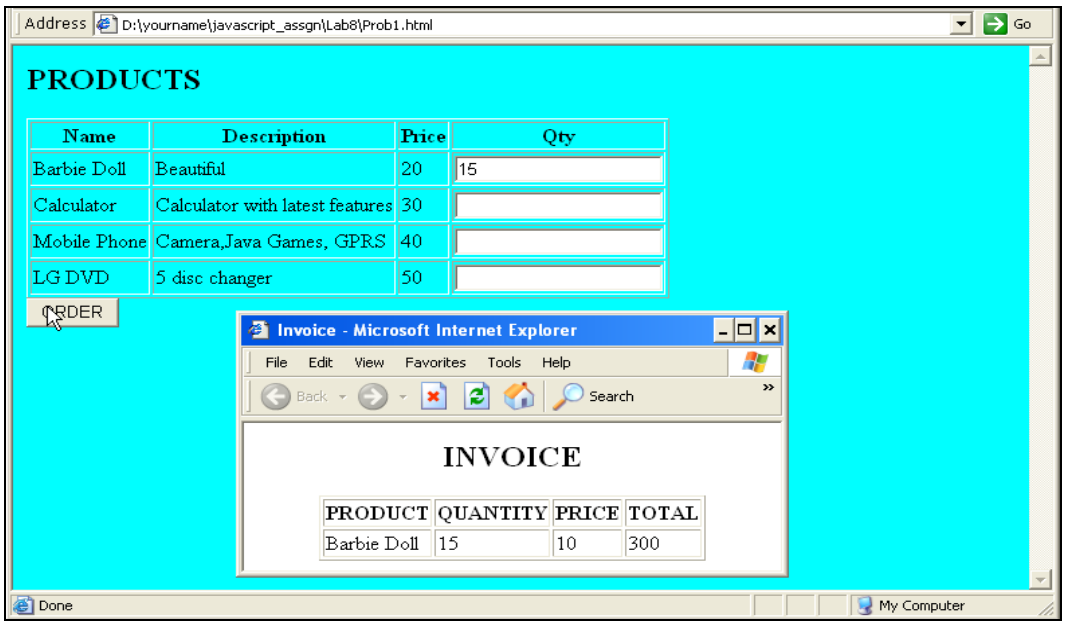



**Figure 18: Displaying Invoice details in a new window**

**Solution:**

**Step 1:** Write the code and save it as **Prob1.html**.

**Step 2:** Open **prob1.html** page in the browser, and verify that you get the same output as required.

## Lab 8:Working with Form Object

| | |
|---|---|
| **Goals** | • Understand and use Form Object. |
| **Time** | 90 minutes |

### 8.1: Form Validation

Create a **prob1.html** web page, as shown below, and calculate **Payment Information** based on **Loan Information**. Validate **Loan information** textfields for numbers. The **Payment Information** textfields should be uneditable. The other constraints are as follows:

- Amount of Loan should not be more than 15 lakhs.
- Repayment period should be between 7 yrs to 15 yrs.

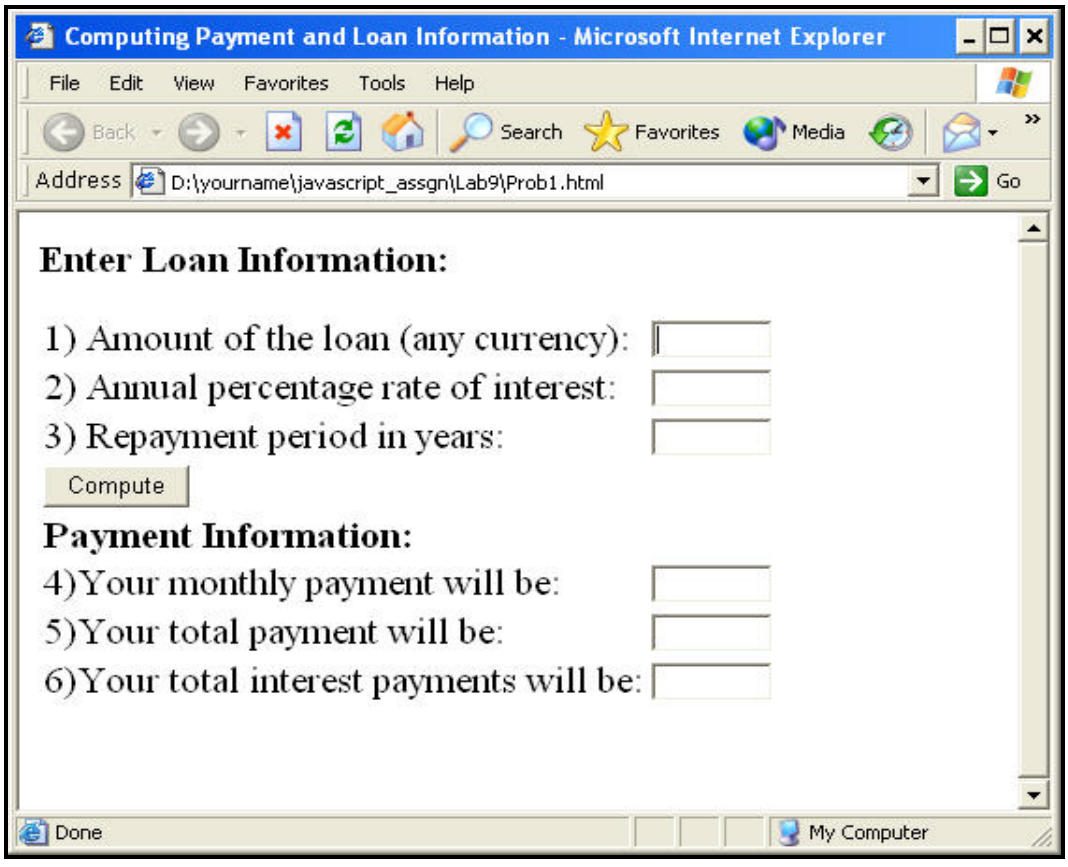



**Figure 19: Validating Form elements**

If the repayment period is not between 7 and 15, then an error message should be displayed next to this control.

Similar kind of error message should be displayed if the amount of loan exceeds 15 lakh.

```
In the function calculatePayment()
/*
TODO:
calculate the monthly payment, total payment, total interest payment on click of the button with label "compute"
*/
```

**Example 9: Lab 8: Prob1.html**

Open **prob1.html** page in the browser, and verify that you get the same output as required.

### 8.2 Validate Field

Create a **prob2.html** page as shown in the below figure.


**Figure 20: Lab 8.2 Product Details**

Data should be prepopulated in category list box (Electronics, Grocery). Based on selection of category, product list need to be populated automatically with values as given in the below table. Also Total price need to be calculated for the entered quantity as per the data in the below table. Total price field should be non-editable field.

| Category | Product | Price per quantity in Rupees |
|---|---|---|
| Electronics | Television | 20000 |
| | Laptop | 30000 |
| | Phone | 10000 |
| Grocery | Soap | 40 |
| | Powder | 90 |

While clicking on submit button, if all the text fields contains valid values then display the filled details in a popup window.

## Lab 9: Regular Expressions in JavaScript

| | |
|---|---|
| **Goals** | • Understand Regular Expression object<br>• Use Regular Expression object for validating |
| **Time** | 90 minutes |

**9.1: Regular Expression**

Create a **prob1.html** page which has two text fields – one for the Regular Expression search pattern and the other for the string in which the pattern has to be checked.

The form has two buttons one **"Test Match"** and the other **"Show Match"**.

**Test Match** will test the regular expression against the string. **Show Match** will show the matching part of the string.

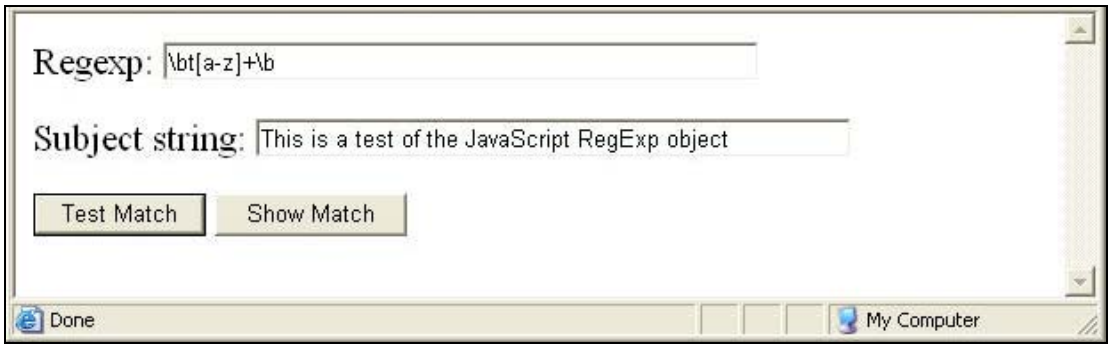



**Figure 20: Regular Expression Pattern**

Type the regular expression in the first textbox and the string in the second text box. Click the **"Test Match"** button.

If the text in the second text box matches the Regular Expression in the first text box, then it should display a message as shown below.

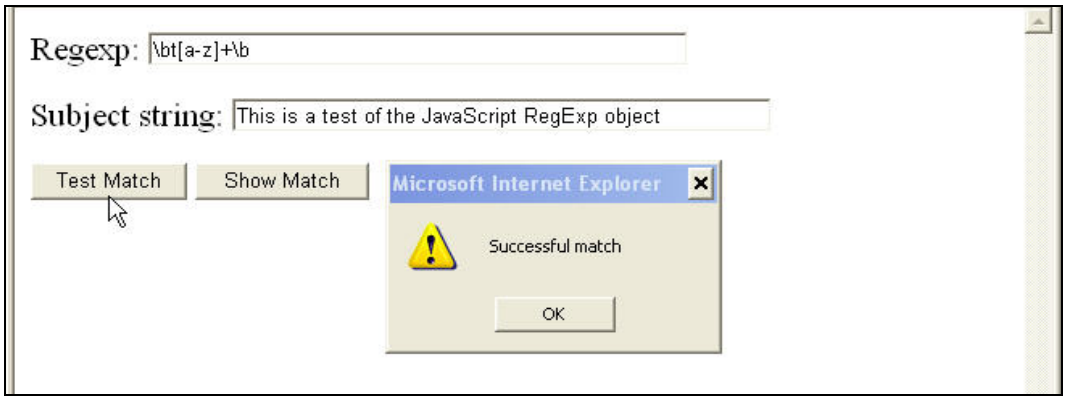



**Figure 21: Validating Regular Expression Pattern**

If you click the **"Show Match"** button, then it should display the match as shown in the figure given below:

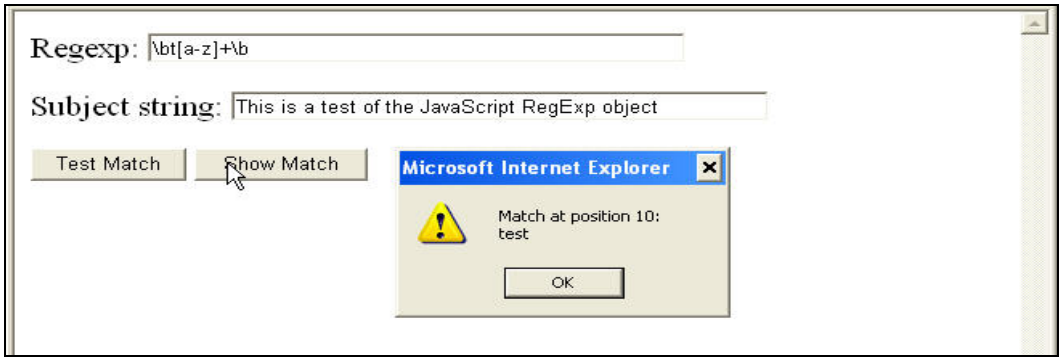



**Figure 22: Displaying text matching Regular Expression Pattern**

**Solution:**

**Step 1:** Write the Code and save it as **Prob1.html.**

Some Tips

In the function demoMatchClick()

/*

TODO:

define a variable re which is the regular expression object, to which the regular expression pattern is passed as an argument. define a variable str which holds the string from subject field match the "str" with "re" using the match method of the string object and display appropriate messages

*/

In the function demoShowMatchClick()

```
/*
TODO:
Store the regular expression pattern in a variable "re" and invoke exec method of regular expression object which takes the matching string as argument and returns the index of the matching string found
*/
```

**Example 10: Prob1.html**

**Step 2:** Open **prob1.html** page in the browser, and verify that you get the same output as required.

### 9.2: Form Validation using Regular Expression

Create a **prob3.html** page as shown in the below figure. Use CSS for designing page The page should be submitted on clicking the **Submit** button when all the form fields are properly validated.

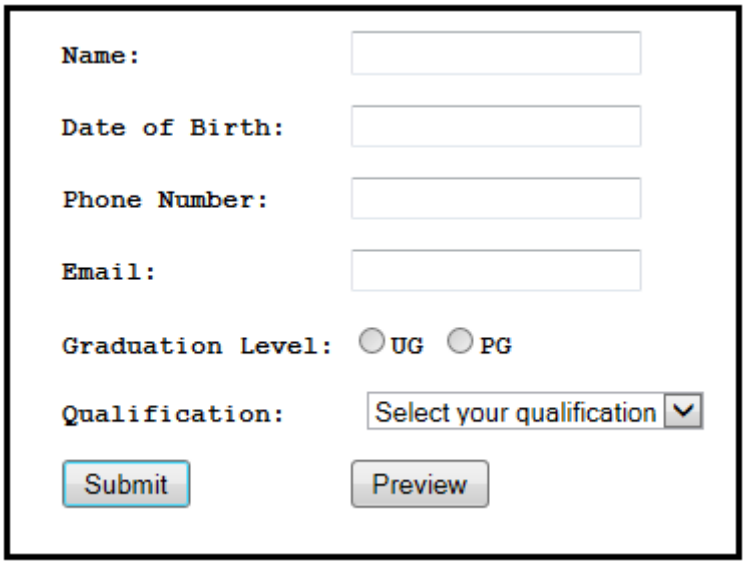



**Figure 23: Form Validation using Regular Expression**

1. None of the fields should be empty ( Use HTML 5 attributes )
2. Name field should be between 3 to 10 characters ( Use HTML 5 attributes )
3. Date of Birth format can be either (DD/MM/YY or DD/MM/YYYY) ( Use HTML 5 date control )
4. Phone Number should be in xxx-xxxx-xxxx format ( Use HTML 5 attributes )
5. Email ID should be valid. ( Use HTML 5 attributes )

6. Based on graduation level selected, qualification need to be populated automatically. For an example, if graduation level selected is UG, then qualification should be B.Sc, B.A, B.Com, etc... If graduation level selected is PG, then qualification should be M.A, M.Tech, MCA, MBA, etc… ( Call function on onChange event )
7. Calculate age of the person and display all the details in a new popup window when "Preview" button is clicked. Details should be printed in the specified format as given below:

    Name:
    Age:
    Phone Number:
    Email:
    Graduation Level:
    Qualification:

Display the appropriate error message when the condition fails.

# Appendices

### Appendix A: JavaScript Standards

1. Naming conventions for variables in JavaScript:

- Variables must begin with prefix indicating the type of the variable.
    - All integer must start with "int".
    - All floating data types must start with "flt".
    - All string must start with "str".
    - All object name must start with "obj".
    - All Boolean variables must start with "bln".
    - All variables that store date must start with "dt".
    - All constants must be in upper case with different words separated by underscore ( _ ).
    - All array variables must start with "arr".
    - Apart from these guidelines all variable name must be sensible enough, so that it's purpose can be identified from it's name.

| Type | Example |
|---|---|
| String | strStringName |
| Boolean | blnPresent |
| Array | arrArrayName |
| Object | objObjectName |
| Date | dtDateName |
| Integer | intValueInteger |
| Float | fltValueFloat |
| Constants | STRING_CONSTANT<br>ARRAY_CONSTANT<br>NUMERIC_CONSTANT |

- All HTML elements must be prefixed with appropriate types.

- TextBox "txt"
- Image "img"
- Image map Area "img"
- option button "opt"
- CheckBox "chk"
- DropDown List "lst"
- Form Name "frm"
- Buttons "btn"
- All div tags "div"
- All class names must start with "cls"
- All user-defined objects must start with "u"
- First letter of each variable/function name must be in upper case. Rest all letters must be in lowercase.
- Use of underscore and digits for naming variables must be avoided.

| Tag | Example |
|---|---|
| Div | divContent |
| Class | clsInterest |
| Form | frmContainer |
| Image | imgMapThis |
| Button | btnOk |
| TextBox | txtInterestRate |
| CheckBox | chkAllow |
| Option Button | rdbRate |
| DropDownList | lstState |

It must be noted that this naming style does not apply to HTML elements. However, when these elements are accessed in the JavaScript functions, these naming conventions must be followed. This document describes coding convention only for JavaScript.

2. Commenting

- Comments related to a particular line of code should be on the same line after the statement gets over.

```
If (dtToday == “15/07/99”) {          // Is date birthdate?
  alert (“Happy Birthday”);           // Give birthday message
} else {
  alert (“Happy Day”);                          // Give standard message
}
```

- Over all commenting should consist of two parts – Comment header and Comment footer. Comment header must precede the block of code and Comment footer must follow the block of code.

```
//Function Name:    calculateInterest
//Description:      This function calculates the interest. It accepts the initial investment and
//                  period for which the amount is invested. Rate of interest is fixed.
//                  Formula is
//                  fltInterest = fltAmount * fltPeriod          *fltRATE/100
//                  Dhrumil Dalal
//                  15/07/1999
//Author:           fltAmount – indicated the amount invested
//Start Date:       fltPeriod – Indicates the period of investment
//Input             Calculated interest
Parameters:
//
//Return Value:
Function calculateInterest(fltAmount,fltPeriod){
}


//End of function for calculating the rate of interest
```

3. Documentation

- All variables used in the function must be declared in brief.
- Only one variable declaration per line.
- Describe each variable on the same line and description should not be more than one line.
- All functions must be preceded by comments. Comments must describe the following:
    - Input parameters.
    - Return value.
    - Function logic in brief.
    - Starting date.

- Name of the author.
- Revision history.

- After the end of function, there must be block of comment indicating the end of function.

```
//Function Name:        calculateInterest
//Description:          This function calculates the interest. It accepts the initial
//                      investment and period for which the amount is invested.
//                      Rate of interest is fixed. Formula is
//                      fltInterest = fltAmount * fltPeriod        *fltRATE/100
//                      Dhrumil Dalal
//                      15/07/1999
//Author:               fltAmount – indicated the amount invested
//Start Date:           fltPeriod – Indicates the period of investment
//Input Parameters:     Calculated interest
//
//Return Value:
Function calculateInterest(fltAmount,fltPeriod){
Var fltRATE = 12.5; // fixed rate of interest
Var fltInterest;  // The variable to store calculated interest
fltInterest = fltAmount * fltPeriod * fltRATE/100 ;
return fltInterest;
}

//End of function for calculating the rate of interest
```

4. 4: Coding Styles

- For statements which may have block of code enclosed in {}, the opening brace "{" must immediately follow the statement and the closing brace "}"must be below the statement. That is to say, the closing brace and first letter of the statement must be same in the column.

```
If (condition) {
  ...
} else {
  if (condition) {
    ...
  } else {
    ...
  }
}

for(intCounter=0; intCounter <= 5; intcounter++){
  //Perform calculation.
  //Display Result}
```

- All statements within corresponding opening and closing brace must be indented. Indentations must be in odd columns.

Column no

```
123456789……
if (condition) {
  ...
} else {
  if (condition) {
    ...
  } else {
    ...
  }
}
```

- Also the code should not extend past the 80th column so that it is required to scroll to the right or left to edit a particular line. In the case of strings which do not fit on one line, it is recommended that temporary variables be used with the string concatenation operator (+=) to construct strings of longer lengths. The following example illustrates this:

```
    Column no
    123456789……80
    strMessage  = “Demonstrating the use of …    “;
    strMessage  += “prepared on 15-07-1999”;
```

**Appendix B: Coding Best Practices**

**JavaScript Best Practice**

The following demonstrate the best practices that should be followed while writing JavaScript code.

1. Inline JavaScript source code

Any JavaScript code that does not write out to the document should be placed within the head of the document.

```
<HTML>
<HEAD>
<SCRIPT>
<!--
function functionName() {
    alert(text);
}

var text = 'Hello World';
//-->
</SCRIPT>
</HEAD>
```

**Example 11: Sample Code**

This ensures that the browser has loaded the JavaScript function definitions before it is required. It also makes it slightly easier to maintain the JavaScript code if it can always be found in the head of the document.

2. JavaScript Links

Avoid using the **javascript:** protocol as a default URL within a link.

If JavaScript is disabled, then the link will not work. Do not use the following:

```
<SCRIPT>
<!--
function functionName() {
    alert('Hello world');
}
//--></SCRIPT>
<A HREF="javascript:functionName()">text link</A>
```

**Example 12: Sample Code**

Instead, use JavaScript itself to override the **href** property of the link:

```
<SCRIPT>
<!--
function functionName() {
    alert('Hello world');
}
//-->
</SCRIPT>

<A HREF="default.htm" onClick="this.href='javascript:functionName()'">text link</A>
```

**Example 13: Sample Code**

3. Avoid Using Void

All browsers do not support the **void** function. Create your own **void** function.

The in built **void()** function is supported since JavaScript 1.1. Therefore it is best to create your own void function rather than rely on JavaScript 1.1 being available.

```
<SCRIPT>
<!--
function myVoid() { } // create a void function
//-->
</SCRIPT>

<A HREF="#" onClick="this.href='javascript:myVoid()'">non functional text link</A>
```

**Example 14: Sample Code**

4. JavaScript Performance

Avoid writing output multiple times to the document, concatenate the data, and then write all in one go.

With the introduction of Netscape Navigator 4, the rendering of JavaScript generated HTML slowed down considerably.

The following writes the HTML output to the document in one go:

```
<SCRIPT>
<!--
var output = '<P>';
output += 'Last modified: ';
output += document.lastModified;
output += '<\/P>'
document.write(output);
//-->
</SCRIPT>
```

**Example 15: Sample Code**

5. Select Form Fields

Use the Netscape method to correctly navigate select field properties.

The following technique works in Microsoft Internet Explorer. However it should be avoided.

```
<SCRIPT>
<!--
var property = document.formName.selectName.propertyName
//-->
</SCRIPT>
```

**Example 16: Sample Code**

Whereas the following will work correctly in all browsers:

```
<SCRIPT>
<!--
var property =
document.formName.selectName.options[document.formName.selectName.options.selectedIndex]
.propertyName
//-->
</SCRIPT>
```

**Example 17: Sample Code**

6. Changing Location

Do not use the following:

```
<SCRIPT>
<!--
location = 'page.htm';
//-->
</SCRIPT>
```

**Example 188: Sample Code**

The later approach is confusing as it is not clear whether you are changing the location property of the “window” or the “document object”.

Changing the location using the document is deprecated and causes problems on later browsers.

Use the following:

```
<SCRIPT>
<!--
window.location.href = 'page.htm';
//-->
</SCRIPT>
```

**Example 19: Sample Code**

7. Opening Windows

While opening a new popup window using JavaScript, there are several points to bear in mind.

To be able to control the popup window from the **opener** window, always retain the returned reference from the window's open method:

```
<SCRIPT>
<!--
var windowHandle = window.open('page.htm','windowName','width=600,height=320');
//-->
</SCRIPT>
```

**Example 19: Sample Code**

To avoid errors while referring to the **opener** window from the **popup** window, always check for the in-built browser support for the **opener** property. If necessary, provide your own:

```
<SCRIPT>
<!--
var windowHandle = window.open('page.htm','windowName','width=600,height=320');
if (!windowHandle.opener)
   windowHandle.opener = self;
//-->
</SCRIPT>
```

**Example 20: Sample Code**

While updating the contents of a newly opened window, give the browser time to open the window and to load the initial contents:

```
<SCRIPT>
<!--
function update() {
   windowHandle.document.open();
   windowHandle.document.write('<H1>Hello World<\/H1>');
   windowHandle.document.close();
}

var windowHandle = window.open('page.htm','windowName','width=600,height=320');
if (!windowHandle.opener)
   windowHandle.opener = self;
setTimeout('update()',2000);
//-->
</SCRIPT>
```

**Example 21: Sample Code**

8. JavaScript Entities

JavaScript Entities are only supported by Netscape Navigator. Avoid their use.
The following will cause errors in other browsers:

```
<HR WIDTH="&{barWidth};%">
```

**Example 22: Sample Code**

Instead the following can be used.

```
<SCRIPT>
<!--
document.write('<HR WIDTH="' + barWidth + '%">');
//-->
</script>
```

**Example 23: Sample Code**

**Appendix C: Table of Figures**



**Appendix D: Table of Examples**

