Web Designing
Introduction to Java Script

Outline

- Introduction to Java Script
  - Introduction to Java Scripting and <script>
  - Client-Side JavaScript
  - Advantages of JavaScript, Limitations of JavaScript
  - Placement of Script
  - JavaScript Datatypes
  - JavaScript Variables, JavaScript Variable Scope, JavaScript Variable Names, JavaScript Reserved Words,
  - Operators in JavaScript
  - Control Statements in JavaScript
  - Introduction to Java Script Functions
  - Introduction to Java Script Events
  - Introduction to Java Script Built-in Objects

What is JavaScript

- JavaScript is a client and server-side object-based scripting language that is used to make interactive Web pages.
- A scripting language is a lightweight programming language with less complexity.
- JavaScript is the most used scripting language to add dynamism and interactivity to Web pages.
- This is because JavaScript, written on the client-side, executes on a client browser, thereby reducing the load on the server.

3

Tasks performed by client-side scripts

- Checking correctness of user input
- Monitoring user events and specifying reactions
- Replacing and updating parts of a page
- Changing the style and position of displayed elements dynamically
- Modifying a page in response to events
- Getting browser information
- Making the Web page different depending on the browser and browser features
- Generating HTML code for parts of the page

4

JavaScript is an Interpreted Language

- JavaScript is an interpreted language, which implies that scripts written to JavaScript are processed line by line.
- These scripts are interpreted by the JavaScript interpreted, which is a built-in component of the Web browser.
- JavaScript can be written on the client-side as well server-side.
- Client-side JavaScript allows you to validate only those programs that execute and produce the result on the client-machine. In counterpoint/contrast, server-side JavaScript validates only those programs that execute on the server.
- JavaScript includes various built-in objects and features that can be used to make your HTML pages dynamic.

5

JavaScript is platform-independent

- JavaScript is platform-independent, which implies that you need to write the script once and can run it on any platform or browser without affecting the output of the script.

6

Why to Learn JavaScript?

- There are the three languages, all web developers must know, these are the following:
  - HTML - to define the content of web pages
  - CSS - to define the layout of web pages
  - JavaScript - to program the behavior of web pages
- So to program the behavior of Web pages, you must have to learn JavaScript.

7

Pros & Cons of Client Side Scripting

- **Pros**
  - Allow for more interactivity by immediately responding to users' actions.
  - Execute quickly because they do not require a trip to the server.
  - The web browser uses its own resources, and eases the burden on the server.
  - It saves network bandwidth.
- **Cons**
  - Code is loaded in the browser so it will be visible to the client.
  - Code is modifiable.
  - Local files and databases cannot be accessed.
  - User is able to disable client side scripting

8

Client V/S Server Side Scripting

Server Side Scripting	Client Side Scripting
Server side scripting is used to create dynamic pages based on a number of conditions when the users browser makes a request to the server.	Client side scripting is used when the users browser already has all the code and the page is altered on the basis of the users input.
The Web Server executes the server side scripting that produces the page to be sent to the browser.	The Web Browser executes the client side scripting that resides at the user's computer.
Server side scripting is used to connect to the databases and files that reside on the web server.	Client side scripting cannot be used to connect to the databases and files on the web server.

9

Client V/S Server Side Scripting

Server Side Scripting	Client Side Scripting
Server resources can be accessed by the server side scripting.	Browser resources can be accessed by the client side scripting.
Server side scripting can't be blocked by the user.	Client side scripting is possible to be blocked by the user.
Examples of Server side scripting languages : PHP, JSP, ASP, ASP.Net, Ruby, Perl and many more.	Examples of Client side scripting languages : Javascript, VB script, etc.

10

Features of Java Script

- **Imperative and Structured**
  - The Statement Control Syntax System of Java Script is very similar to Statement Control Syntax used in C Language.
  - Java Script allows more abilities to perform as HTML is only capable of designing websites and incapable of performing logic operations like condition checking, statements looping (while & for), statement decision making (else and if) at client edge and adding two numbers.
  - The only syntactical difference between C and Java Script is that, in Java Script semi-colon is not necessary to terminate a statement, whereas in C it is necessary to terminate a statement.

11

Features of Java Script

- **Dynamic Typing**
  - JavaScript supports dynamic typing which means types of the variable are defined based on the stored value.
  - For example, if you declare a variable x then you can store either a string or a Number type value or an array or an object. This is known as dynamic typing.
  - To understand this, in languages like Java, we explicitly mention that a particular variable will store a certain type of data, whereas in JavaScript we do not have to provide the data type while declaring a variable.
  - In JavaScript, we just have to use var or let keyword before the variable name to declare a variable without worrying about its type.

12

Features of Java Script

- **Functional Style**
  - This implies that JavaScript uses a functional approach, even objects are created from the constructor functions and each constructor function represents a unique object-type.
  - Also, functions in JavaScript can be used as objects and can be passed to other functions too.
- **Platform Independent**
  - This implies that JavaScript is platform-independent or we can say it is portable; which simply means that you can simply write the script once and run it anywhere and anytime.
  - In general, you can write your JavaScript applications and run them on any platform or any browser without affecting the output of the Script.

13

Features of Java Script

- **Prototype-based Language**
  - JavaScript is a prototype-based scripting Language. This means javascript uses prototypes instead of classes or inheritance.
  - In languages like Java, we create a class and then we create objects for those classes. But in JavaScript, we define object prototype and then more objects can be created using this object prototype.

14

<script> tag

- Syntax of a JavaScript program/code is nothing, it's just a set of rules that defines how JavaScript programs are constructed.
- A JavaScript program consists of JavaScript statements that is placed inside the HTML **<script> ... </script>** tag in a web page.
- You are free to place the <script> containing your JavaScript program anywhere within your web page, but to keep it within the <head> tag is the preferred way.
- Here is the general form shows how script tag plays a role in placing JavaScript code inside it:

<script ...>

JavaScript code

</script>

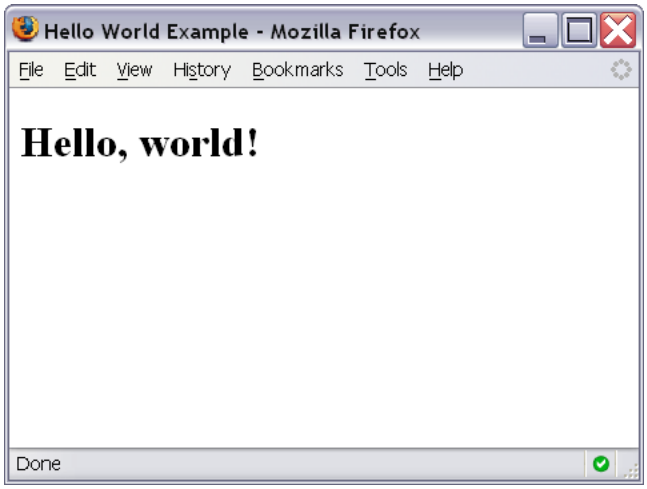
15

Hello world in JavaScript example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World Example</title>
  </head>
  <body>
    <script type="text/javascript">
      <!--
        document.write("<h1>Hello, world!</h1>");
      //-->
    </script>
  </body>
</html>
```

16

Hello World Screenshot



17

<script> tag

- The script tag takes two important attributes:
  - **language** - The language attribute is used to specify what scripting language you are using
  - **type** - The type attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript“
- Syntax of your JavaScript code segment will be:

```
<script language="javascript" type="text/javascript">
        JavaScript code
</script>
```

18

<script> tag

- You can place or include your JavaScript code anywhere in your HTML document, even you can also place your JavaScript code in an external file and include this file in any HTML document to use your JavaScript code present in that file.
- Here are the place, where you can include your JavaScript code:
  - In the <head> .. </head> section
  - in the <body> .. </body> section
  - In the <body> .. </body> and <head> .. </head> sections
  - In an external file and then include this file in the <head> .. </head> section
- You can place any number of scripts in an HTML document. Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

19

<script> tag

- The best practice is to put JavaScript <script> tags just before the closing </body> tag rather than in the <head> section of your HTML.
- The reason for this is that HTML loads from top to bottom. The head loads first, then the body, and then everything inside the body. If we put our JavaScript in the head section, the entire JavaScript will load before loading any of the HTML, which could cause a few problems :
- If you have code in your JavaScript that alters HTML as soon as the JavaScript code loads, there won't actually be any HTML elements available for it to affect yet, so it will seem as though the JavaScript code isn't working, and you may get errors.
- If you have a lot of JavaScript, it can visibly slow the loading of your page because it loads all of the JavaScript before it loads any of the HTML.

20

<script> tag

- When you place your JavaScript at the bottom of your HTML body, it gives the HTML time to load before any of the JavaScript loads, which can prevent errors, and speed up website response time.
- In the HEAD, scripts are run before the page is displayed
- In the BODY, scripts are run as the page is displayed
- In the HEAD is the right place to define functions and variables that are used by scripts within the BODY

21

<script> tag

- The <script> tag is used to define a client-side script (JavaScript).
- The <script> element either contains scripting statements, or it points to an external script file through the src attribute.
- Example :

Code

```
<html>
  <head>
    <title>HTML script Tag</title>
  </head>
  <body>
    <script type="text/javascript">
      // Java Script Code Here
    </script>
  </body>
</html>
```

Code

```
<html>
  <head>
    <title>HTML script Tag</title>
  </head>
  <body>
    <script src="PathToJS">
    </script>
  </body>
</html>
```

22

Script in Head

```
<!DOCTYPE HTML>
<HTML>
        <HEAD>
          <TITLE>Adding script in HEAD element</TITLE>
          <H1>Adding script in the HEAD Element</H1>
           <SCRIPT type="text/javascript">
                document.write("Welcome to the World of
                JavaScript <BR/>");
                document.write("In this example we have used the
                JavaScript in the HEAD element.");
           </SCRIPT>
        </HEAD>
</HTML>
```

23

Script in body

```
<!DOCTYPE HTML>
<HTML>
        <HEAD>
                <TITLE>Adding script in Body Element</TITLE>
        </HEAD>
        <BODY>
                <H1>Adding JavaScript in Body Element</H1>
                <SCRIPT type="text/javascript">
                        document.write("Welcome to the World of
                         JavaScript <BR/>");
                        document.write("In this example we have
                        used the JavaScript in the BODY element.");
                </SCRIPT>
        </BODY>
</HTML>
```

24

Script in head and body

```
<!DOCTYPE HTML>
<html>
 <head>
   <script type="text/javascript">
        function fun1() {
              alert("Hello HTML in Head")  }
    </script>
 </head>
 <body>
    <script type="text/javascript">
         document.write("Hello HTML in body")
     </script>
     <br/>
      <input type="button" onclick="fun1()" value="Click Here" />
  </body>
</html>
```

25

JavaScript in external file

- JavaScript Code in External File is the most preferred and safe way to use your all JavaScript code from an external file.
- Because if you want to use same JavaScript code in multiple HTML file, then this is the most preferred way.
- You can write all your JavaScript code in a simple text file having name like **myjavascript.js.** Use **.js** extension after the filename.
- Now you can include this file in any HTML document to use all the code present in that file. And later if you want to change the JavaScript code, you only have to make changes in a single file (say myjavascript.js), and all the HTML document automatically changes.

26

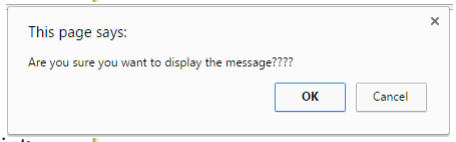
JavaScript in external file

message.js

```
function myAlert(msg) {
        if(confirm("Are you sure you want to display the message????")) {
                alert(msg);
        }
        else {
                alert("Message not Displayed as User Cancled Operation");
        }
}
```

myHtml.html

```
<html>
   <head>
      <script src="message.js"></script>
   </head>
   <body>
      <script> myAlert("Hello World"); </script>
   </body>
</html>
```



27

Script in External File

```
<!DOCTYPE HTML>
<HTML>

        <HEAD>
                <TITLE>Using the External Script file</TITLE>
        </HEAD>

        <BODY>
                <H2>The sum of First Four Number is:</H2>
                <H2><script src="MyScript.js"></H2>
                </SCRIPT>
        </BODY>
</HTML>
```

28

MyScript.js

```
var i, count=0;
for(i=0;i<5;i++)
{
        count=count+i;
}
document.write(count);
```

29

Fundamentals of JavaScript

- Programming Fundamentals of JS
- Lexical Structure
- Variables
- Operators
- Control Flow Statements
- Popup Boxes

30

Lexical Structure of JavaScript

- The lexical Structure of JS defines the rules for the following aspects:
  - Character Set
  - Case Sensitivity
  - White spaces and Line breaking
  - Optional Semicolons
  - Comments
  - Literals
  - Identifiers
  - Reserved Keywords

31

Lexical Structure of JavaScript

- **Understanding Character Set**
  - Character Set is a set of character reserved to write JS programs. ASCII character set.
- **Case Sensitivity**
  - JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.
  - So the identifiers Time and TIME will convey different meanings in JavaScript.
  - **NOTE** − Care should be taken while writing variable and function names in JavaScript.

32

Lexical Structure of JavaScript

- **Whitespace and Line Breaks**
  - JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.
- **Semicolons are Optional**
  - Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

33

Lexical Structure of JavaScript

```
<script language = "javascript" type = "text/javascript">
  <!--
    var1 = 10
    var2 = 20
  //-->
</script>
```

- But when formatted in a single line as follows, you must use semicolons

```
<script language = "javascript" type = "text/javascript">
  <!--
    var1 = 10; var2 = 20;
  //-->
</script>
```

- **Note** − It is a good programming practice to use semicolons.

34

Example

```
<!DOCTYPE HTML>
<HTML>
        <HEAD>
           <TITLE>Result</TITLE>
        </HEAD>
        <BODY>
           <H1>Result </H1>
           <SCRIPT type="text/javascript">
                var i="MCA"
                document.write("<BR/>Welcome to : " + i)
                document.write("<BR/>Today topic is JS ")
                </SCRIPT>
        </BODY>
</HTML>
```

35

Lexical Structure of JavaScript

- **Comments in JavaScript**
  - JavaScript supports both C-style and C++-style comments
  - Two types of comments
    - Single line : Uses two forward slashes (i.e. //)
    - Multiple line : Uses /* and */
  - Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
  - Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
  - JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
  - The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

36

Comments in JavaScript

- Single line : Uses two forward slashes (i.e. //)

```
<script type="text/javascript">
 <!--
  // This is my JavaScript comment
  document.write("<h1>Hello!</h1>");
      //-->
</script>
```

- Multiple line : Uses /* and */

```
<script type="text/javascript">
 <!--
  /* This is a multiple line comment.
   * The star at the beginning of this line is optional.
   * So is the star at the beginning of this line.
   */
  document.write("<h1>Hello!</h1>");
 //-->
</script>
```

37

Lexical Structure of JavaScript

- **JavaScript Literals**
  - JavaScript Literals are the fixed value that cannot be changed, you do not need to specify any type of keyword to write literals.
  - Literals are often used to initialize variables in programming, names of variables are string literals.
  - A JavaScript Literal can be a numeric, string, floating-point value, a boolean value or even an object.
  - In simple words, any value is literal, if you write a string "Studytonight" is a literal, any number like 7007 is a literal, etc.
  - JavaScript supports various types of literals which are : Numeric Literal, Floating-Point Literal, Boolean Literal, String Literal, Array Literal, Regular Expression Literal, Object Literal

38

Lexical Structure of JavaScript

- **JavaScript Literals**
  - **Numeric Literal**
    - It can be expressed in the decimal(base 10), hexadecimal(base 16) or octal(base 8) format.
    - Decimal numeric literals consist of a sequence of digits (0-9) without a leading 0(zero).
    - Hexadecimal numeric literals include digits(0-9), letters (a-f) or (A-F).
    - Octal numeric literals include digits (0-7). A leading 0(zero) in a numeric literal indicates octal format.
    - JavaScript Numeric Literals Example
      - 120 // decimal literal
      - 021434 // octal literal
      - 0x4567 // hexadecimal literal

39

Lexical Structure of JavaScript

- **JavaScript Literals**
  - **Floating-Point Literal**
    - It contains a decimal point(.)
    - A fraction is a floating-point literal
    - It may contain an Exponent.
    - JavaScript Floating-Point Literal Example
      - 6.99689 // floating-point literal
      - -167.39894 // negative floating-point literal
  - **Boolean Literal**
    - Boolean literal supports two values only either true or false.
    - JavaScript Boolean Literal Example
      - true // Boolean literal
      - false // Boolean literal

40

Lexical Structure of JavaScript

- **JavaScript Literals**
  - **String Literal**
    - A string literal is a combination of zero or more characters enclosed within a single(') or double quotation marks (").
    - JavaScript String Literal Example
      - " MCA" // String literal
      - ' Department' // String literal

41

Special characters in String Literal

Character	Description
\b	It represents a backspace.
\f	It represents a Form Feed.
\n	It represents a new line.
\r	It represents a carriage return.
\t	It represents a tab.
\v	It represents a vertical tab.
\'	It represents an apostrophe or single quote.
\"	It represents a double quote.
\\	It represents a backslash character.
\uXXXX	It represents a Unicode character specified by a four-digit hexadecimal number.

42

Lexical Structure of JavaScript

- **JavaScript Literals**
  - **Array Literal**
    - An array literal is a list of zero or more expressions representing array elements that are enclosed in a square bracket([]).
    - Whenever you create an array using an array literal, it is initialized with the elements specified in the square bracket.
    - JavaScript Array Literal Example
      - var emp = ["aman","anu","charita"]; // Array literal

43

Lexical Structure of JavaScript

- **JavaScript Literals**
  - **Regular Expression Literal**
    - Regular Expression is a pattern, used to match a character or string in some text. It is created by enclosing the regular expression string between forward slashes.
    - JavaScript Regular Expression Example
      - var myregexp = /ab+c/;
      - var myregexp = new RegExp("abc");

44

Lexical Structure of JavaScript

- **JavaScript Literals**
  - **Object Literal**
    - It is a collection of key-value pairs enclosed in curly braces({}). The key-value pair is separated by a comma.
    - JavaScript Object Literal Example
      - var games = {cricket :11, chess :2, carom: 4}

45

Lexical Structure of JavaScript

- **JavaScript Literals**
  - **Identifiers**
    - Identifiers are names.
    - In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).
    - The rules for legal names are much the same in most programming languages.
    - In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).
    - Subsequent characters may be letters, digits, underscores, or dollar signs.
    - Numbers are not allowed as the first character.
    - This way JavaScript can easily distinguish identifiers from numbers.

46

Lexical Structure of JavaScript

- **JavaScript Literals**
  - **Reserved Words**
    - A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	instanceof	switch	const	**for**	private	typeof
boolean	enum	int	synchronized	continue	function	protected	var
break	export	interface	this	debugger	goto	public	void
byte	extends	long	throw	default	if	return	volatile
case	false	native	throws	delete	implements	short	while
catch	final	new	transient	do	import	static	with
char	finally	null	true	double	in	super	
class	float	package	try				

JavaScript Datatypes

- One of the most fundamental characteristics of a programming language is the set of data types it supports.
- These are the type of values that can be represented and manipulated in a programming language.
- JavaScript allows you to work with three primitive data types −
  - Numbers, eg. 123, 120.50 etc.
  - Strings of text e.g. "This text string" etc.
  - Boolean e.g. true or false.
- JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value.
- In addition to these primitive data types, JavaScript supports a composite data type known as object.

JavaScript Variable

- Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers.
- You can place data into these containers and then refer to the data simply by naming the container.
- A variable can contain several types of value:
  - **Number** : a numeric value e.g. 156, 100, 1.2
  - **String** : character wrapped in quotes e.g. “rajkot”
  - **Boolean** : a value of true or false
  - **Null** : an empty variable
  - **Function** : a function name
  - **Object** : an object

49

JavaScript Variable

- Attributes of Javascript variables :
  - It is a case sensitive. (mynum and MyNum are different variables)
  - It cannot contain punctuation, space or start with a digit
  - It cannot be a JavaScript reserved word
- Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type = "text/javascript">
  <!--
    var money;
    var name;
  //-->
</script>
```

50

JavaScript Variable

- You can also declare multiple variables with the same keyword as follows:

```
<script type = "text/javascript">
   <!--
      var money, name;
   //-->
</script>
```

- Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.
- For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

51

JavaScript Variable

```
<script type = "text/javascript">
   <!--
      var name = "Ali";
      var money;
      money = 2000.50;
   //-->
</script>
```

- **Note** − Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

52

Declaring a Variable

```
<!DOCTYPE HTML>
<HTML>
        <HEAD>
                <TITLE>Declaring a Variable</TITLE>
        </HEAD>
        <BODY>
                <H1>Using a variable in the script</H1>
                <SCRIPT type="text/javascript">
                    var countBooks=100;
                    document.write("Number of Books=" +
                    countBooks);
                </SCRIPT>
        </BODY>
</HTML>
```

53

Assigning Values to Multiple Variable

```
<!DOCTYPE HTML>
<HTML>
  <HEAD>
        <TITLE>Assigning values to Multiple Variables</TITLE>
   </HEAD>
   <BODY>
        <H1>Using Multiple Variable in the Script</H1>
        <SCRIPT type="text/javascript">
            var countBooks=50, minBookPrice=150,
             maxBookPrice=3000, bookName;
             countBooks=2000;
             minBookPrice=500;
             maxBookPrice=minBookPrice;
            document.write("countBooks = " +
            countBooks+"<BR/>");
```

54

Assigning Values to Multiple Variable

```
            document.write("maxBookPrice = " +
            maxBookPrice+"<BR/>");
            document.write("bookName = " + bookName);
        </SCRIPT>
    </BODY>
</HTML>
```

55

JavaScript Variable

- JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold.
- The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.
- JavaScript Variable Scope
- The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.
  - **Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
  - **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

56

JavaScript Variable

- Within the body of a function, a local variable takes precedence over a global variable with the same name.
- If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

57

JavaScript Variable

```
<html>
  <body onload = checkscope();>
    <script type = "text/javascript">
        var myVar = "global";      // Declare a global variable
         function checkscope( ) {
           var myVar = "local";    // Declare a local variable
           document.write(myVar);
         }
      </script>
  </body>
</html>
```

58

JavaScript Operators

- Operators in JavaScript, are used to perform some mathematical or logical manipulations. There are following types of operators available in JavaScript:
  - Arithmetic Operators : (+, - , *, /, %)
  - Assignment Operators : (=, +=, -=, *= /=, %=, ++, --)
  - Comparison Operators : (<, >, <=, >=, ==)
  - Logical (Relational) Operators : (&&, ||, !)
- **Note**: + also does string concatenation

59

JavaScript Arithmetic Operators

- JavaScript arithmetic operators are used to perform arithmetical task. Here, the following table lists the arithmetic operators available in JavaScript:
- **Note** - JavaScript addition operator (+) works for numeric as well as strings. For example, "a" + 20 will give "a20"

Operator	Meaning
+	This operator adds two operands
-	This operator subtracts second operand from the first
/	This operator divide numerator by denominator
*	This operator multiply both operands
%	This is a modulus operator, used to calculate the remainder
--	This is a decrement operator, decreases integer value by one
++	This is a increment operator, increases integer value by one

60

JavaScript Assignment Operators

- Assignment operators in JavaScript, are used to assign the values of right side operand to the left side operand. Here, this table lists the assignment operators available in JavaScript:

Operator	Meaning
=	This operator Assigns values from the right side operands to left side operand
-=	This operator subtracts right operand from the left operand and assign the result to the left operand
+=	This operator adds right operand to the left operand and assign the result to the left operand
/=	This operator divides left operand with the right operand and assign the result to the left operand
*=	This operator multiplies right operand with the left operand and assign the result to the left operand
%=	This operator takes modulus using two operands and assign the result to the left operand

JavaScript Comparison Operators

- JavaScript comparison operators are basically used in logical statements which is to determine the equality or difference between variables or values. The following table lists the comparison operators available in JavaScript:

Operator	Name
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

62

JavaScript Logical Operators

- JavaScript logical operators are used to determine the logic between the variables or values. The following table lists the logical operators available in JavaScript:

Operator	Name
||	or
&&	and
!	not

63

JavaScript Conditional (ternary) Operators

- Conditional operators in JavaScript, are used to assign a value to a variable based on some condition. Here is the general form to use JavaScript conditional operators:

- variable_name = (condition) ? value1:value2

64

JavaScript Operators Precedence

- The following table describes the operators in the decreasing order of their precedence:

Operator Name	Operator
member operator	. []
new operator	new
function call operator	()
increment operator	++
decrement operator	--
logical NOT operator	!
bitwise NOT operator	~
unary plus operator	+
unary negation operator	-
typedef operator	typedef

65

JavaScript Operators Precedence

Operator Name	Operator
void operator	void
delete operator	delete
multiplication operator	*
division operator	/
modulus operator	%
addition operator	+
subtraction operator	-
bitwise shift operator	<< >> >>>
relational operator	< <= > >=

66

JavaScript Operators Precedence

Operator Name	Operator
in operator	in
instanceof operator	instanceof
equality operator	== != === !==
bitwise AND operator	&
bitwise XOR operator	^
bitwise OR operator	|
logical AND operator	&&
logical OR operator	||
conditional operator	?:

67

JavaScript Operators Precedence

Operator Name	Operator
assignment operator	= += -= *= /= %= <<= >>= >>>= &= ^= |=
comma operator	,

68

Using Operators Precedence

```
<!DOCTYPE HTML>
<HTML>
   <HEAD>
     <TITLE>Using Operators</TITLE>
   </HEAD>
   <BODY>
     <H1>Using Operators in the Script </H1>
     <SCRIPT type="text/javascript">
          var math=95, English=80, science=90, avgMarks;
          avgMarks=math + English + science / 3;
          document.write("Average Marks = " + avgMarks);
          avgMarks=(math + English + science) / 3;
          document.write("<BR/>Average Marks = " + avgMarks);
     </SCRIPT>
  </BODY>
</HTML>
```

69

JavaScript Conditional Statements

- JavaScript conditional statements or decision making statements are used to perform particular actions based on the given conditions.
  - if Statement
  - if-else Statement
  - switch Statement

If condition

```
if(a>10)
{
        alert("A is > that 10");
}
```

ternary operator

```
max = a>b ? a : b ;
```

switch

```
switch(expression)
{
    case lbl1:
        // code to execute
         break;
    case lbl2:
        // code to execute
        break;
}
```

JavaScript If Statements

- The if statement in JavaScript is used to execute a group of one or more than one script statements only when the given/particular condition is met.
- **Syntax:**

```
if(condition)
{
        Statement1
}
```

71

Using If Statement

```
<!DOCTYPE HTML>
<HTML>
  <HEAD>
    <TITLE>Using if Statement</TITLE>
  </HEAD>
  <BODY>
    <H1>Using the if Statement in the script </H1>
    <SCRIPT type="text/javascript">
      var Number=45;
      if((Number%2) != 0)  {
            document.write(Number + " is an odd number");
      }
      document.write("<BR/>Thank you!");
    </SCRIPT>
  </BODY>
</HTML>
```

72

JavaScript If-else Statements

- The if-else statement in JavaScript is used to execute the code that is written in the if statement, if the specified condition is true. Otherwise, the code written in the else statement is executed.
- **Syntax**:

```
if(condition)
{
        Statement1
}
else
{
        Statement2
}
```

73

Using Ifelse Statement

```
<!DOCTYPE HTML>
<HTML>
  <HEAD>
      <TITLE>if...else Statement</TITLE>
  </HEAD>
  <BODY>
      <H1>Using the if...else Statement in the Script</H1>
      <SCRIPT type="text/javascript">
          var Number=44;
          if((Number%2) != 0)
          {
           document.write(Number + " is an odd number");
          }
```

74

Using Ifelse Statement

```
                else
                {
                  document.write(Number + " is an even  number");
                }
                document.write("<BR/>Thank you!");
        </SCRIPT>
   </BODY>
</HTML>
```

75

JavaScript Nested If-else Statements

- You are free to define one if-else statement into another, such statements are called as nested if-else statements.
- Nested if-else statement are useful in situations when additional checking is required.
- **Syntax**:

```
if(condition){
                Statement1}
else{     if(condition) {
                Statement2 }
          else {
                Statement3}
}
```

76

Using NestedIf Statement

```
<!DOCTYPE HTML>
<HTML>
  <HEAD>
      <TITLE>Nested if... else Statement</TITLE>
  </HEAD>
  <BODY>
      <H1>Using Nested if... else Statement</H1>
      <SCRIPT type="text/javascript">
            var letter="I";
            if (letter=="A")
                  document.write("vowel A");
            else
                  if(letter=="E")
                        document.write("vowel E");
                  else
```

77

Using NestedIf Statement

```
                  if(letter=="I")
                        document.write("vowel I");
                  else
                        if(letter=="O")
                           document.write("vowel O");
                        else
                           if(letter=="U")
                             document.write("vowel U");
                           else
                             document.write("consonant");
      document.write("<BR/>Thank you!");
    </SCRIPT>
  </BODY>
</HTML>
```

78

JavaScript Switch Statements

- The switch statement in JavaScript is used to select a particular group of statements to be executed among several other groups of statements.
- **Syntax**:

```
switch(expression)
{
    case value1: statement1
    break;
    case value2: statement2
     break;
    default : statement_default
    break;
}
```

79

Using Switch Statement

```
<!DOCTYPE HTML>
<HTML>
  <HEAD>
      <TITLE>Using switch Statement</TITLE>
  </HEAD>
  <BODY>
      <H1>Using switch Statement in the script</H1>
      <SCRIPT type="text/javascript">
        var letter="I";
         switch(letter)
         {
            default:document.write("consonant");
            break;
            case "A":document.write("A is a vowel");
            break;
            case "E":document.write("E is a vowel");
```

80

Using Switch Statement

```
                break;
                case "I":document.write("I is a vowel");
                break;
                case "O":document.write("O is a vowel");
                break;
                case "U":document.write("U is a vowel");
                break;
            }
            document.write("<BR/>Thank You!");
        </SCRIPT>
    </BODY>
</HTML>
```

81

JavaScript Loops

- Loops or iteration statements are control flow statements that allows you to execute group of statements several number of times.
- There are the following three loops provided by JavaScript:
  - for Loop
  - while Loop
  - do-while Loop

82

JavaScript Loops

for loop

Use when you know how many repetitions you want to do

syntax

```
for(initialize ; condition ; increment) { … }
```

example

```
for(x=0;x<10;x++) {
   // Code Here
}
```

while loop

Loops through block of code while condition is true.
The while loop in JavaScript is used when you want to check the condition at the start of the loop and then the group of statements that is to be executed is specified after the condition.

syntax

```
while(condition) { … }
```

example

```
while (x<10) {
    //Code Here   }
```

do while loop

Execute block at least once then repeat while condition is true

syntax

```
do{ … } while (condition);
```

example

```
do{
   // Code Here
} while (x<10)
```

83

Using For Loop

```
<!DOCTYPE HTML>
<HTML>
   <HEAD>
        <TITLE>Using for loop</TITLE>
   </HEAD>
   <BODY>
        <H1>Using the for loop in the script</H1>
        <SCRIPT type="text/javascript">
           var Number;
           document.write("Even numbers from 0 to 10 <BR/>");
          for(Number=0;Number<=10;Number=Number+2)
          {
                document.write(Number+"<BR/>");
           }
        </SCRIPT>
  </BODY>  </HTML>
```

84

Using While Loop

```
<!DOCTYPE HTML>
<HTML>
  <HEAD>
        <TITLE>Using while loop</TITLE>
  </HEAD>
  <BODY>
        <H1>Using while Loop in the Script</H1>
        <SCRIPT type="text/javascript">
           var totalDistance=0, fare=0;
            while(totalDistance<=10)
           {
                 fare=fare+5;
                 totalDistance=totalDistance+2;
                 document.write("Rs." + fare + " for " +
                 totalDistance + " KM distance. <BR/>");
            }
```

85

Using While Loop

```
           document.write("Thank You!");
        </SCRIPT>
    </BODY>
</HTML>
```

86

Using do…while Loop

```
<!DOCTYPE HTML>
<HTML>
    <HEAD>
        <TITLE>Using do...while loop</TITLE>
    </HEAD>
    <BODY>
        <H1>Using do...while Loop in the Script</H1>
        <SCRIPT type="text/javascript">
           var totalDistance=0, fare=0;
           do
           {
                document.write("Rs." + fare + " for " +
                totalDistance + " KM distance. <BR/>");
                document.write("<HR/>");
                fare=fare+5;
                totalDistance=totalDistance+2;
```

87

Using do…while Loop

```
                }
                while(totalDistance<=10);
        </SCRIPT>
   </BODY>
</HTML>
```

88

JavaScript Break and Continue Statement

- JavaScript break and continue statement/keyword is used to control the loop.
- **Break Statement**
  - JavaScript break statement is used to exit from the loop when the required condition is met.
- **Continue Statement**
  - JavaScript continue statement is used to skip the remaining part of code and goes to the condition-checker part, if the required condition is met.

Using Break Statement

```
<!DOCTYPE HTML>
<HTML>
  <HEAD>
        <TITLE>Using break Statement</TITLE>
  </HEAD>
   <BODY>
        <H1>Using break Statement in while loop</H1>
        <SCRIPT type="text/javascript">
              var count=0;
              while(count<10)
              {
                    ++count;
                    if((count%5==0))
                          break;
                    else
                    document.write("count="+count+"<BR/>");
```

Using Break Statement

```
                }

                document.write("while loop exited");
        </SCRIPT>
    </BODY>
</HTML>
```


Using Continue Statement

```
<!DOCTYPE HTML>
<HTML>
    <HEAD>
        <TITLE>Using continue Statement</TITLE>
    </HEAD>
    <BODY>
        <SCRIPT type="text/javascript">
                var count=0;
                while(count<10)
                {
                        ++count;
                        if(count%2==0)
                                continue;
                        else
```

Using Continue Statement

```
            document.write("count="+count+"<BR/>");
          }
          document.write("while loop exited");
      </SCRIPT>
    </BODY>
</HTML>
```

93

JavaScript Popup/Dialog Box

- Popup boxes can be used to raise an alert, or to get confirmation on any input or to have a kind of input from the users.
- JavaScript dialog box are of following three types:
  - JavaScript dialog box for getting confirmation on any user input (confirm Box)
  - JavaScript dialog box for raising an alert box (alert Box)
  - JavaScript dialog box for a kind of input from the user (prompt Box)
- There are three built-in methods of doing simple user interaction
  - **alert(msg)** alerts the user that something has happened
  - **confirm(msg)** asks the user to confirm (or cancel) something
  - **prompt(msg, default)** asks the user to enter some text

94

JavaScript Popup/Dialog Box

- alert("There's a monster on the wing!");
- confirm("Are you sure you want to do that?");
- prompt("Enter you name", "Adam");
- **JavaScript alert Box**
  - An alert box in JavaScript is mainly used to display an alert message while executing JavaScript code.
- **JavaScript confirm Box**
  - The confirm box in JavaScript is used to display a message and return a true or false value.
- **JavaScript prompt Box**
  - The prompt box in JavaScript contains a text box along with OK and Cancel buttons. It returns the text string when OK is clicked and null when Cancel is clicked.

95

Alert Box

- An alert box is used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will **have to click "OK"** to proceed.
- It can be used to display the result of validation.

Code

```
<html>
    <head>
        <title>Alert Box</title>
    </head>
    <body>
        <script>
            alert("Hello World");
        </script>
    </body>
</html>
```

96

Using Alert Box

```
<!DOCTYPE HTML>
<HTML>
        <HEAD>
                <TITLE>Using alert box</TITLE>
        </HEAD>
        <BODY>
                <H1>Showing the alert box</H1>
                <SCRIPT type="text/javascript">
                        alert( "Hello, I am an alert box." );
                </SCRIPT>
        </BODY>
</HTML>
```

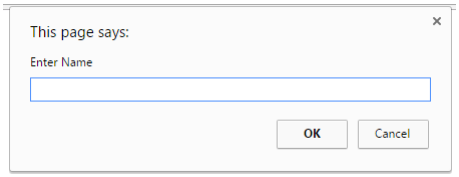
97

Prompt Box

- A prompt box is used if you want the user to input a value.
- When a prompt box pops up, user have to click either "OK" or "Cancel" to proceed, If the user clicks "OK" the box returns the input value, If the user clicks "Cancel" the box returns null.

Code

```
<script>
   var a = prompt("Enter Name");
   alert("User Entered " + a);
</script>
```



98

Using Prompt Box

```
<!DOCTYPE HTML>
<HTML>
   <HEAD>
      <TITLE>Using prompt Box</TITLE>
   </HEAD>
   <BODY>
      <H1>Accepting your name using prompt box</H1>
      <SCRIPT type="text/javascript">
            var name= prompt( "Please enter your name: " );
            if (name==null || name=="") name = " visitor ";
            {
                  document.write ("Hi " + name + ", Welcome
                  to JavaScript");
                  }
      </SCRIPT>
   </BODY>   </HTML>
```

99

Confirm Box

- A confirm box is used if you want the user to accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed, If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Code

```
<script>
  var a = confirm("Are you sure??");
  if(a==true) {
        alert("User Accepted");
  }
  else   {
        alert("User Cancled");
  }
</script>
```

100

Using Confirm Box

```
<!DOCTYPE HTML>
<HTML>
        <HEAD>
                <TITLE>USING CONFIRM BOX</TITLE>
        </HEAD>
        <BODY>
                 <SCRIPT type="text/javascript">
                 var qry= confirm ( "Are you sure to send this file to
                Recycle Bin ? " );
                 if (qry)
                 document.write ("Welcome to JavaScript. <BR/>
                 You have moved the file to Recycle Bin.");
                 else
                 document.write ("Welcome to JavaScript <BR/>
                The file is not moved");
                 </SCRIPT>
        </BODY>
</HTML>
```

101

102