Introduction to Java Script

G. Karthika

Overview of Java Script:

Origin:

Java Script initially named as Live Script which is developed in the year 1995 by Netscape and Microsoft and later its named is changed as Java Script with version 1.0 to version 1.5 by adding many new features.

* A language standard for Java Script is developed in 1990's by European Computer Manufactures association named as ECMA Script used in Internet Explorer and firefox.
* ECMA Script version 3 is similar to Java Script version 1.5.

Java Script is divided into 3 types.

① Core ② Client Side ③ Server side.

Core: It is heart of language, including its operators, Expressions, Statements and Sub programs.

Client Side: It is a collection of objects that Supports control of browsers and interaction with users.

Server Side: is Collection of objects that is useful for web server which means it helps server to communicate with DBMS.

* Server side Javascript is used for less frequently than client side JS

* Client-Side JavaScript is an HTML embedded Scripting language and its code is referred as a Script.

Difference B/w Java & JavaScript.

Java	JavaScript
① Java Supports Object Oriented programming.	① It support object Oriented programming, but the model don't follow object oriented Software develop ment paradigm.
② The data types and operand types are all known at compile time.	② The variables need not declared and compile time check is impossible
③ Objects are static ie., data members and methods are fixed at compile time.	③ Objects are dynamic here, the data members and methods can change during execution.

* Both the Javaa and JavaScript Syntax are differs from each other but main similarity is the Syntax of their expressions, assignment Stmts and control Stmts.

* JavaScript, checks the values and Transmit to server for processing and same followed by forms.

JavaScript → forms → checks values → transmit to Server for processing

Uses of JavaScript:

① The JavaScript is used to provide programming capability at both server and client for web connection.

② It can be used as an alternative to Java Applet

③ JavaScript are integral part of HTML, so that, no secondary downloading is necessary.

④ User interaction through elements such as Buttons, and menus are convenient in JavaScript.

⑤ The development of document object model by JavaScript allows access and modify the css properties and content of any element of a displayed HTML document, by that making a static document to highly dynamic.

⑥ Java Script also do Event Driven Computation, which means actions executed in response to actions of user document.

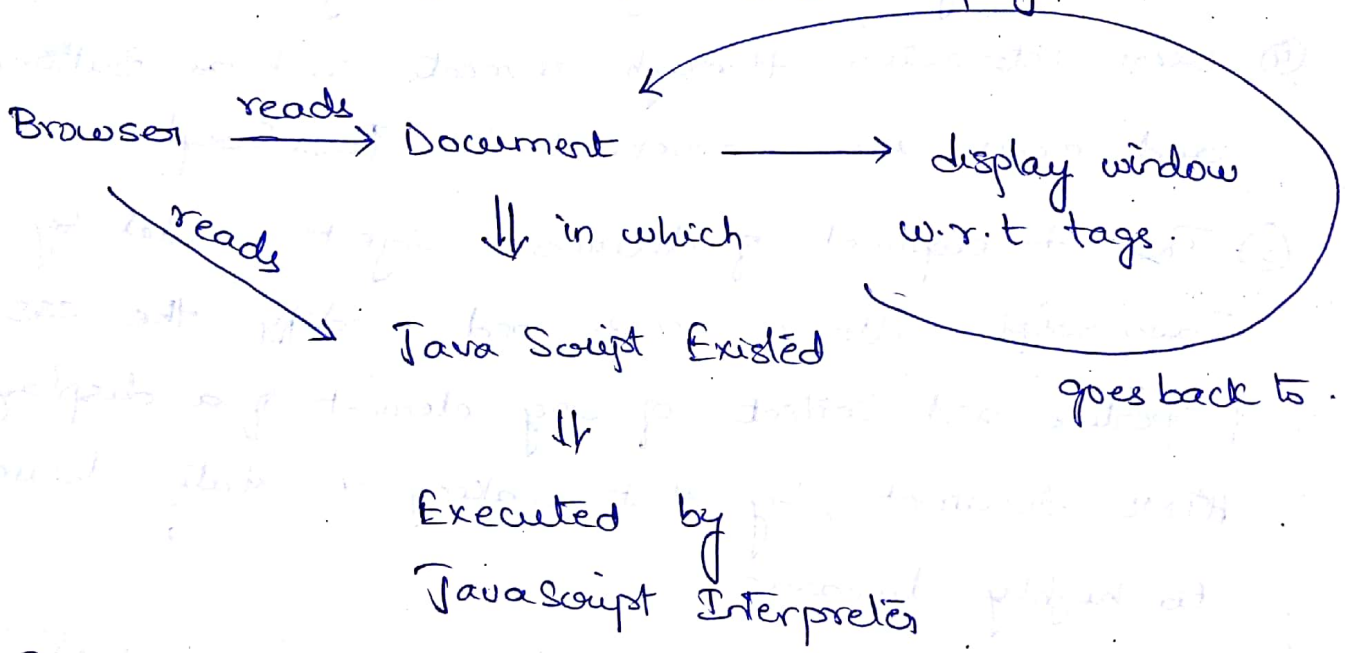
⑦ JavaScript used to check values provided in forms, by users by transmitting to the server for processing.

* HTML pages are static (CSS) and using JavaScript is a dynamic page creation.

HTML pages → Static (CSS) ⟶ Using JavaScript → Dynamic

Browsers and HTML / JavaScript Documents:

* Browser reads the lines of document and gives its window according to the tags.
* when the browser reads JavaScript in the document, it uses Java Script Interpreter to Execute the JavaScript.
* After its execution, the browser goes back to read the HTML document and display its content.



* Java Script Scripts can appear in either part of an HTML document, the head or body, depending on the purpose of the script.
* Script in Head of document:

which produce the content only when requested or that react to user interactions, means fn definitions & code associated with from elements like buttons

* Script in Body of document:

Scripts that are to be interpreted just once, when the interpreter finds them.

* The interpreter notes the existence of scripts that appear in the head of a document, but it does not interpret them while processing the head.
* Scripts that are found in the body of a document are interpreted as they are found.

Event-Driven Computation:

One of the common uses of JavaScript is to check the values provided in forms by users to determine whether the values are sensible.

* Without client-side checks of such values, form values must be transmitted to the server for processing without any prior reality checks.
* The program or script on the server that processes the form data must check for invalid input data.
* when invalid data is found, the server must transmit that information back to the browser, which then must ask the user to resubmit corrected input.
* It is more efficient to perform input data checks and carry on this user dialog entirely on the client.
* It saves both server time and Internet time.

* One example, if the data is to be put in a database where invalid data could corrupt the database.

General Syntactic Characteristics:

Scripts can appear as the content of a <script> tag. The type attribute of <script> must be set to "text/javascript".

* JavaScript script indirectly embedded in HTML document using src attribute of <script> tag, value is name of a file that contains script.

Eg: <script type="text/javascript" src="file1.js">
</script>

* The indirect method of embedding javascript in HTML documents has the advantage of hiding the script from the browser user.

* In JavaScript, identifiers or names are similar to those of other common programming language.

* They must begin with a letter, an underscore (_) or a dollar sign ($).

* There is no length limitation for identifiers.

* JavaScript has 25 reserved words.

They are:

break	delete	function	return
case	do	if	switch
catch	else	in	this
continue	finally	instance of	throw
default	for	new	try

typeof var void while with.

* JavaScript has a large collection of pre-defined words, including alert, open, java & self.

JavaScript has 2 forms of comments:

① Whenever 2 adjacent slashes (//) appear on a line, the rest of the line is considered a comment.

② both single - and multiple - line comments can be written using /* */ to terminate it.

Two issues regarding embedding JavaScript:

① There are some browsers still in use that recognize the <script> tag but do not have JavaScript interpreter. Browser simply ignore the contents of script elements.

② There are still a few browsers in use that are so old they do not recognize the <script> tag. Browsers will display the contents of script element as text.

* HTML validator also has a problem with embedded JavaScript

Eg:
 tag is embedded in JavaScript then in the output of JavaScript causes validation errors. We enclose it using comments.

<!-- : works as a hiding prelude to JavaScript code.

closing of comment : // -->.

Eg: <!--
-- Javascript script --
//-->.

* The use of semicolons in JavaScript is unusual.
* When the end of a line coincides with what could be the end of a stmt, the interpreter effectively inserts a semicolon there.
* It leads to a problem

Eg: return
x;

* The interpreter puts a semicolon after return, making x an illegal.
* The safest way to organize JavaScript Stmt for each line and terminate each stmt with a semicolon
* If Stmt not fit in one line, then to break the stmt at a place ensure the first line does not have the form of a complete stmt.
* One line of JavaScript in the document, the call to write through the document object to display the message.

Eg:
```
<html>
<head>
<title> Hello world </title>
</head>
```

```
<body>
  <script type = "text/JavaScript">
  <!--
    document.write("Hello, fellow web programmer!");
    //-->
  </script>
</body>
</html>.
```

Primitives, Operations & Expressions:

Primitive Types:

Java script has 5 primitive types

① Number ② String ③ Bodean ④ Undefined ⑤ Null.

Wrapper Objects:

Java-Script includes predefined objects that are closely related to the Number, String & Bodean types named Number, String and Bodean.

* Each contains a property that stores a value of corresponding primitive type.

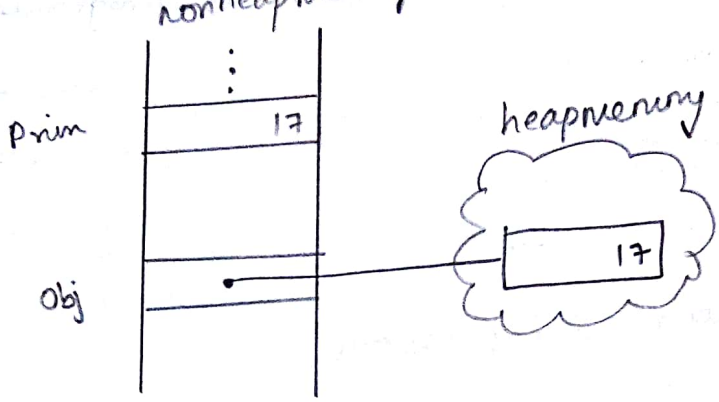
Purpose of Wrapper Object:

To provide properties and methods that are convenient for use with values of the primitive Types.

* Number: Uses more properties
* String: Uses more methods.

The difference b/w primitives & objects:

Eg: prim is a primitive variable with the value 17 and obj is a Number object whose property value is 17.



Numeric & String Literals:

Numeric Literals are values of type Number.

* These values of JavaScript is represented in double-precision floating-point form.
* Literal numbers in a script can have the forms of either integer or floating-point values.
* Integer literals are strings of digits
* floating-point literals can have decimal points or exponents or both.
* Exponents are specified with an uppercase or lowercase e and a signed integer literal.

Legal numeric literals are:

72 7.2 .72 72. 7E2 7e2 .7e2 7.e2 7.2E-2

* Integer literals can be written in hexadecimal form by preceding their first digit with 0x or 0X (zero).

String Literals:

It is a sequence of zero or more characters delimited by either single quote (' ') or double quote (" ")

* To access the single quote (') character in a string literal, the embedded single quote must be preceded by a backslash:

'You\'re the most freckly person I've ever met'

o/p: You're the most freckly person I've ever met.

* There is no difference blw single quoted string literal and double quoted.

Other Primitive Types are

① Null, ② undefined, ③ boolean.

* The value of type Null is reserved word null, shows no value.

* If variable is null, if it is not declared or assigned value, and tries to use value then a run-time error is caused.

* If a variable has explicitly declared but not assigned a value then it has a undefined value.

* The value of type Undefined is undefined.

* If a value of undefined type variable is displayed the word 'undefined" is displayed.
* The only values of type Boolean are true & false.
* These values are computed as a results of evaluating a relational or boolean expression.
* The existence of both boolean primitive type & the boolean object can lead to some confusion.

Declaring Variables:

A variable can have the value of any primitive type, or it can be reference to any object.

* A variable can be declared either by assigning it a value, in which the interpreter implicitly declares it to be a variable, or listing it in a declaration stmt that begins with the reserved word "var".

Eg:
```
var counter,
    index,
    pi = 3.14159265,
    quarterback = "Elway",
    stop_flag = true;
```

* A variable is declared, but not assigned a value has the value undefined.

examples:

→ Undefined as o/p:

```
<html>
<head> </head>
<body>
<p> javascript used for dynamic webpages
</p>
<script>
    var x;
    var y=5;
    document.write (x + "<br>" +y);
</script>
</body> </html>
```

o/p:

javascript used for dynamic webpages

undefined

5.

→ Sum of 2 numbers: o/p: 9.

```
<html> <body>
<script>
    var x = 4;
    var y=5;
    document.write ("The sum is ", x+y);
</script> </body>
</html>.
```

String literal with (') & (") quotes.

```
<html> <body>
  <script>
    document.write("you're the most freckly person I've ever met");
  </script> </body> </html>
```

O/P: you're the most freckly person I've ever met.

~~Numeric Operator~~:

→ String as a value.

```
<html>
<body> <style> h2 { color: green; font-size: 10pt; } </style>
  <h2> JavaScript Examples </h2>
  <script>
    var a = "hello";
    var b = "world";
    document.write(a+b); (or)
    document.write(a+"\n"+b);
  </script> </body> </html>.
```

O/P: helloworld

O/P: hello world

O/P: JavaScript Examples → green color & 10pt).
hello world

Numeric Operators:

JavaScript has a collection of numeric operators.

The Binary operators: + (addition), - (subtraction), *, /, %.

The Unary operators: +, -, ++, --

(+ : plus, - : negate)

The increment & decrement operators can be prefix or postfix.

Eg:
```
<script>
var a = 7;
document.write ((++a) * 3);
</script>
```

O/p: 24.

Eg:
```
<script>
var a = 7;
document.write ((a++) * 3);
</script>
```

O/p: 21 (but a value is set to 8 for both egs.)

* All numeric operations are done in double-precision floating point.

* Precedence Rules:

Specifies which operator is evaluated first when 2 operators with diff. precedence are adjacent in an expression.

Eg: a * b + 1

Eg: O/P:

```
<html> <body>
<script>
var x = 7;
var y = 8;
document.write(x * y + 3);
</script>
</body> </html>
```

Associativity rules:

Specifies which operator is evaluated first when 2 operators with same precedence are adjacent in an expression.

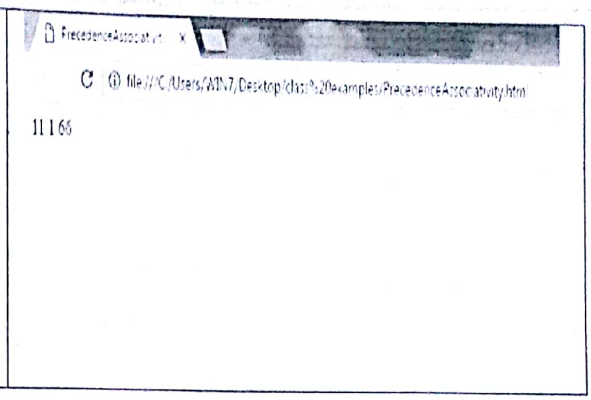
Precedence & associativity of the numeric operators

Operator	Associativity
++, --, unary -, unary +	Right (though it is irrelevant)
*, /, %	left
Binary +, binary -	left.

Eg: of precedence & associativity

```
<html> <body> <script>
var a = 2;
var b = 4;
var c, d;
```

```
<html>
<body>
<script>
var a=2, b=4, c, d,e;
c=3+a*b;
d=b/a/2;
e=(a+b)*c;
document.write(c+"\n"+d+"\n"+e);
</script>
</body>
</html>
```



The Math Object :

The Math object provides a collection of properties of Number Objects and methods that operate on Number objects.

* Math object has methods for the trigonometric fns such as sin & cos.
* Other mathematical operations are :

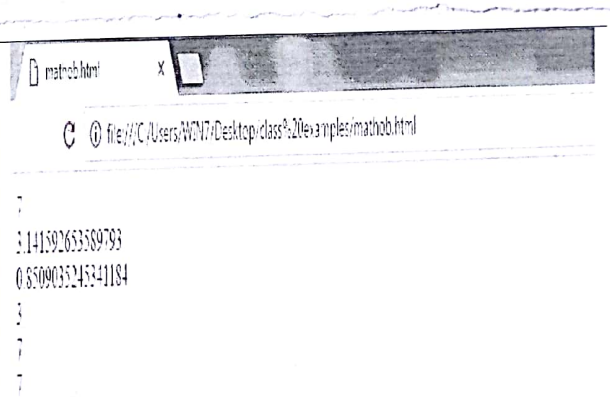
floor : to truncate a number

Round : To round a number

Max : To return a largest of 2 given numbers.

All the Math methods are referenced through the

```
<html>
<body>
<script>
var a=7,b=5;
document.write(Math.round
(7.4)+"<br>"+Math.PI+"<br>"+Math.sin
(45)+"<br>"+Math.sqrt(9)+"<br>"+Math.ma
x
(a,b)+"<br>"+Math.floor(7.88));
</script>
</body>
</html>
```



ject </h2>

59
PI + "\n"
:8) + "\n".
.b));
h object
0 7 2 7

The Number Object:

The number object includes a collection of useful properties that have constant values.

Properties of Number:

Property	Meaning	value
MAX_VALUE	Largest representable number	1.7976931348623157e + 308
MIN_VALUE	Smallest representable number	5e -324
NaN	Not a number	-
POSITIVE_INFINITY	Special value to represent infinity	Infinity
NEGATIVE_INFINITY	Special value to represent negative infinity	-Infinity
PI	The value of π	3.14

* To determine whether a variable has a NaN value the predefined predicate function is NaN() must be used.

Eg:
```
<html>
<body>
<h2> Number object Example </h2>
<script>
```

```
<html>
<body>
<h2> number object example</h2>
<script>
var res=" ";
res= res+isNaN(123)+":123<br>";
res= res+isNaN(0)+":0<br>";
res= res+isNaN("hellow")+":hellow<br>";
res= res+isNaN(false)+":false<br>";
res= res+isNaN('NaN')+":NaN<br>";
document.write(res);
document.write(Number.MAX_VALUE

+"<br>"+Number.MIN_VALUE

+"<br>"+Number.POSITIVE_INFINITY

+"<br>"+Number.NEGATIVE_INFINITY);
</script>
</body>
<html>
```

numobj.html

file:///C:/Users/MINI7/Desktop/class%20examples/numobj.html

number object example

false 123
false 0
true hellow
false false
true NaN
1.7976931348623157e+308
5e-324
Infinity
-Infinity

er. MIN_VALUE

+ "\n" + Number.POSITIVE_INFINITY + "\n" +

Number.NEGATIVE_INFINITY);

</script> </body> </html>.

O/P: Number object Example.

false : 123

false : 0

true : 'Hello'

false : true

true : NaN

1.7976931348623157e+308 5e-324 Infinity -Infinity

* The Number object has a method, 'toString', which it inherits from Object but overrides.
* The toString method converts the number through which it is called to a string
* Because, numeric primitives and Number objects are always coerced to the other when necessary.

toString can be called through numeric primitive.

Eg:
```
<html>
<body>
<script>
    var price = 427, str_price;
    str_price = price.toString();
    document.write(str_price);
</script>
</body> </html>.
```

O/P: 427.

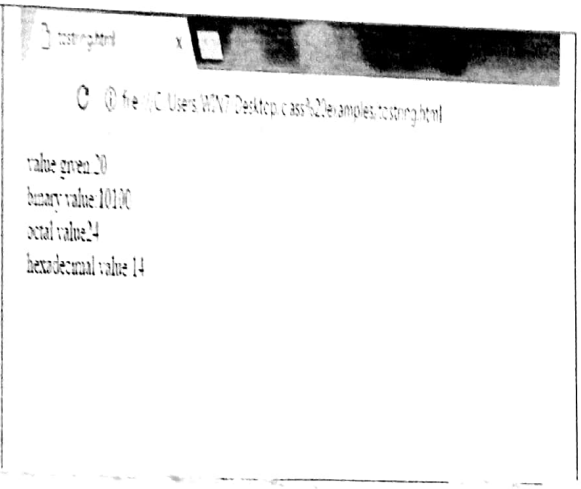
Radix: (Parameter values).

Optionally used, base to use for representing a numeric value.

2 - The number will show as a binary value
8 - The number will show as an octal value
16 - The no. will show as an hexadecimal value.

Eg:
```
<html>
<body>
<script>
    var num=20;
    var a = num.toString();
    var b = num.toString(2);
    var c = num.toString(8);
    var d = num.toString(16);
```

```
<html>
<body>
<script>
var num=20;
var a=num.toString();
var b=num.toString(2);
var c=num.toString(8);
var d=num.toString(16);
document.write("value given:",a +"<br>"+"binary
value:",b+"<br>"+"octal
value",c+"<br>"+"hexadecimal value:",d);
</script>
</body>
</html>
```



+d;

24

14.

String Concatenation Operator:

Javascript strings are not stored or treated as arrays of characters.

* String concatenation is specified with the operator denoted by (+) sign

Eg:
```
<html> <body>
<script>
var first = "freddie";
document.write(first + "freeloader");
</script> </body> </html>
```

O/P: freddie freeloader.

Concatination of a string:

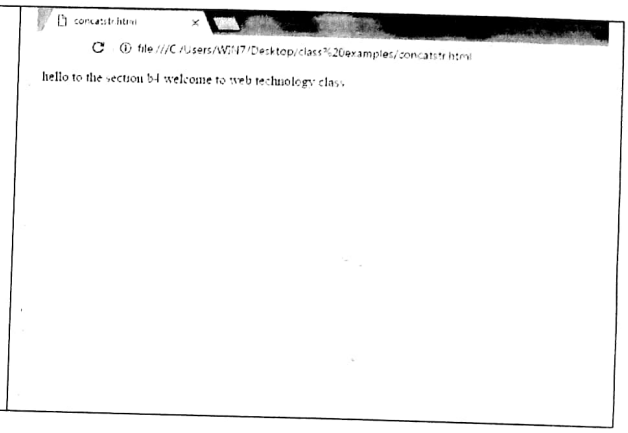
```
<html> <body>
<script>
var str1 = "Hello!";
var str2 = "Section B10, B4";
```

```
var str2 = " Welcome to the class";
var kar = str1.concat(str2, str3);
document.write(kar);
</script></body></html>.
```

O/P: Hello! Section B4, By Welcome to the class.

Other Eg:

```
<html>
<body>
<script>
 var str1="hello ";
var str2="to the section b4";
var str3=" welcome to web technology class";
var kar=str1.concat(str2,str3);
document.write(kar);
</script>
</body>
</html>
```



String Properties and Methods:

String methods can be used through String primitive values, as if the values were objects.

* The String object includes one property, length and a large collection of methods.
* The number of characters in a string is stored in the length property as:

```
var str = "George";
var len = str.length;
```

Explanation: len is set to no. of characters in str: 6

* The expression str.length, str is a primitive variable, but treated as an object.

* when str is used with the length property, JavaScript implicitly builds a temporary String object with a property whose value is that of the primitive variable.
* The second stmt is executed, the temporary String object is discarded.

String Methods:

Method	Parameters	Result:
charAt	A number	Returns the character in the String object that is at the specified position.
indexOf	One-character String	Returns the position in the String object of the parameter
substring	Two numbers	Returns the substring of the String object from the first parameter position to the second.
toLowerCase	None	Converts any uppercase letters in the string to lowercase.
toUpperCase	None	Converts any lowercase letters in the string to uppercase.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript</h2>
<p>String Properties and Methods</p>
<p id="demo"></p>
<script>
var str = "Robert W.SebEstA Robert";
var len=str.length;
var sri=str.charAt(2);
var srii=str.indexOf('b');
var sriii=str.substring(2,4);
var sri1=str.toLowerCase();
var sri2=str.toUpperCase();
var pos = str.lastIndexOf("Robert");
document.write(len+"<br>"+sri+"<br>"+
srii+"<br>"+sriii+"<br>"+ sri1+"<br>"+
sri2+"<br>"+ pos );
</script>
</body>
</html>
```

string FM.html

file:///C:/Users/MTN17/Desktop/class%20examples/string%20FM.html

JavaScript

String Properties and Methods

23
b
2
be
robert w.sebesta robert
ROBERT W.SEBESTA ROBERT
17

The typeof Operator:

The typeof operator returns the type of its single operand. typeof evaluates to "number", "string" or "boolean", if the operand is of primitive type Number, String or Boolean.

* If the operand is an object or null, typeof evaluates to "Object".

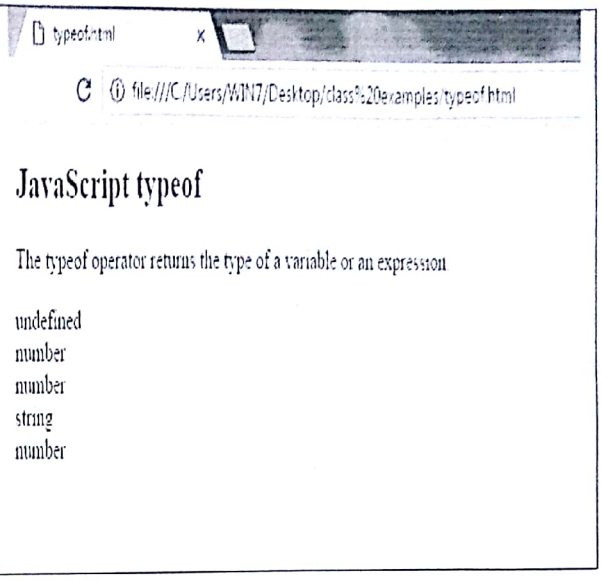
Fundamental characteristic of JavaScript - Objects don't have types.

* If the operand is a variable that has not been assigned a value, typeof evaluates to "undefined".
* The typeof operator always returns a string.

Syntax: typeof x and typeof(x) are equivalent.

Eg:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript typeof</h2>
<p>The typeof operator returns the type of a variable or an expression.</p>
<script>
document.write(
typeof a + "<br>" +
typeof 314 + "<br>" +
typeof 3.14 + "<br>" +
typeof "karthi" + "<br>" +
typeof (3 + 4));
</script>
</body>
</html>
```



Assignment Statements:

Assignment statement in JavaScript is like the assgn. stmt of C-progn. lang.

* A simple assignment operator, denoted by =,
  A host of compand assignment operators, += & /=

Eg: the stmt.

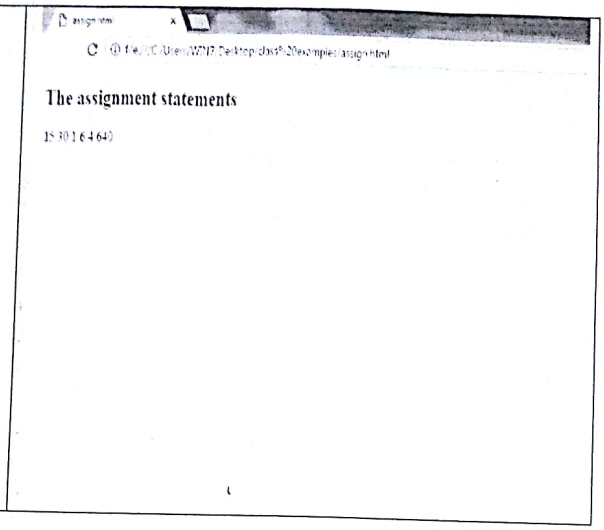
a += 7; means a = a + 7;

JavaScript has 2 kinds of values:

Primitives & objects.

* Objects are allocated on the heap, and variables that refer to them are reference variables.
* To refer to an object, a variable stores an address only.
* Assigning the address of an object to a variable is fundamentally diff. from assigning the primitive value to a variable.

```
<!DOCTYPE html>
<html>
<body>
<h2>The assignment statements </h2>
<script>
var x = 10,y=6,z=8,a=9;
b=x<<y;
x += 5;
y *= 5;
z /= 5;
a %= 5;
document.write(x+"\n"+y+"\n"+z+"\n"+a+"\n"+b)
;
</script>
</body>
</html>
```



The Date Object :

The information about the current date and time is useful in a program. It is convenient to be able to create objects that represent a specific date and time.

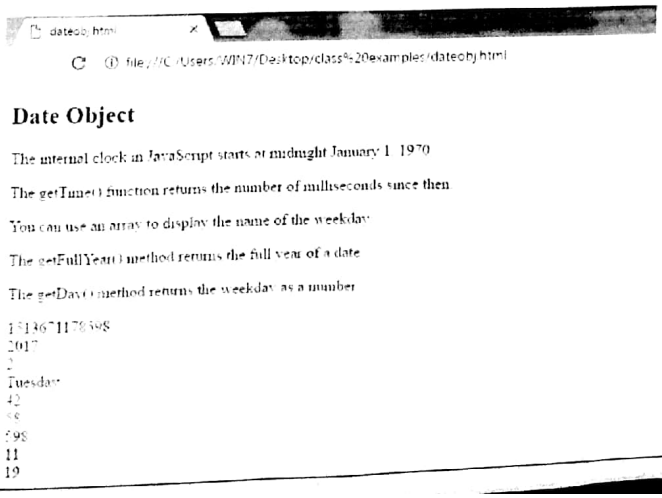
* This is available in JavaScript through the Date object and collection of methods.
* A Date object is created, with the new operator and the Date constructor.
* The simplest Date constructor which takes no parameters and builds an object with the current date and time for its properties.

Methods for the Date object :

Method	Returns
toLocaleString	A string of the Date information
getDate	The day of the month.

getMonth	The month of the year, as a number in the range of 0 to 11
getDay	the day of the week, as a number in the range of 0 to 6
getfullYear	The year
getTime	The number of milliseconds since January 1, 1970
getHours	The no. of hours, as a number in range of 0 to 23.
getMinutes	The no. of the minute, as a number in range of 0 to 59
getSecords	The no. of the second, as a number in range of 0 to 59

```
<!DOCTYPE html>
<html>
<body>
<h2> Date Object </h2>
<p>The internal clock in JavaScript starts at midnight January 1, 1970.</p>
<p>The getTime() function returns the number of milliseconds since then:</p>
<p>You can use an array to display the name of the weekday:</p>
<p>The getFullYear() method returns the full year of a date:</p>
<p>The getDay() method returns the weekday as a number:</p>
<script>
var d = new Date();
var days = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"];
document.write(d.getTime()+"<br>"+d.getFullYear()+"<br>"+d.getDay()+"<br>"+days[d.getDay()]+"<br>
"+d.getMinutes()+"<br>"+d.getSeconds()+"<br>"+d.getMilliseconds()+"<br>"+d.getMonth()+"<br>"+d.g
etDate());
</script>
</body>
</html>
```



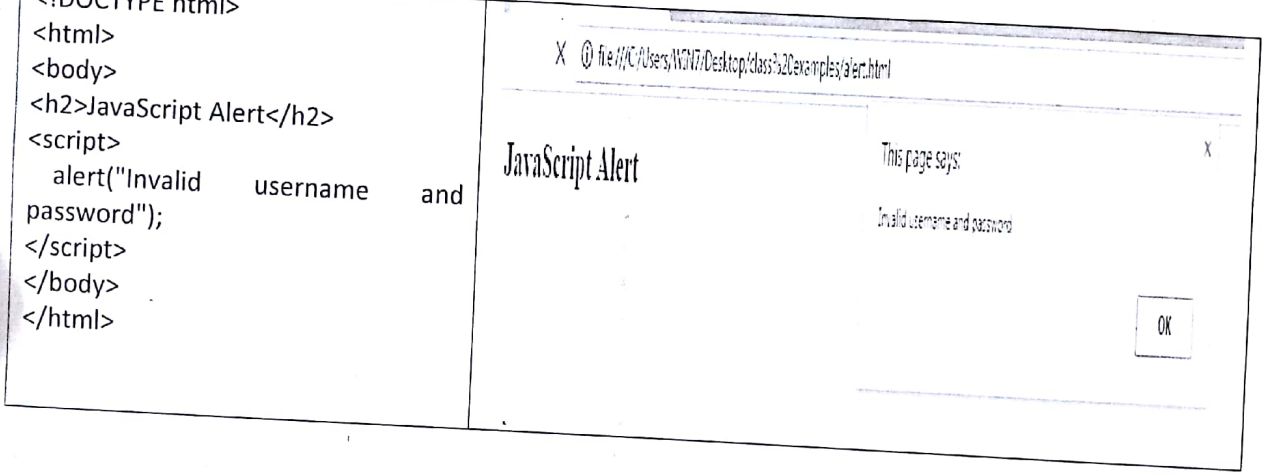
Screen Output and Keyboard Input:

Alert: The alert method opens a dialog window and displays its parameter in the window.

* It also displays an OK button.
* The parameter string to alert is not HTML code, it is plain text.
* The string parameter to alert may include \n but never should include
.

Eg:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Alert</h2>
<script>
  alert("Invalid username and password");
</script>
</body>
</html>
```

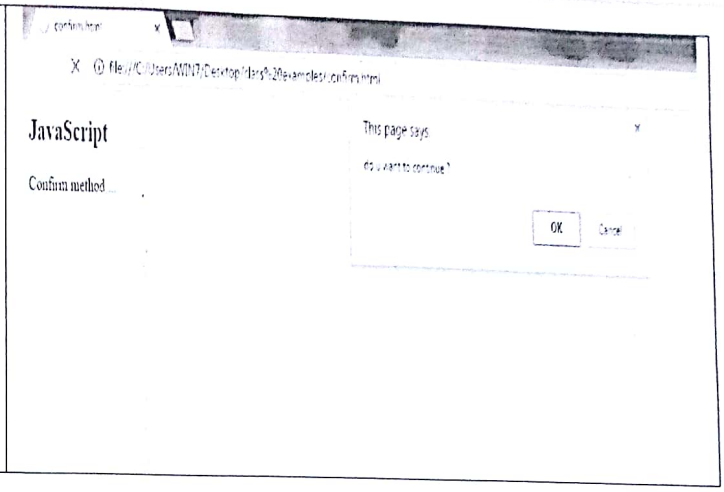


Confirm: The confirm method opens a dialog window in which it displays its string parameter, with 2 buttons OK & Cancle.

* confirm returns a Boolean value that ~~indicates~~ indicates the user's button input:
  true for OK and false for Cancel.
* This method is often used to offer the user the choice of continuing some process.

Eg:

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

<h2>JavaScript</h2>
<p>Confirm method....</p>
<script>
var question=confirm("do u want to
continue ?");
</script>
</body>
</html>
```



Prompt: The prompt method creates a dialog window that contains a text box.

* The text box is used to collect a string of input from the user, which prompt returns as its value.
* The window has 2 buttons OK and cancel.

Two parameters:

* The string that prompts the user for input
* A default string in case the user does not type a string before pressing one of the two buttons.

Eg: name = prompt("what is your name?", " ");

* An empty string is used for the default input.

→ Alert, prompt and confirm cause the browser to wait for a user response.

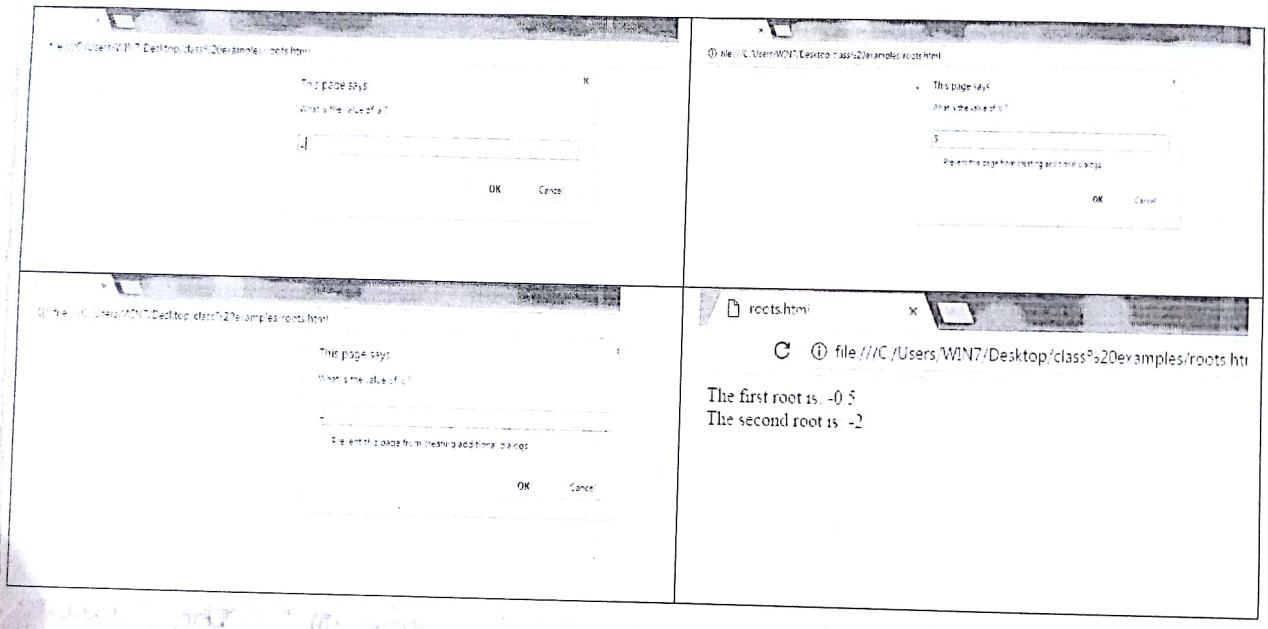
→ alert: OK button must be pressed for the JavaScript interpreter to continue.

→ Promp & confirm : OK or Cancel to be pressed.

```
<html >
 <head>
  <title> roots.html </title>
 </head>
 <body>
  <script type = "text/javascript"  src = "roots.js" >
  </script>
 </body>
</html>
```

roots.js

```
var a = prompt("What is the value of 'a'? \n", "");
var b = prompt("What is the value of 'b'? \n", "");
var c = prompt("What is the value of 'c'? \n", "");

var root_part = Math.sqrt(b * b - 4.0 * a * c);
var denom = 2.0 * a;

var root1 = (-b + root_part) / denom;
var root2 = (-b - root_part) / denom;
document.write("The first root is: ", root1, "<br
/>");
document.write("The second root is: ", root2, "<br
/>");
```



Control Statements :

The flow - control stmts of JavaScript

* Control stmts often require some syntactic containers for sequences of stmts whose execution they are meant to control.
* A compound Stmt : in JavaScript is a sequence of stmts delimited by braces.
* A control construct : is a control Stmt and the Stmt or compound Stmt whose execution it controls.

Control Expressions:

The expressions upon which stmt flow control can be based include primitive values, relational expressions, and compand expressions.

* The result in boolean values true or false are evaluated in control expressions.
* The interpreted as true if it is empty string (" ") or a zero string ("0")
* The value is a number, it is true unless it is zero (0).
* The relational expression has 2 operands & one relational operator.

Relational operators:

operation	operator
Is equal to	==
Is not equal to	!=
Is less than	<
Is greater than	>
Is less than or equal to	<=
Is greater than or equal to	>=
Is strictly equal to	===
Is strictly not equal to	!==

* In type conversion, the last 2 operands operators are disallowed.

* The expression "3" ===3 evaluates to false.
"3" == 3 evaluates to true.

* JS has operators for AND, OR and NOT Boolean operations.

* These are && (AND), || (OR), and ! (NOT) used.

→ The properties of the object Boolean must not be confused with the primitive values true & false.

For conditional Expression: The boolean object is used and evaluates to true if it has any value other than null or undefined.

Boolean object has a method: "toString".

It inherits from objects, that convert the value of the object through which it is called one of the string "true" or "false".

Operator Precedence and Associativity

Operators	Associativity
++, --, unary -	Right
*, /, %	left
+ -	"
<, >, >=, <=	"
==, !=	"
===, !==	"

&& || =, +=, -=, *=, /=, &&=, ||=, %=	left left. Right

Selection Statements:

The selection stmt (if-then and if-then-else) of JS are similar to those of the common prog. lang.

* Single or compound stmt can be selected.

Eg:
```
if (a > b)
    document.write("a is greater than b <br/>");
else
{
    a = b;
    document.write("a was not greater than b <br/>",
        "Now they are equal <br>");
}.
```

The Switch Statement:

* JavaScript has a Switch stmt similar to C.

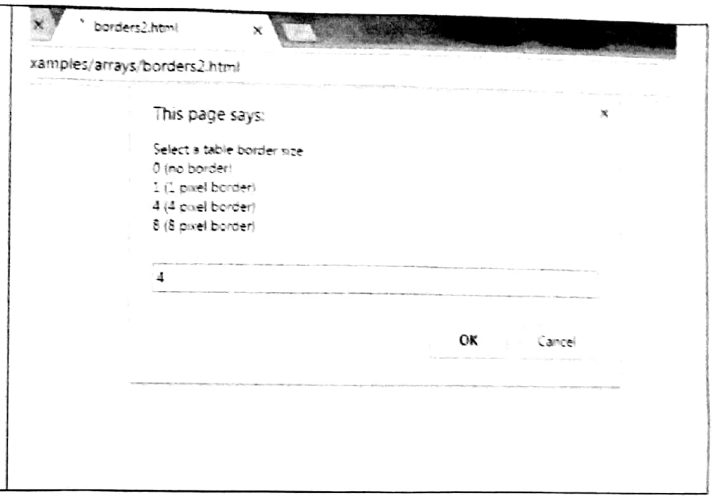
Syntax:
```
switch (expression)
{ case value-1:
      // stmts
  case value-2:
      // stmts
  ...
  [default; // stmts ]
}
```

* In case segment, the segments can be either a stmt sequence or compound stmt.

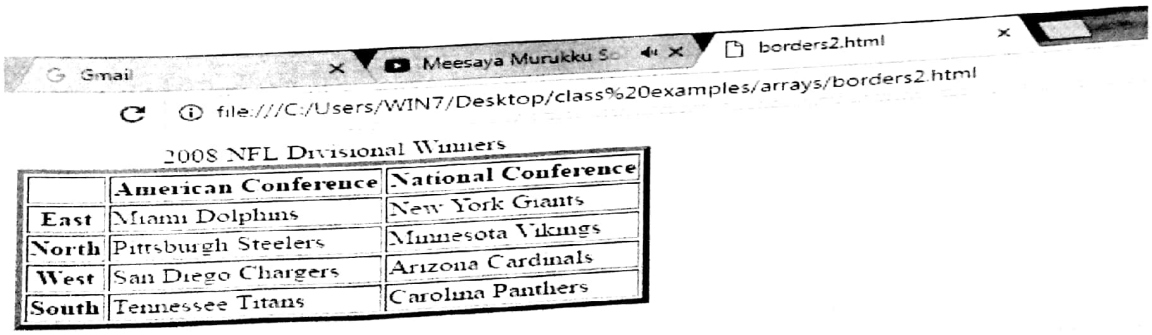
Semantics of switch construct:

* When switch stmt is executed the expression is evaluated.
* The value compared to the values in the cases.
* If it matches, control is tranferred to stmts immediately following that case value.
* Execution continues through the remainder of construct.
* The break stmt appears as last stmt in each sequence of stmts.
* It (break stmt) transfers control out of compound stmt. in which it appears.
* The control expressions of a switch stmt could evaluate to a number, a string or a Boolean value.
* Case labels also can be numbers, strings or Booleans, and diff case values can be of diff. types.

```
<!-- borders2.html
   A document for borders2.js
   -->
<html >
 <head>
  <title> borders2.html </title>
 </head>
 <body>
  <script type = "text/javascript"  src =
"borders2.js" >
  </script>
 </body>
</html>
```




```
// borders2.js  An example of a switch statement for table border size selection
var bordersize;
bordersize = prompt("Select a table border size \n" +
            "0 (no border) \n" +
            "1 (1 pixel border) \n" +
            "4 (4 pixel border) \n" +
            "8 (8 pixel border) \n");
switch (bordersize) {
 case "0": document.write("<table>");
        break;
 case "1": document.write("<table border = '1'>");
        break;
 case "4": document.write("<table border = '4'>");
        break;
 case "8": document.write("<table border = '8'>");
        break;
 default:  document.write("Error - invalid choice: ",
                bordersize, "<br />");
}
document.write("<caption> 2008 NFL Divisional",
         " Winners </caption>");
document.write("<tr>",
         "<th />",
         "<th> American Conference </th>", "<th> National Conference </th>", "</tr>",
         "<tr>",
         "<th> East </th>","<td> Miami Dolphins </td>",  "<td> New York Giants </td>","</tr>",
         "<tr>",
         "<th> North </th>","<td> Pittsburgh Steelers </td>",  "<td> Minnesota Vikings </td>","</tr>",
         "<tr>",
         "<th> West </th>","<td> San Diego Chargers </td>",  "<td> Arizona Cardinals </td>","</tr>",
         "<tr>",
         "<th> South </th>",  "<td> Tennessee Titans </td>", "<td> Carolina Panthers </td>","</tr>",
         "</table>");
```



2008 NFL Divisional Winners

	American Conference	National Conference
East	Miami Dolphins	New York Giants
North	Pittsburgh Steelers	Minnesota Vikings
West	San Diego Chargers	Arizona Cardinals
South	Tennessee Titans	Carolina Panthers

Loop Statements:

The loop stmts are while, dowhile, for are same in Javascript as of other languages.

while stmt: Syntax:

```
while (control expression)
    stmt or compound stmt.
```

for stmt Syntax:

```
for (initial expression; control expression;
                         increment expression)
    Stmt or comp. stmt.
```

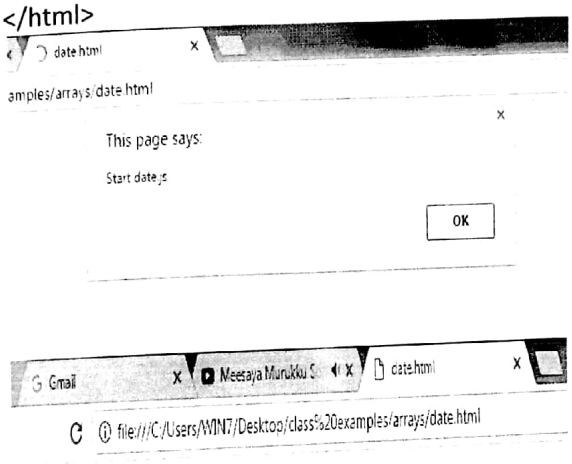
Eg: for

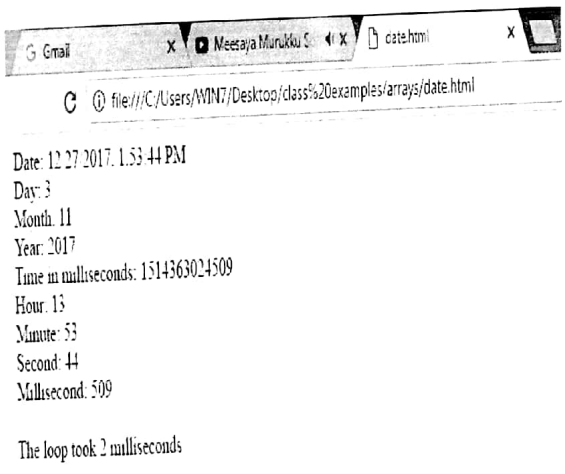
```
var sum = 0, count;
for (count = 0; count <= 10; count++)
    sum += count;
```

* Both the initial expression & the increment expression can be multiple expressions, seperated by commas.
* The initial expression of a for stmt can be include variable declarations.
* Such variables are visible in the entire script unless for stmt is in a function definition.

Example Illustrates the Date object and a simple for loop.

```
<!-- date.html
   A document for date.js
   -->
<html>
 <head>
  <title> date.html </title>
 </head>
 <body>
  <script type = "text/javascript"  src = "date.js">
  </script>
 </body>
</html>
```






```
// date.js
//  Illustrates the use of the Date object by
//  displaying the parts of a current date and
//  using two Date objects to time a calculation
// Get the current date
alert("Start date.js");
    var today = new Date();
// Fetch the various parts of the date
    var dateString = today.toLocaleString();
    var day = today.getDay();
    var month = today.getMonth();
    var year = today.getFullYear();
    var timeMilliseconds = today.getTime();
    var hour = today.getHours();
    var minute = today.getMinutes();
    var second = today.getSeconds();
    var millisecond = today.getMilliseconds();
// Display the parts
    document.write(
     "Date: " + dateString + "<br />",
     "Day: " + day + "<br />",
     "Month: " + month + "<br />",
     "Year: " + year + "<br />",
     "Time in milliseconds: " + timeMilliseconds +
"<br />",
     "Hour: " + hour + "<br />",
     "Minute: " + minute + "<br />",
     "Second: " + second + "<br />",
     "Millisecond: " + millisecond + "<br />");
// Time a loop
    var dum1 = 1.00149265, product = 1;
    var start = new Date();
    for (var count = 0; count < 10000; count++)
     product = product + 1.000002 * dum1 /
1.00001;
    var end = new Date();
    var diff = end.getTime() - start.getTime();
    document.write("<br />The loop took " + diff +
          " milliseconds <br />");
```

dowhile Stmt : Syntax :

do stmt or compound stmt
while (control expression).

* The do-while stmt is related to the while stmt, the test for completion is logically at the

end rather than the beginning of the loop construct.

* The body of a do-while construct is always executed atleast once.

Eg:

```
do {
    count++;
    Sum = Sum + (Sum * count);
} while count <= 50;
```

Basic Examples:

If-else:

```
<html>
 <body>
 <script>
 var a=5, b=3;
 if (a>b)
   document.write("a is greater");
 else
   document.write("b is greater");
 </script> </body> </html>.
```

For:

```
<html>
<body>
<script>
  var i;
  Var Sum = 0;
for (i=1; i<10; i++)
Sum = document.write
        (i);
</script>
</body>
</html>.
```

Switch:

```
<html> <body> <script>
 var a = 2, element;
 Switch(a)
 {
    case 0: element = "zero";
        document.write(element), break;
```

```
case 1 : element = "one";
    document.write(element); break;
case 2 : element = "two";
    document.write(element);
    break;
} </script> </body> </html>
```

While:

```
<html> <body>
<h1> Heading </h1>
<script>
var names = ["CSE", "ECE", "Mech"];
var i=0;
document.write("names in array list");
while (i<3)
{
document.write("<br>" + names[i] + "<br>");
i++;
} </script> </body> </html>.
```

do-while:

```
<html> <body>
<script>
var names = ["a", "b", "c"];
var i=0;
document.write("names in the list");
do
{ document.write("<br>" + names[i] + "<br>");
i++;
} while (i<3);
</script> </body> </html>
```

Arrays:

Arrays are objects that have special functionality and elements can be primitive values or references to other objects; including other arrays.

Array Object Creation:

Array objects, are created in 2 distinct ways.

① To create any object is with the new operator and a call to a constructor. The constructor is named Array:

ⓐ var my_list = new Array(1, 2, "three", "four");

ⓑ var your_list = new Array(100);

First declaration (a):

An Array object of length 4 is created and initialized. The elements of an array need not have the same type.

Second declaration (b):

A new Array object of length 100 is created, without creating any elements.

* when a call to the Array constructor has a single parameter, that parameter is taken to be the no. of elements, not the initial value of one element array.

② To create an Array object is with a literal array value, which the list enclosed in brackets.

```
var my_list_2 = [1, 2, "three", "four"];
```

Declaration of Array:

① `var a = [1, 2, 3, 4];`

② `var a = new Array(1, 2, 3, 4);`

③ `var a = [1, 2, "CSE", "IT"];`

④ `var a = new Array(1, 2, "CSE", "IT");`

⑤ `var a = new Array(4);` (should not follow).

Characteristics of Array Objects:

The lowest index of JavaScript Array is "zero"

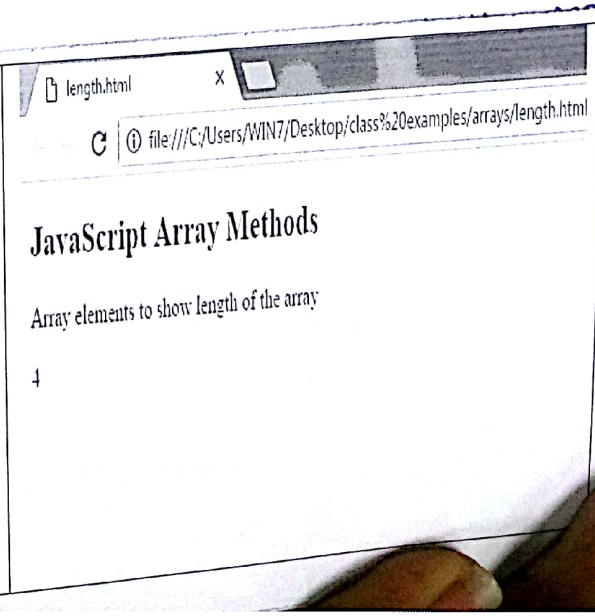
The length of an Array is highest subscript to which the value has been assigned is "+1".

Eg: `my_list[47] = 2222;`

Index = 47 newlength = 47 + 1 = 48.

Length Property:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array Methods</h2>
<p>Array elements to show length of the
array</p>
<p id="demo"> </p>
<script>
var prabas = ["varsham", "bahubali", "munna",
"yogi"];
prabas[0] = "bahubali2";
document.getElementById("demo").innerHTML =
prabas.length;
</script>
</body>
```



d - and
t is added
structor
ng

* length property can be lengthen, shorten or not affect the array's length.

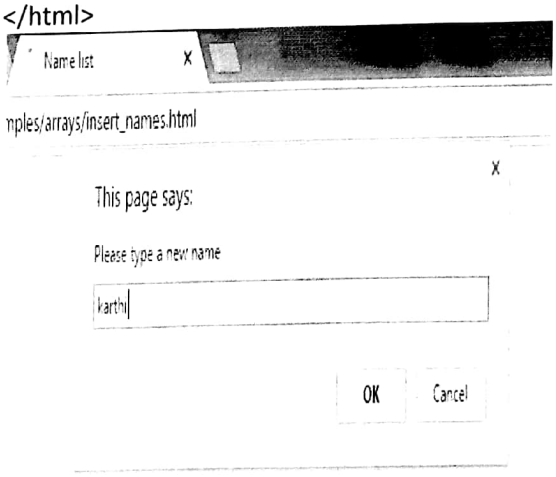
Eg: To use the subscript range of 100 to 150 (not 0 to 99).

array length created : 151

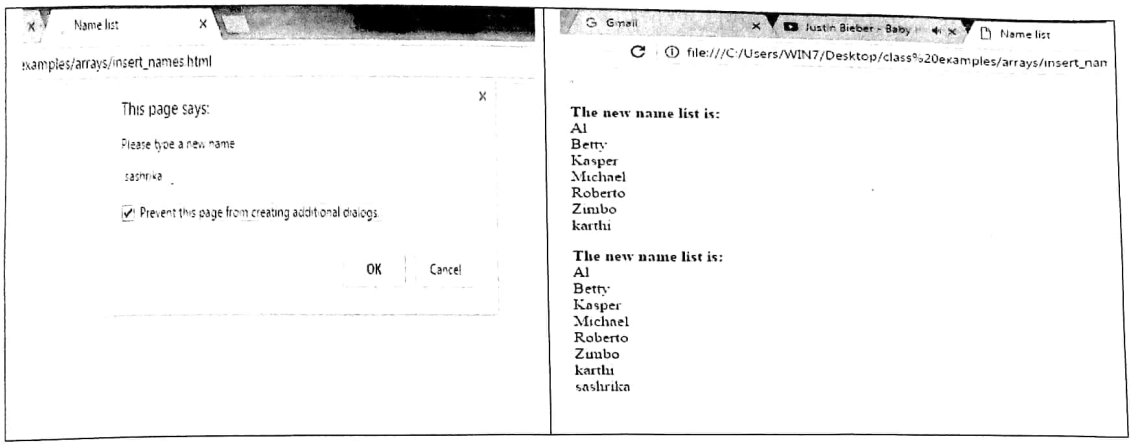
* But if only the elements indexed 100 to 150 are assigned values, the array requires only space 51 but not 151.

→ To support JavaScript's dynamic arrays, all array elements are allocated dynamically from the heap.

```
<!-- insert_names.html
    A document for insert_names.js
    -->
<html>
 <head> <title> Name list </title>
 </head>
 <body>
  <script type = "text/javascript"  src =
"insert_names.js" >
  </script>
 </body>
</html>
```




```
// insert_names.js
//  The script in this document has an array of
//  names, name_list, whose values are in
//  alphabetic order. New names are input
through
//  prompt. Each new name is inserted into the
//  name array, after which the new list is
//  displayed.
// The original list of names
    var name_list = new Array("Al", "Betty",
"Kasper",  "Michael", "Roberto", "Zimbo");
    var new_name, index, last;
// Loop to get a new name and insert it
    while (new_name =
          prompt("Please type a new name", "")) {
// Loop to find the place for the new name
     last = name_list.length - 1;
     while (last >= 0 && name_list[last] >
new_name) {
      name_list[last + 1] = name_list[last];
      last--;
     }
// Insert the new name into its spot in the array
     name_list[last + 1] = new_name;
// Display the new array
     document.write("<p><b>The new name list
is:</b> ","<br />");
     for (index = 0; index < name_list.length;
index++)
      document.write(name_list[index], "<br />");
     document.write("</p>");
   } //** end of the outer while loop
```



Array Methods :

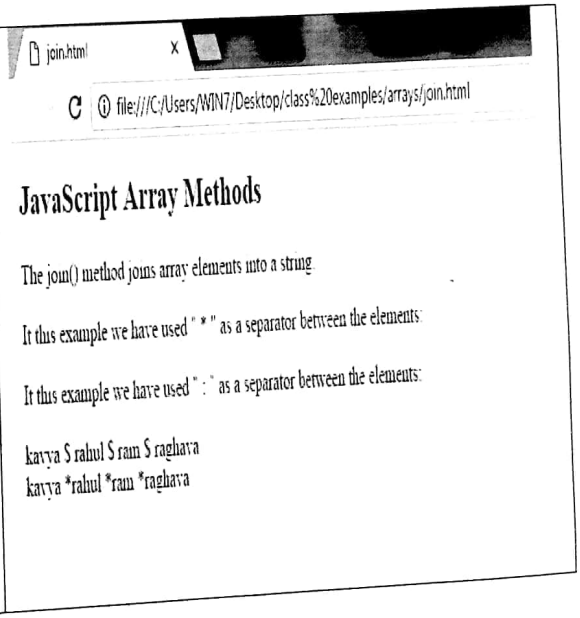
Array objects have a collection of methods,

Methods :

① Join Method : It converts all of the elements of an array to strings and concatenates them into a single string.

* If string parameter is provided, it is used as element seperator eg: ($, *, :)
* If no parameter is provided, the values in new string are separated by commas.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array Methods</h2>
<p>The join() method joins array elements into a
string.</p>
<p>It this example we have used " * " as a
separator between the elements:</p>
<p>It this example we have used " : " as a
separator between the elements:</p>
<script>
var list = ["kavya", "rahul", "ram", "raghava"];
document.write(list.join(" $ ")+"<br>"+list.join("
*"));
</script>
</body>
</html>
```



Syntax: var names = new Array ["mary", "murray", "murphy", "max"];

...

var name_string = names.join(" : ");

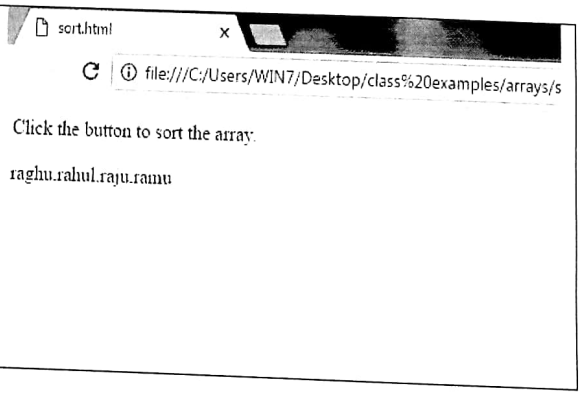
O/p: "Mary : Murray : Murphy : Max".

Reverse method: It reverses the order of the elements of the Array object through which it is called.

Sort Method: It coerces, the elements of the array to strings, if they are not already strings, and sorts them alphabetically.

Syntax: names.sort();

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to sort the array.</p>
<script>
var names = ["rahul", "raghu", "ramu", "raju"];
  names.sort();
  document.write(names);
</script>
</body>
</html>
```



Concat Method: It concatenates its actual parameter to the end of the Array object on which it is called.

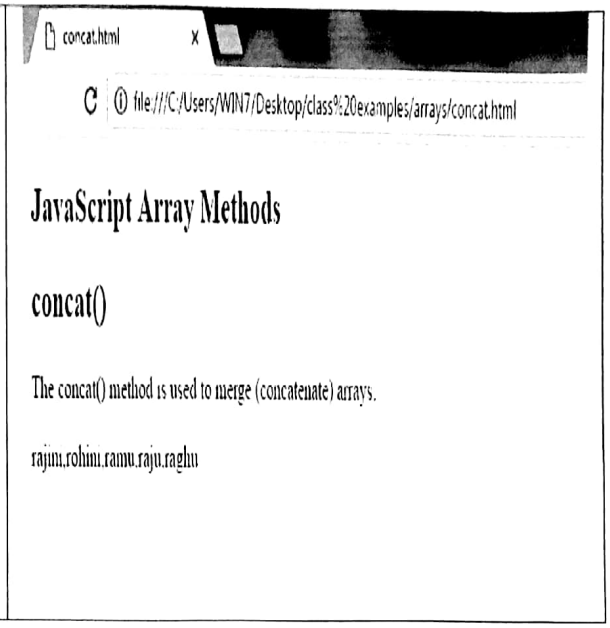
Eg: var names = new Array ["Mary", "Murray", "Murphy", "Max"];

...

var new_names = names.concat("Moo", "Meow");

Length of new_names is 6; with the elements of names, along "Moo" & "Meow".

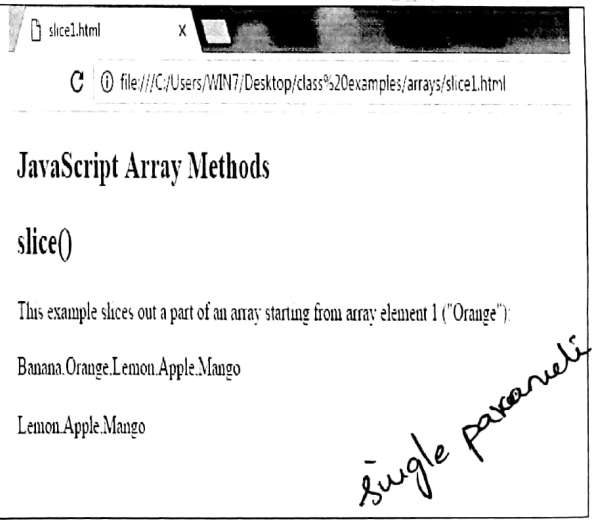
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>concat()</h2>

<p>The concat() method is used to merge
(concatenate) arrays:</p>
<script>
var Girls = ["rajini", "rohini"];
var Boys = ["ramu", "raju", "raghu"];
var Children = Girls.concat(Boys);
document.write(Children);
</script>
</body>
</html>
```



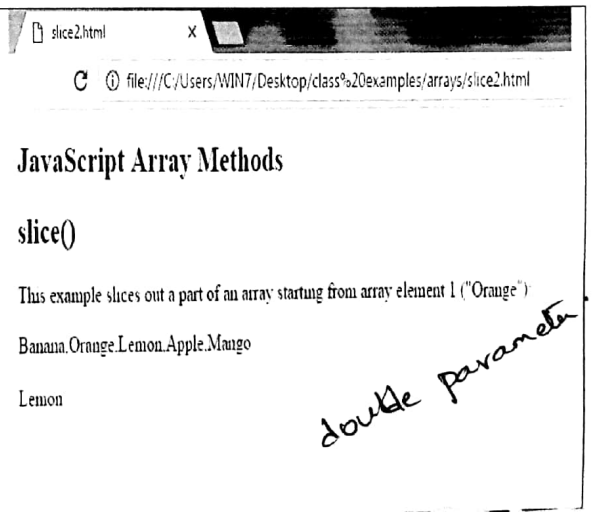
Slice Method: It does for arrays what the substring method does for string.

* It returns the part of the Array object specified by its parameters, which are used as subscripts.

```
<!DOCTYPE html>                                    ①
<html>
<body>
<h2>JavaScript Array Methods</h2>
<h2>slice()</h2>
<p>This example slices out a part of an array
starting from array element 1 ("Orange"):</p>
<script>
var fruits = ["Banana", "Orange", "Lemon",
"Apple", "Mango"];
var citrus = fruits.slice(2);
document.write(fruits + "<br><br>" + citrus);
</script>
</body>
</html>
```



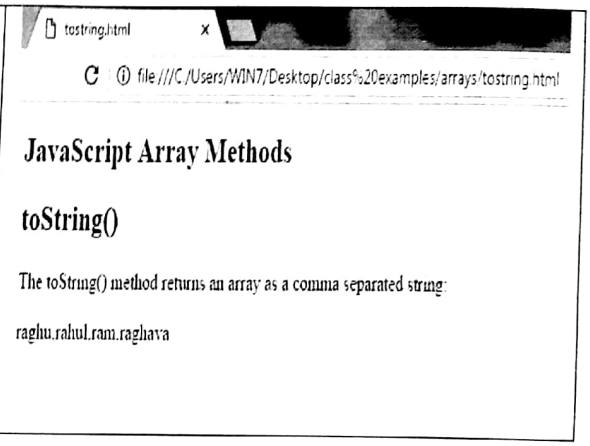

```
<!DOCTYPE html>                                    ②
<html>
<body>
<h2>JavaScript Array Methods</h2>
<h2>slice()</h2>
<p>This example slices out a part of an array
starting from array element 1 ("Orange"):</p>
<script>
var fruits = ["Banana", "Orange", "Lemon",
"Apple", "Mango"];
var citrus = fruits.slice(2,3);
document.write(fruits + "<br><br>" + citrus);
</script>
</body>
</html>
```



toString method: It is called through Array object, Each elements of the object is converted to string. and they are colenated and seperated by commas.

* the toString and join methods are behaves same.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array Methods</h2>
<h2>toString()</h2>
<p>The toString() method returns an array as a
comma separated string:</p>
<script>
var list = ["raghu", "rahul", "ram", "raghava"];
document.write(list.toString());
</script>
</body>
</html>
```



Push, Pop, Unshift and shift Methods: Array allow the easy implementation of stacks and queues in array.

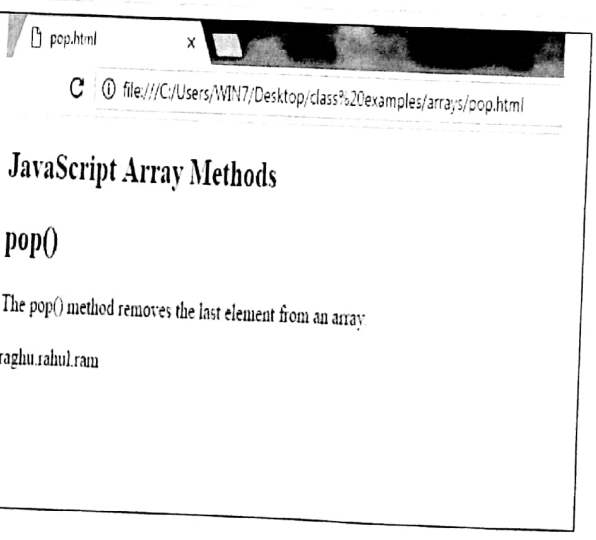
The pop and push methods: Remove and add an element to the high end of an array.

Eg: var list = { "Dasher", "Dancer", "Donner", "Blitzen"};

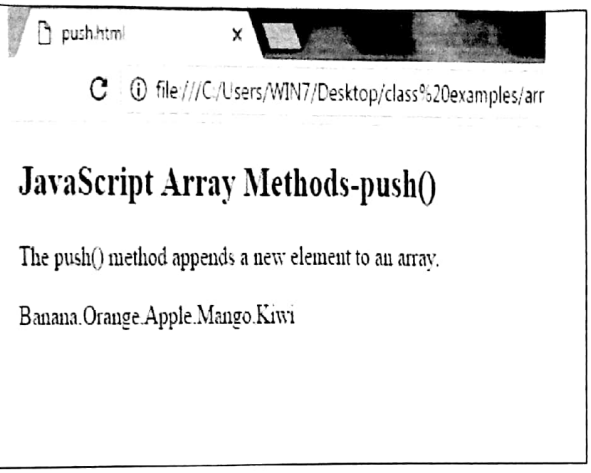
var deer = list.pop(); // deer is Blitzen"

list.push("Blitzen"); // This puts "Blitzen back on list.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array Methods</h2>
<h2>pop()</h2>
<p>The pop() method removes the last element
from an array.</p>
<script>
var list = ["raghu", "rahul", "ram", "raghava"];
list.pop();
document.write(list);
</script>
</body>
</html>
```



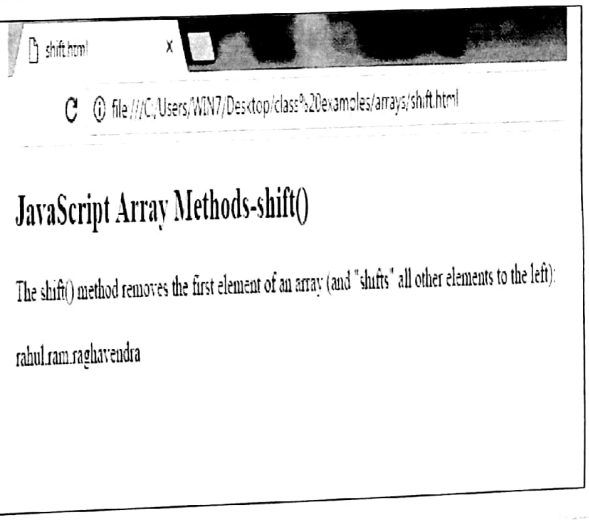
```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array Methods-push()</h2>
<p>The push() method appends a new element to an array.</p>
<script>
var list = ["Banana", "Orange", "Apple", "Mango"];
list.push("Kiwi");
document.write(list);
</script>
</body>
</html>
```



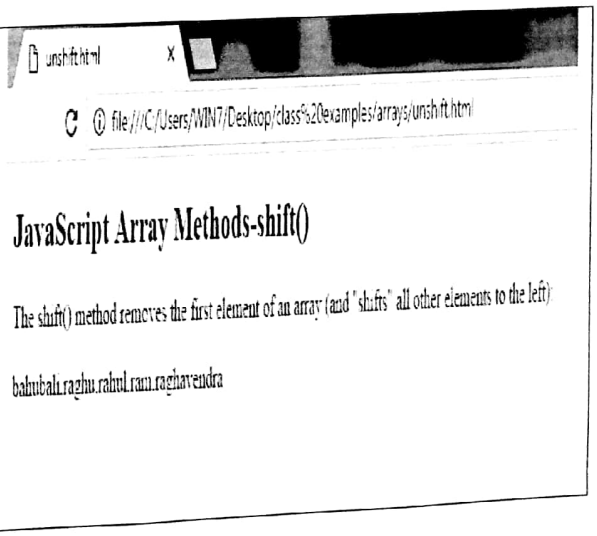
Shift and unshift methods : Remove and add an element to the beginning of an array.

Eg: var deer = list.shift(); // deer = now "Dasher"
list.unshift("Dasher"); // this puts "Dasher" back on list.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array Methods-shift()</h2>
<p>The shift() method removes the first element of an array (and "shifts" all other elements to the left):</p>
<script>
var list=["raghu","rahul","ram","raghavendra"];
list.shift();
document.write(list);
</script>
</body>
</html>
```

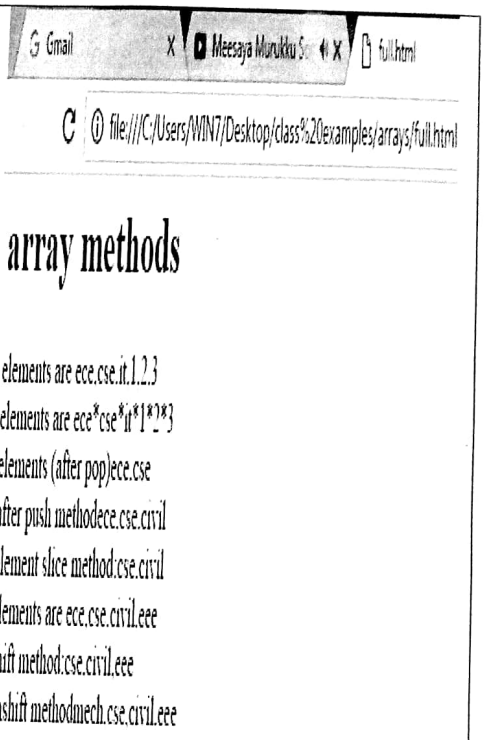



```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array Methods-shift()</h2>
<p>The shift() method removes the first element of an array (and "shifts" all other elements to the left):</p>
<script>
var list=["raghu","rahul","ram","raghavendra"];
list.unshift("bahubali");
document.write(list);
</script>
</body>
</html>
```



A Complete Example on Array Methods:

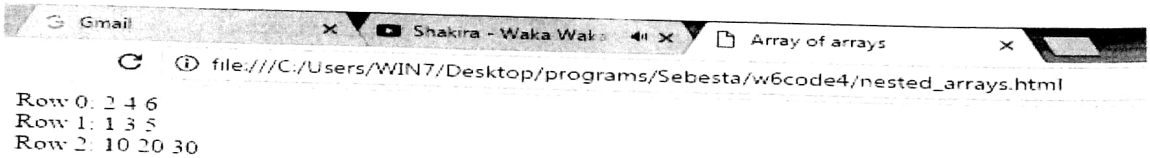
```
<html>
<body>
<h1> array methods</h1>
<script>
var branch=["ece", "cse","it"];
var num=[1,2,3];
var A=branch.concat(num);
document.write("elements are"+" " +A+"<br>");
var J=A.join("*");
document.write("elements are"+" " +J+"<br>");
branch.pop();
document.write("elements (after pop)"+branch+"<br>");
branch.push("civil");
document.write("after push method"+branch+"<br>");
var c=branch.slice(1);
document.write("element slice method:"+c+"<br>");
branch.push("eee");
document.write("elements are"+" " +branch+"<br>");
branch.shift();
document.write("shift method:"+branch+"<br>");
branch.unshift("mech");
document.write("unshift method"+branch+"<br>");
</script>
</body>
</html>
```



Text Book Example for Nested arrays: It is a 2 dimensional array implemented in JavaScript as an array of arrays.

```
<!-- nested_arrays.html
    A document for nested_arrays.js
    -->
<html >
 <head> <title> Array of arrays </title>
 </head>
 <body>
  <script type = "text/javascript"  src =
"nested_arrays.js" >
  </script>
 </body>
</html>
```

```
// nested_arrays.js
//  An example illustrate an array of arrays
// Create an array object with three arrays as its
element
    var nested_array = [[2, 4, 6],
                        [1, 3, 5],
                        [10, 20, 30]
                       ];
// Display the elements of nested_list
    for (var row = 0; row <= 2; row++) {
     document.write("Row ", row, ": ");
     for (var col = 0; col <=2; col++)
      document.write(nested_array[row][col], " ");
     document.write("<br />");
    }
```



Functions:

A function consists of the function header and compand statement that describes its actions.

* The compand stmt is called the body of the fn.
* A fn. header consists of the reserved word function, the fn's names or parameters list or if no parameter is there all are mentioned in a parenthesis.
* A return stmt returns control from the fn. in which it appear to the fn's caller.
* If no return stmt in a fn, the returned value is undefined.

Eg: fun1();
result = fun2();

→ if fun1 is a parameterless fn, that returns undefined.
→ if fun2 which also has no parameters, returns a useful value.

* JavaScript functions are objects, so variables that reference them can be treated as other object references.

Eg: function fun() { document.write("This Surely is fun!
"); }

ref_fun = fun; // now, ref_fun refers to the fun object
fun(); // A call to fun
ref_fun(); // Also a call to fun.

JS functions are objects, their addresses can be properties in other objects, they act as methods.

Local Variables:

The scope of a variable is the range of stmts over which it is visible.

* A variable that is not declared with a var stmt is implicitly declared by the JS interpreter at the time it is encountered in the script.
* Variables implicitly declared, it occurs within a fn definition, have ① global scope — they are visible in the entire HTML document.

② Global scope — Variables that are explicitly declared outside fn. definitions.

* Variables that are used only within a fn to have local scope, they are visible and can be used only within the body of the fn.
* Variable explicitly declared with var in the body of a fn. has local scope.

Advantage of Local variables:

If a variable ie., defined both local variable & a global variable appears in a fn., local variable has precedence, effectively hiding the global variable with the same name.

Parameters :

Actual Parameter : The parameter values that appear in a call to a fn.

Formal Parameter : The parameter names that appear in the header of a fn definition, which correspond to the actual parameter in calls to the fn.

Eg: An array is passed as a parameter to a fn.

```
function fun1 (my_list)        <-- calling code
{
    var list2 = new Array(1,3,5);
    my_list[3] = 14;           <-- to refer to a diff array object
    ...
    my_list = list2;
    ...
}
...
var list = new Array (2,4,6,8)
fun1(list);
```

The Sort Method, Revisited:

Sort method for array objects converts the array elements to strings, alphabetically.

* To sort other than strings, the comparision operation must be supplied to the sort method by the caller.
* It is passed as a parameter sort.
* The comparision fn must return ① a negative number if two elements compared is the desired order, ② zero if they are equal, ③ a number greater than zero if they must be interchanged.

For numbers: subtract the second from the first produces the required result.

Eg: to sort the array of numbers num_list into descending order using the sort method,.

```
function num_order(a, b)
{
  return b-a;
}
num_list.sort(num_order);
```

Errors in Scripts:

The JS interpreter is capable of detecting various errors in scripts. Like Syntax errors, uses of undefined variables are also detected.

* Errors are detected by the JS interpreter, while the browser attempts to display a document.
* Script errors cause the browser to not display the document and do not produce an error message.

To find the problem:

The default settings for IE7 provide no debugging help for JS.

Changes to be followed:

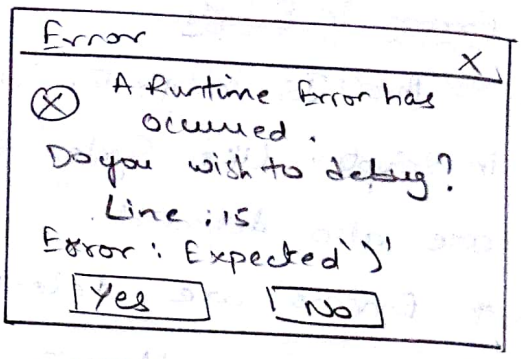
* Select Internet Options from the Tools menu.
* Choose the Advanced tab. It opens a window with a long list of checkboxes.
* Uncheck the disable script debugging box and Check the Display a notification about every script error box.
* Press the Apply button in window.

→ JS error will cause the browser to open and display a small window with an explanation of problem.

Eg: // debugdemo.js

```
var row;
row=0;
while (row!=4
{
  document.write("row is",
        row, "<br>");
  row++;
}
```

o/p:



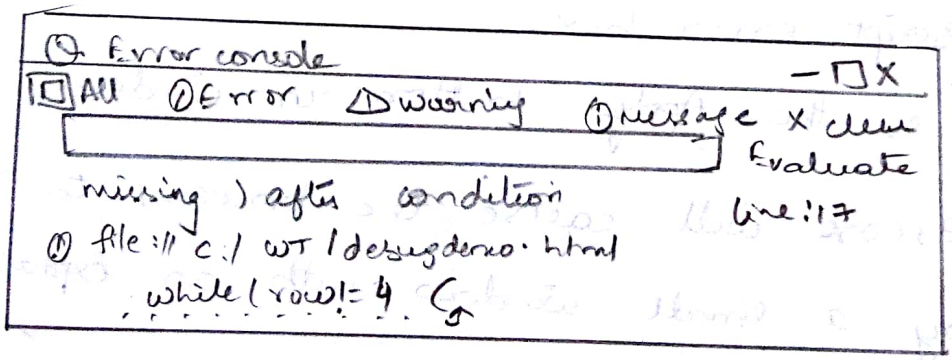
→ The Syntax error in the while Stmt.

→ The FX2 browser has a special console window that displays script errors.

* Select Tools & Error console to open window (when using this browser to display documents that include JS, this window should be kept open).

* After error message has appeared and fixed, press the Clear button on console.

o/p:



* The problems are detectable only during execution or interpretation.