INTRODUCTION TO JAVASCRIPT

Mr Rajiv. S. Bal

M.Tech(CSE)

Outlines

To achieve

- Introduction
- Variables & console
- Operators
- Control flow
- Looping structure
- Functions
- Events
- Objects

Introduction

- There are three types of script languages :

	Developed By
JAVAScript	- Netscape
JScript	- Microsoft
VBScipt	- Microsoft

JAVASCRIPT

- In client-side, JavaScript is traditionally embedded into a standard HTML program.
- It is embedded between <script>…… </script> HTML tags.
- These tags can be embedded within the Head document i.e. <head> … </head> or body document i.e. <body> … </body> tags of the HTML program.
- Syntax of JavaScript :

```
<html>
<head>

<script
language="JavaScript">
… … … … … …
//Java Script codes. … … ... …
… … … … … …
</script>

<title> … text… </title>
</head>
<body>

<script
language="JavaScript">
… … … … … …
//Java Script codes. … … ... …
… … … … … …
</script>

</body>
</html>
```

Javascript Data Types

- There are two types of data types in JavaScript.
  - Primitive data type
  - Non-primitive (reference) data type.
- We need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc.
- For example:
  - var a=40;//holding number
  - var b="Rajiv";//holding string
- **Primitive data types**
  - There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

Javascript Data Types (conti.)

- **Non-primitive data types**
  - The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

- **Comment Statements**
  - // (double slash) i.e. single comment line.
  - /* (Multi-comment statement)

```
....
statements
....
*/
```

- **Output Formatting Statement**
  - document.write(" Message ");

OR

  - document.writeln(" Message ");

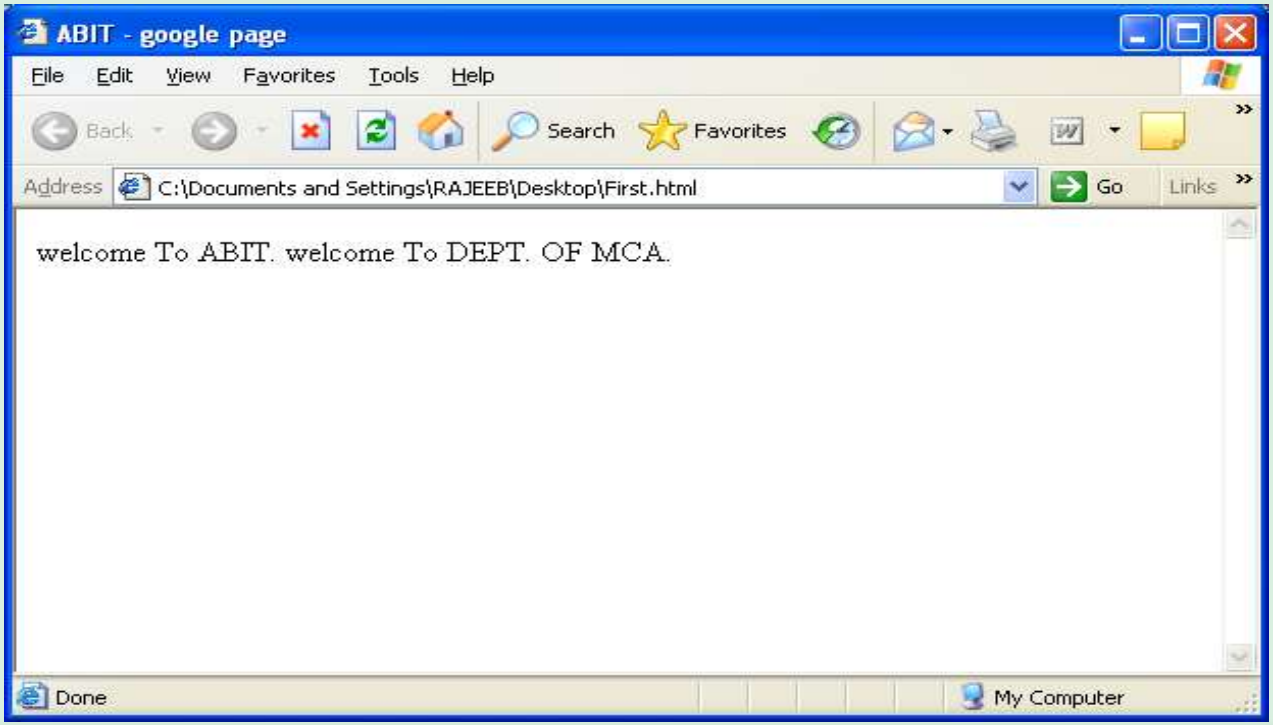
Example of “document.write”statement

First - Notepad

File Edit Format View Help

```
<html>
<head>
<script language="JavaScript">
document.writeln("welcome To ABIT.");
</script>
<title>ABIT</title>
</head>
<body>
<script language="JavaScript">
document.write("welcome To DEPT. OF MCA.");
</script>
</body>
</html>
```

Output



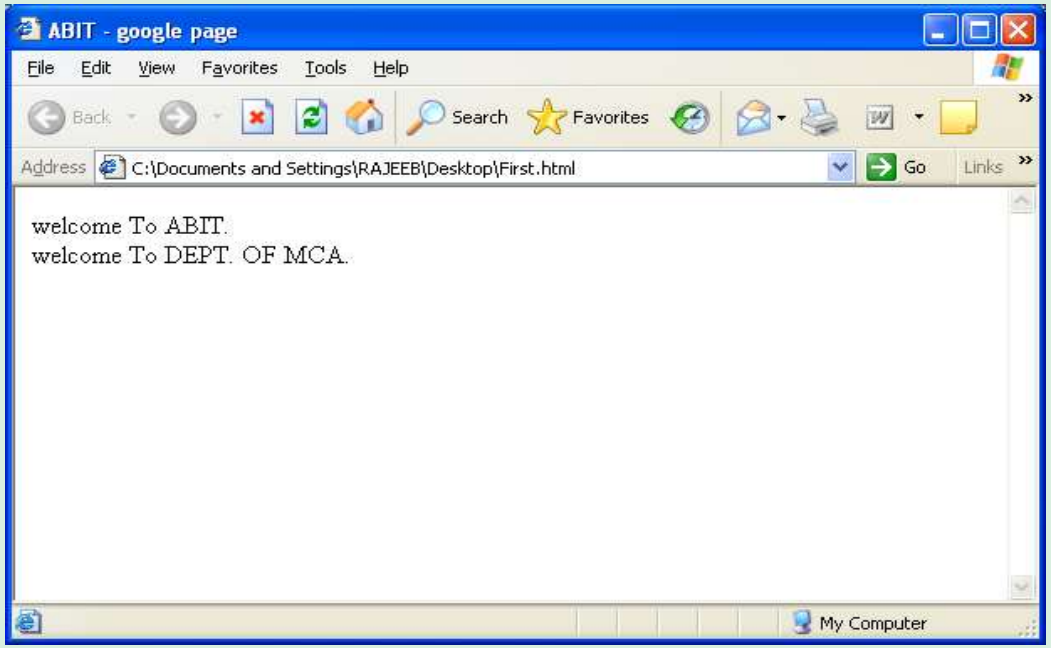
Using
 tag in “document.writeln ()”

First - Notepad

File Edit Format View Help

```
<html>
<head>
<script language="JavaScript">
document.writeln("welcome To ABIT. <br>");
</script>
<title>ABIT</title>
</head>
<body>
<script language="JavaScript">
document.write("welcome To DEPT. OF MCA.");
</script>
</body>
</html>
```

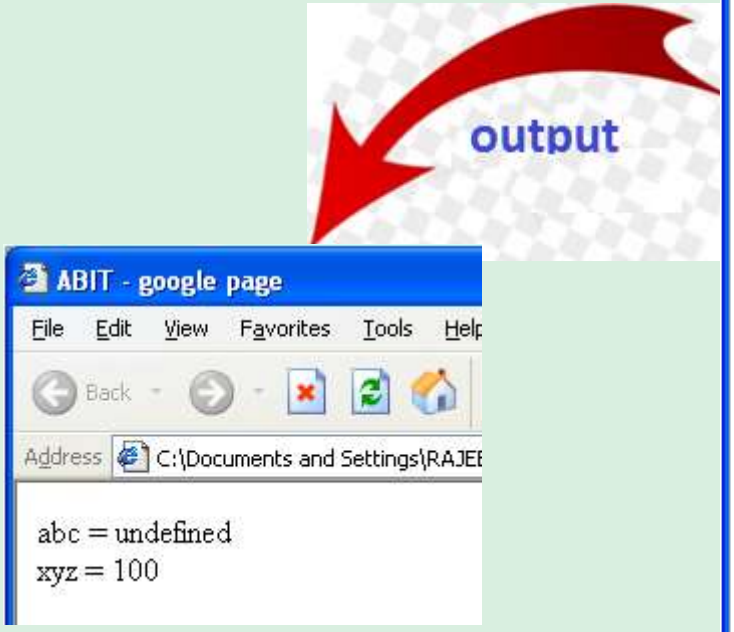
Output



More About document.write statement

- In JavaScript, the document.write statement is used displaying some text on the screen.
- The same statements can also used to display the contents of a variable.
- Syntax for document.write() statement :
  - document.write(“message” + variable);

Example of Variable In JavaScript :



First - Notepad

File Edit Format View Help

```
<html>

        <head>

<script language="JavaScript">
        var abc ;
document.write(" abc = " + abc + "<br>");
</script>

        <title>ABIT</title>

        </head>

        <body>

<script language="JavaScript">
        var xyz = 100;
document.write(" xyz = " + xyz);
</script>

        </body>

</html>
```

Escape sequence and Command block

- **Escape Sequence**
  - Escape sequence are special characters with pre-defined meaning.

    \n :- To insert new line.

    \t :- To insert new line.

    \r :- To insert a carriage return.
- **Command Block**
  - Multiple commands can be combined into command block using curly braces ({ and }).
- **The <pre> Tag**
  - The entire text with special formatting characters was enclosed within <pre> and </pre> tags.
  - This to prevent the formatting characters from being treated as HTML white space characters.

Syntax of <pre> tag

```
<html><head>
        <pre>
<script language="JavaScript">
{
// Multi-statement of JavaScript.
}
</script>
        </pre>
<title> … Text… </title>
</head><body>
        <pre>
<script language="JavaScript">
{
// Multi-statement of JavaScript.
}
</script>
        </pre>
</body></html>
```

The Example of <pre> tag

Test - Notepad

File Edit Format View Help

```
<html><head>
<title>JavaScript</title>
                    <pre>
<script language="JavaScript">
{
document.writeln("Welcome To JavaScript");
}
</script>
                     </pre>

</head>
<body>
                    <pre>
<script language="JavaScript">
{
document.writeln("Hi Hello JavaScript");
}
</script>
                    </pre>
</body></html>
```

output

File Edit View History Bookmarks T

Home

file://

Most Visited Getting Started

```
Welcome To JavaScript


Hi Hello JavaScript
```

Operators in Java Script

- JavaScript operators are symbols that are used to perform operations on operands
  - Arithmetic Operators
  - Comparison (Relational) Operators
  - Bitwise Operators
  - Logical Operators
  - Assignment Operators
  - Special Operators
- **Arithmetic Operators**

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

The '+ or addition or plus'operator

Test - Notepad

File Edit Format View Help

```
<html><head>
<title> + operator </title>
<script language="JavaScript">
{
var a , b ,c;
a = 11;
b = 55;
        c = a+b;
document.writeln("a + b =" + c);
}
</script>
</head>
<body><pre>
<script language="JavaScript">
{
var a , b ,c;
a = "11";
b = "55";
        c = a+b;
document.writeln("a + b =" + c);
}
</script></pre>
</body>
</html>
```

output

File Edit View History Bookmarks Tools Help

Home | + operator

file:///C:/Users/JYOSHNA

Most Visited Getting Started

a + b =66

a + b =1155

Operators in Java Script (conti.)

Relational or Comparison Operators

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

Bitwise-Shift Operators

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

Operators in Java Script (conti.)

Assignment or Shorthand or Abbrivation Operators

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

Special Operators

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

Example of **typeof Operator**

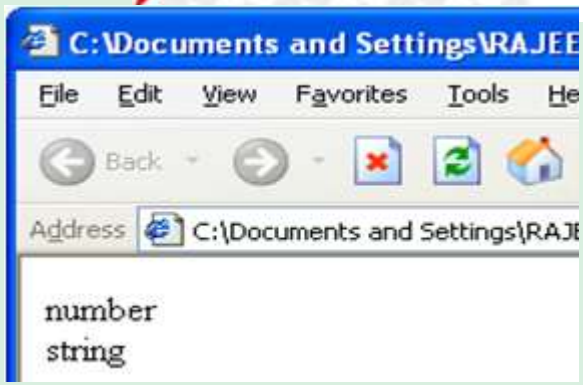
- The typeof operator returns a string indicating the type of operand.

First - Notepad

File Edit Format View Help

```
<html>
        <head>
<script language="JavaScript">
        var a,b;
        a = typeof(12);
        b = typeof("HELLO");
document.write(a+"<br>");
document.write(b);
</script>
        </head>
        <body></body>
</html>
```

output



Conversion Functions

- eval() :- This function converts a string to a numeric value.

  exp : s = eval(“1234”);

- parseInt() :- It extracts the first integer from any given string that begins with an integer.

  exp : s = parseInt(“123Hello”); returns 123;

- parseFloat() :- It extracts the float from any given string that begins with an integer.

  exp : s = parseInt(“1.23”); returns 1.23;

JavaScript alert - prompt - confirm Dialog Boxes

alert() is a simple function to display a message to a dialog box (also called alert box). Here is a simple example to display a text in the alert box.

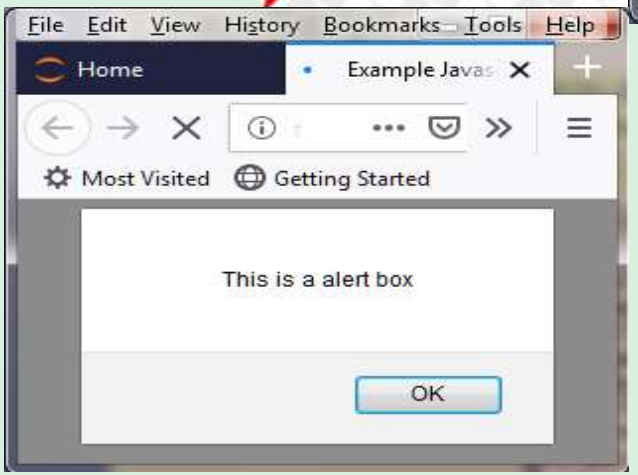
Syntax : alert(message);

Test - Notepad

File Edit Format View Help

```
<html>
<head>
<title>Example Javascript alert box</title>
</head>
<body>
<script language="JavaScript">
alert("This is a alert box");
</script>
</body>
</html>
```

output



JavaScript alert - prompt - confirm Dialog Boxes (conti.)

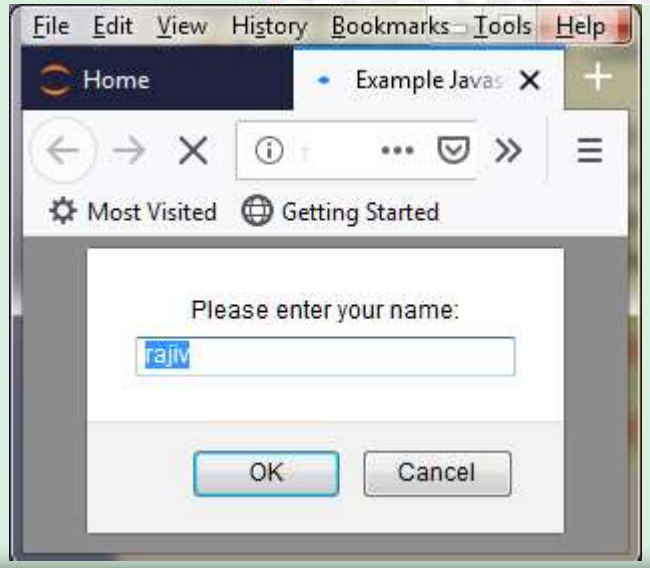
Prompt Box

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

Syntax

```
window.prompt("sometext","defaultText");
or
prompt("sometext","defaultText");
```

output



Test - Notepad

File Edit Format View Help

```
<html>
<head>
<title>Example Javascript alert box</title>
</head>
<body>
<script language="JavaScript">
{
        var txt;
  var person = prompt("Please enter your name:", "rajiv");
  if (person == null || person == "") {
    txt = "User cancelled the prompt.";
  } else {
    txt = "Hello " + person + "! How are you today?";
  }
  alert(txt);
}
</script>
</body>
</html>
```

JavaScript alert - prompt - confirm Dialog Boxes (conti.)

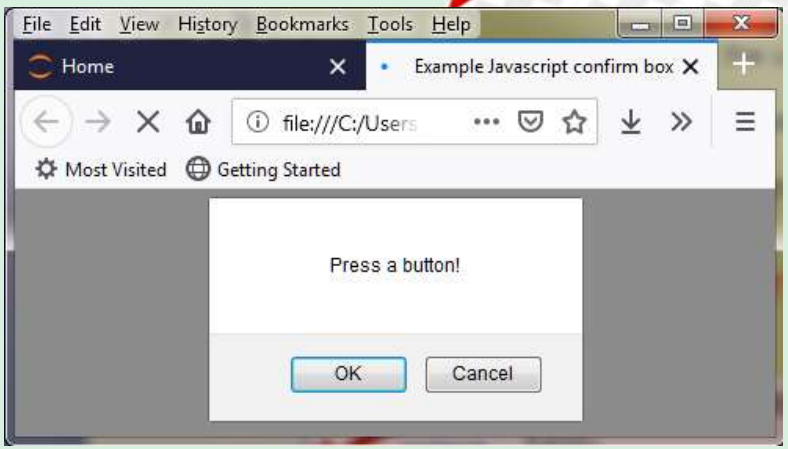
Confirm Box

- A confirm box is often used if you want the user to verify or 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.

Syntax

```
window.confirm("sometext");
or
confirm("sometext");
```

output



Test - Notepad

File Edit Format View Help

```
<html>
<head>
<title>Example Javascript confirm box</title>
</head>
<body>
<script language="JavaScript">
{
        var txt;
  if (confirm("Press a button!")) {
    txt = "You pressed OK!";
  } else {
    txt = "You pressed Cancel!";
  }
  alert(txt);
}
</script>
</body>
</html>
```

JavaScript if-else

- The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.
  - if Statement
  - if else Statement
  - if else if Statement
- **JavaScript If statement**
  - It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

```
if(expression){
    //content to be evaluated
}
```

- **Example**

```
<script>
var a=20;
if(a>10){
document.write("value of a is greater than 10");
}
</script>
```


JavaScript if-else (conti.)

- JavaScript if...else Statement
  - It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

```
if(expression){
  //content to be evaluated if condition is true
  }
  else{
  //content to be evaluated if condition is false
  }
```

- Example

```
<script>
var a=20;
if(a%2==0){
document.write("a is even number");
}
else{
document.write("a is odd number");
}
</script>
```

JavaScript if-else (conti.)

JavaScript if...else if statement

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

```
if(expression1){
//content to be evaluated if6******* expression1 is true
}
else if(expression2){
//content to be evaluated if expression2 is true
}
else if(expression3){
//content to be evaluated if expression3 is true
}
else{
//content to be evaluated if no expression is true
}
```

Example

```
<script>
var a=20;
if(a==10){
document.write("a is equal to 10");
}
else if(a==15){
document.write("a is equal to 15");
}
else if(a==20){
document.write("a is equal to 20");
}
else{
document.write("a is not equal to 10, 15 or 20");
}
</script>
```


JavaScript if-else (conti.)

- **JavaScript switch...case**
  - The **JavaScript switch statement** is used *to execute one code from multiple expressions*. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.
  - The signature of JavaScript switch statement is given below.

```
switch(expression){
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
.....

default:
 code to be executed if above values are not matched;
}
```

JavaScript if-else (conti.)

- Example of switch…case

```
<script>
var grade='B';
var result;
switch(grade){
case 'A':
result="A Grade";
break;
case 'B':
result="B Grade";
break;
case 'C':
result="C Grade";
break;
default:
result="No Grade";
}
document.write(result);
</script>
```


JavaScript Loops

- The JavaScript loops are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.
- There are four types of loops in JavaScript.
  - for loop
  - while loop
  - do-while loop
  - for-in loop
- **JavaScript for loop**
- The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment)
{
    code to be executed
}
```


Example of for loop in javascript

```
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
```

- JavaScript while loop
- The JavaScript while loop iterates the elements for the infinite number of times. It should be used if number of iteration is not known. The syntax of while loop is given below.

```
while (condition)  {
    code to be executed
}
```

- Exmple

```
<script>
var i=11;
while (i<=15)  {
document.write(i + "<br/>");
i++;
}
</script>
```


JavaScript do while loop

- The JavaScript do while loop iterates the elements for the infinite number of times like while loop. But, code is executed at least once whether condition is true or false. The syntax of do while loop is given below.

```
do{
    code to be executed
}while (condition);
```

- Example

```
<script>
var i=21;
do{
document.write(i + "<br/>");
i++;
}while (i<=25);
</script>
```

Using break and continue keyword in loop statements
The endless loop

- We can create endless loop, which runs forever, using a for Statement, using a while statement or a do/while statement.
- // Here is a endless for loop that runs forever.

```
for (;;) {
    statement;
}
```

- // Here is a endless while loop that runs forever.

```
while (true) {
    statement;
}
```

- // Here is a endless do-while loop that runs forever.

```
do {
    statement;
} while (true);
```

- In these loops you have to use at least one break keyword to end the loop.

Using break and continue keyword in loop statements
The endless loop (conti.)

- You can also use the **continue keyword** in any loop. A continue keyword cause a jump to an new expression evaluation .
- The **break keyword** can be used in any loop and cause a jump out of the loop.
- Example of a **endless for loop** with break keyword and continue keyword:

```
<script>
  var count=0, max=5;
 for (;;) {          // A for loop that doesn't end
  if (count < max) {  // test
    count++;         // increment
    if (count==2) {
      continue;
    }
    // All counts are listed except number 2
    document.write("Counter : " + count + "<br>");
  }  else
    // You must have at least one break
    //  in a endless loop; like this
    break;
 }
</script>
```


Using labels

- Any loop can exist in any loop that can exist in any loop and so on. In those cases, the break keyword and continue keyword only works in the loop it exists.
- JavaScript allows the break keyword and the continue keyword to be followed by the name of a label.
  - break labelName;
  - continue labelName;
- Any statement may be labeled by preceding it with an identifier name and a colon:
  - identifier: statement;

Example using break keyword and continue keyword with use of labels:

```
<script>
  var x = 0;
  outerloop:
    for(;;) {
    x++;
    innerloop:
    for(var y = 0; y < 10; y++) {
      if (x == 9 ) break outerloop;  // quit outer loop
      if (y > 3) break;              // Quit the innermost loop
      if (x == 2) break innerloop;  // Do the same thing
      if (x == 4) continue outerloop;  // new outer loop test
      if ((x >= 7) && (x < 9)) continue;  // new inner loop test
      document.write("x = " + x + " y = " + y + "<br>");
    }
  }
  document.write("At end x = " + x + " y = " + y + "<br>");
</script>
```

Nested for Loop in JavaScript

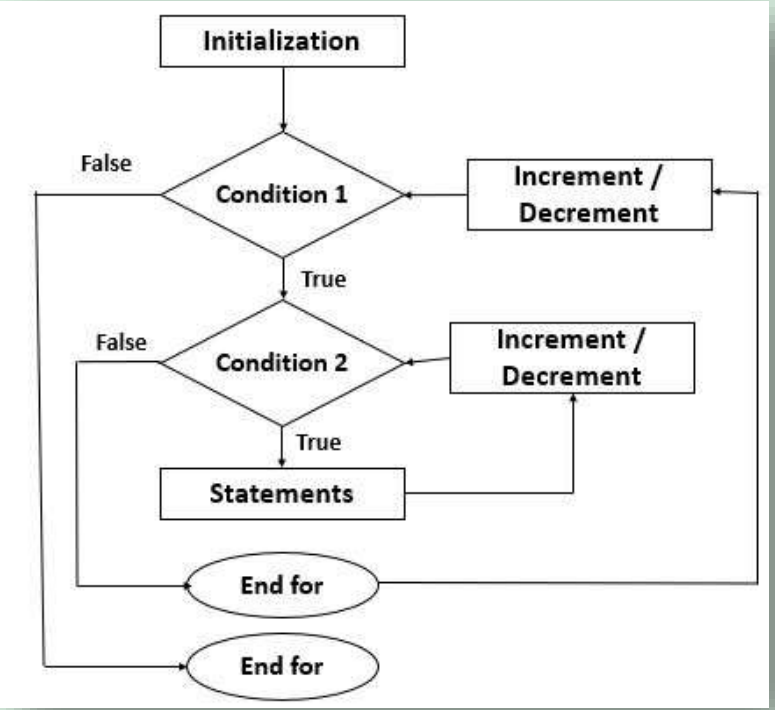
Nested Loop is a loop that is present inside another loop. Javascript supports the nested loop in javascript. The loop can have one or more or simple can have any number of loops defined inside another loop, and also can behave n level of nesting inside the loop. The nested loop is also called as inner loop and the loop in which the nested loop defined is an outer loop. The outer loop always executes first and the inner loop executes, the inner loop executes each time the outer loop executes once. In the case of multi-level nested an outer loop executes first and then the first inner loop executes and then second inner loop executes and so on. Any type of nested loop can be defined inside any type of loops.

The Syntax for Nested loop in javascript :

```
Outerloop
{
Innerloop
{
// statements to be execute inside inner loop
}
// statements to be execute inside outer loop
}
```


Nested for Loop in JavaScript (conti.)

Flowchart for Nested for Loop :



The nested for loop means any type of loop that is defined inside the for loop:
Syntax:

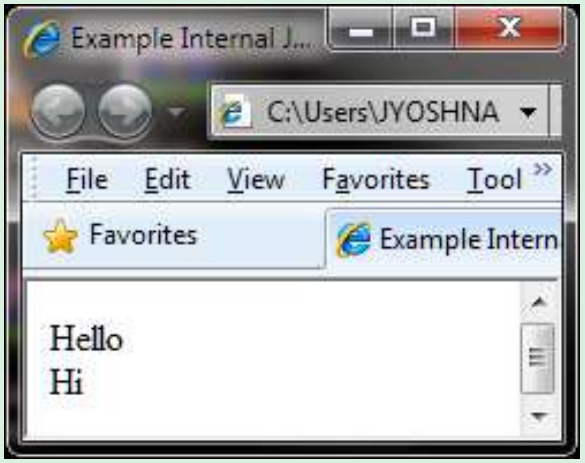
```
for (initialization; cond; increment/decrement)
{
for(initialization; cond; increment/decrement)
{
// statements to be execute inside inner loop.
}
// statements to be execute inside outer loop
}
```

Internal & External JavaScript File

We can use JavaScript code in two ways:

1. We can either include the JavaScript code internally within the HTML document itself. Example of JavaScript program save as **Filename.html**

```
<html>
<head>
<script  type="text/javascript">
document.write("Hello <br>");
</script>
<title>Example Internal JavaScript</title>
</head>
<body>
<script  type="text/javascript">
document.write("Hi");
</script>
</body>
</html>
```



Save : Test.html

2. We can keep the JavaScript code in a separate external file and then point to that file from your HTML document.

- JavaScript provides three places to put the JavaScript code: within body tag, within head tag and external JavaScript file.
- The tree Places to put JavaScript code :
  - Between the body tag of html
  - Between the head tag of html
  - In .js file (external javaScript)

Syntax :

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

Example external javaScript

```
var a=20;
if(a>10){
document.write("value of a is greater than 10");
}
```

Save : Head.js

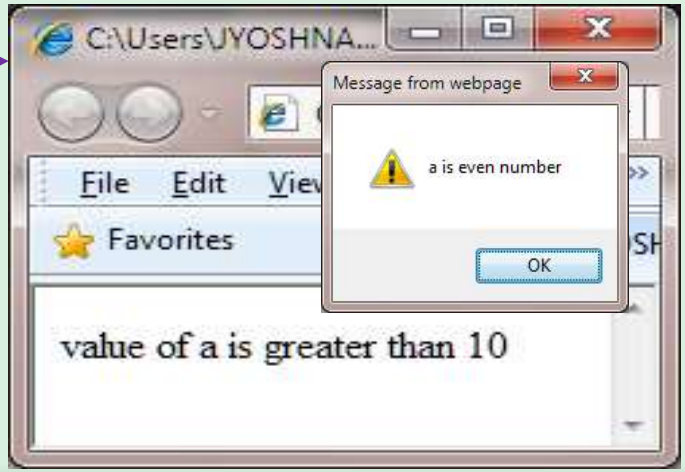
```
var a=20;
if(a%2==0){
alert("a is even number");
}
else{
alert("a is odd number");
}
```

Save : Body.js

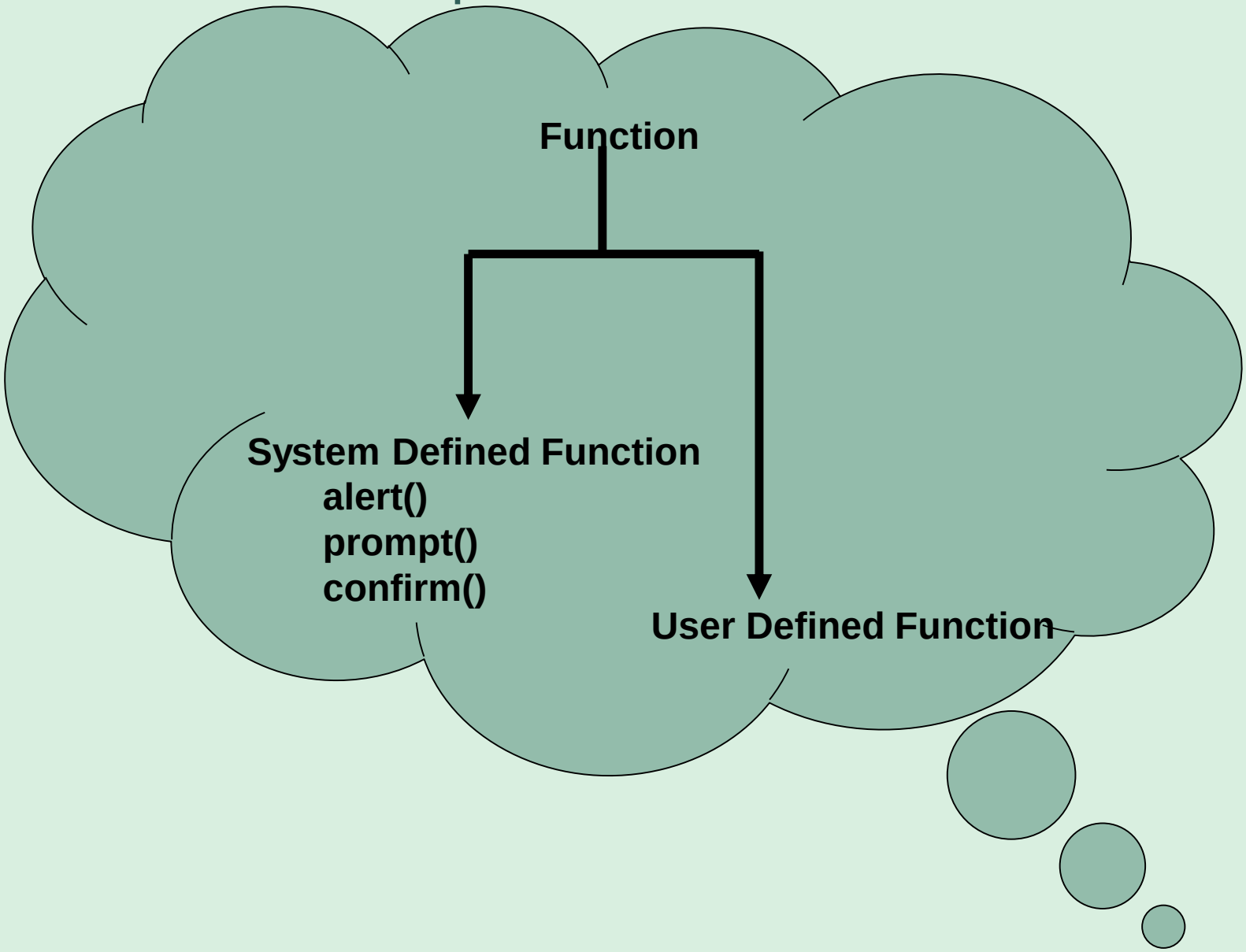
```
<html>
<head>
<script type="text/javascript" src="Head.js"></script>
</head>
<body>
<script type="text/javascript" src="Body.js"></script>
</body>
</html>
```

Save : Example.html
Run by : Browser

OUTPUT



Function in JavaScript



User-Defined Function

- Function is a program module, which is complete in its own.
- Functions may or may not return a value.
- Every function has a name and an argument list which is optional.
- In JavaScript function are declared in following manner.

```
function function_name(arg1 … argN)
        {
        statement (s);
        }
```

Function in JavaScript (conti.)

- **The general form of function is followed :**

```
<html> <head>
<script language="JavaScript">
{
function function_name(arg1 … argN)
    {
        statement (s);
    }
}
</script>
 <title> … Text… </script>
</head>
<body>
<script language="JavaScript">
{
 function_name(arg1 … argN);
}
</script>
</body>
</html>
```

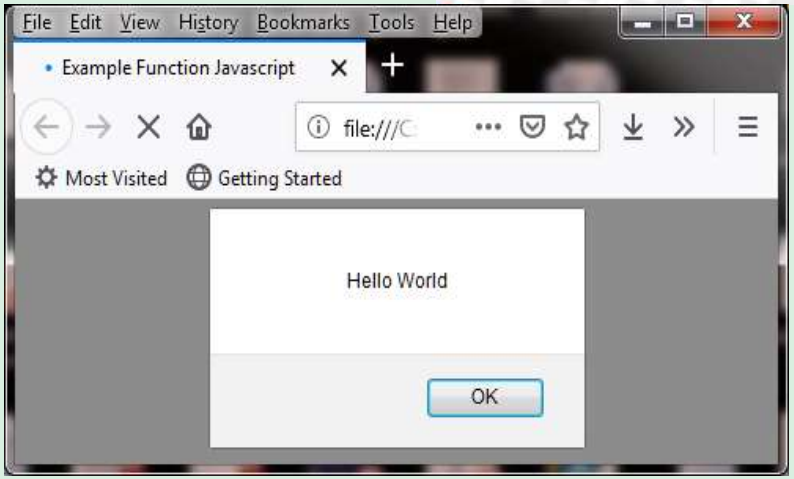
Example of function in JavaScript

Test - Notepad

File Edit Format View Help

```
<html>
<head>
<script language="JavaScript">
{
function show(){
var txt = "Hello World";
alert(txt);
}
}
</script>
<title>Example Function Javascript</title>
</head>
<body>
<script language="JavaScript">
{
show();
}
</script>
</body>
</html>
```

output



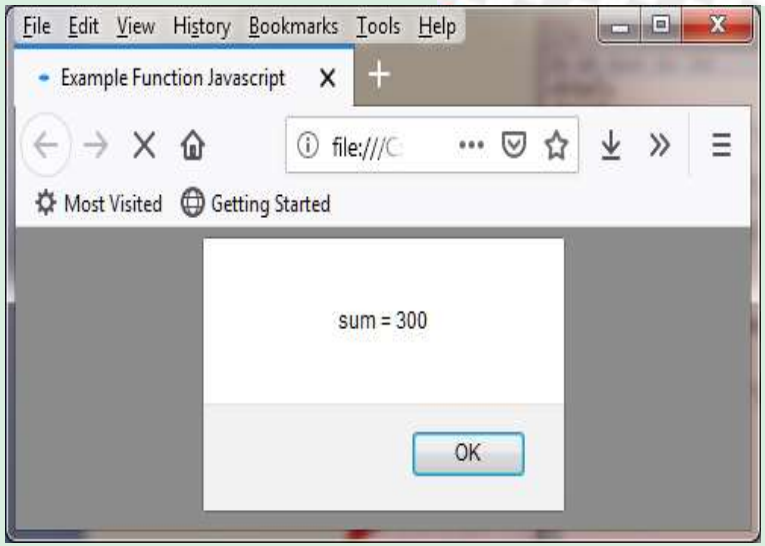
Example of function in JavaScript

Test - Notepad

File Edit Format View Help

```
<html>
<head>
<script language="JavaScript">
{
function show(x,y){
var txt = x+y;
alert("sum = "+txt);
}
}
</script>
<title>Example Function Javascript</title>
</head>
<body>
<script language="JavaScript">
{
var a , b;
a=100;
b=200;
show(a,b);
}
```

output



Another form of function in JavaScript

- **Syntax**

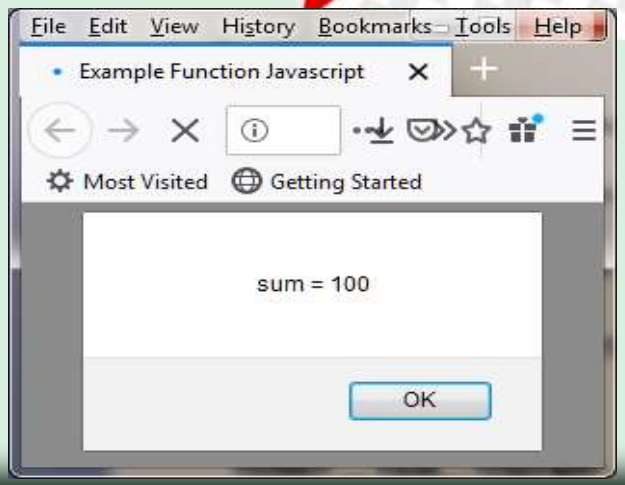
```
<html> <head>
<script language="JavaScript">
{
function function_name(arg1 ... argN)
    {
                statement (s);
    }
 function_name(arg1 ... argN);

}
</script>
 <title> ... Text... </script>
</head>
<body>
</script>
</body>
</html>
```

Test - Notepad

File Edit Format View Help

```
<html>
<head>
<script language="JavaScript">
{
function show(x,y){
var txt = x-y;
alert("sum = "+txt);
}
var a , b;
a=200;
b=100;
show(a,b);
}
</script>
<title>Example Function Javascript</title>
</head>
<body>
</body>
</html>
```

output



The return statement

- This statement is used to return a value from a function. The value is returned to a the statement, which invoked the function.

  Syntax of the return statement :

  return (expression);

- Complete form the function with return statement :

```
<html> <head>
<script language="JavaScript">
{
function function_name(arg1 … argN)
    {
        statement (s);
        return (someThing);
    }
}
</script>
 <title> … Text… </script></head>
<body>
<script language="JavaScript">
{
 function_name(arg1 … argN);
}
</script></body></html>
```

The return statement (conti.)

- Another form the function with return statement

```
<html> <head>
<script language="JavaScript">
{
function function_name(arg1 … argN)
    {
        statement (s);
        return (someThing);
    }
}
</script>
 <title> … Text… </script>
</head>
<body>
<script language="JavaScript">
{
 function_name(arg1 … argN);
}
</script>
</body>
</html>
```

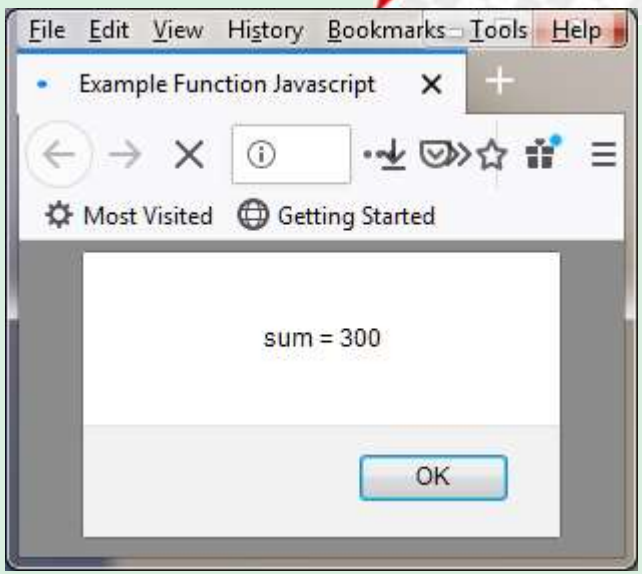
Example of function with return statement

Test - Notepad

File Edit Format View Help

```
<html>
<head>
<script language="JavaScript">
{
function show(x,y){
var txt = x+y;
return txt;
}
var a , b , c;
a=200;
b=100;
c = show(a,b);
alert("sum = "+c);
}
</script>
<title>Example Function Javascript</title>
</head>
<body>
</body>
</html>
```

output



Local And Global Variables

- JavaScript supports two types of variables :
  1. Local Variables.
  2. Global Variables.

Syntax:

```
<html> <head>
<script language="JavaScript">
{
    global variable declaration.
function function_name(arg1 ... argN)
    {
     local variable declaration.
          statement (s);
          return (someThing);
    }
 function_name(arg1 ... argN);

}
</script>
 <title> ... Text... </script>
</head>
<body>
</script>
</body>
</html>
```

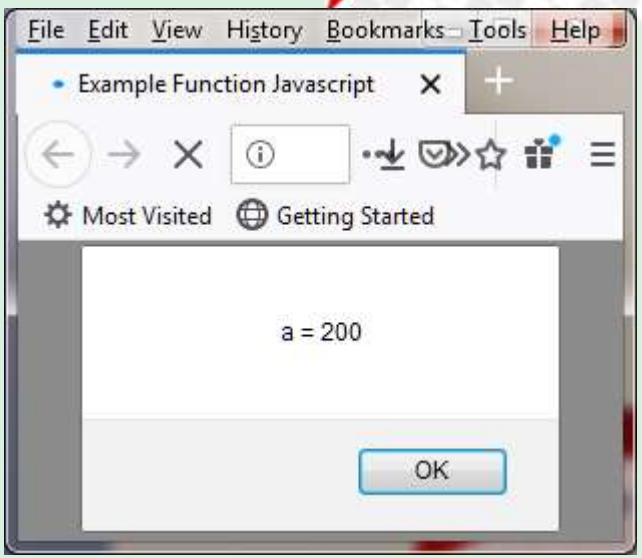

Example of Global and Local variable

Test - Notepad

File Edit Format View Help

```
<html>
<head>
<script language="JavaScript">
{
var a = 500;
function show(){
var a=200;
alert("a = "+ a);
}
show();
}
</script>
<title>Example Function Javascript</title>
</head>
<body>
</body>
</html>
```

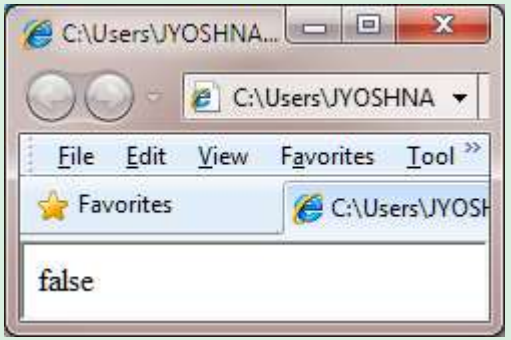
output



Example of "===" :

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
  var x = 10;
  document.write(x === "10");
</script>
</body>
</html>
```

OUTPUT :



JavaScript Objects

- JavaScript is an object-based language. Everything is an object in JavaScript.
- JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

• Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

1. JavaScript Object by object literal

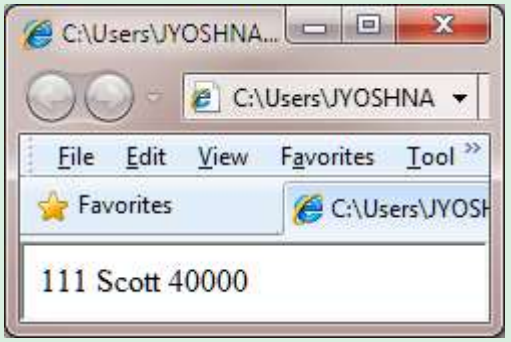
The syntax of creating object using object literal is given below:

Object = { property1:value1,property2:value2.....propertyN:valueN }

Where property and value is separated by : (colon).

Example of creating object in JavaScript

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
     emp={id:111,name:"Scott",salary:40000}
     document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```



2. By creating instance of Object

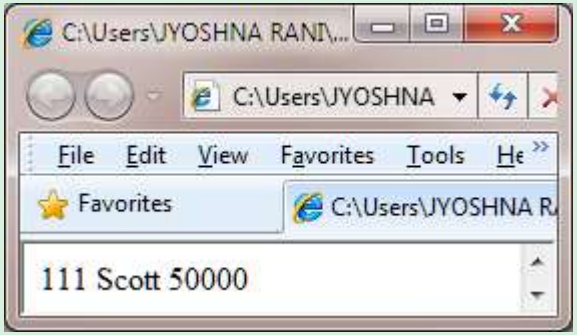
The syntax of creating object directly is given below:

var objectname=new Object();

Here, new keyword is used to create object.

Example of creating object directly

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
var emp=new Object();
emp.id=111;
emp.name="Scott";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```



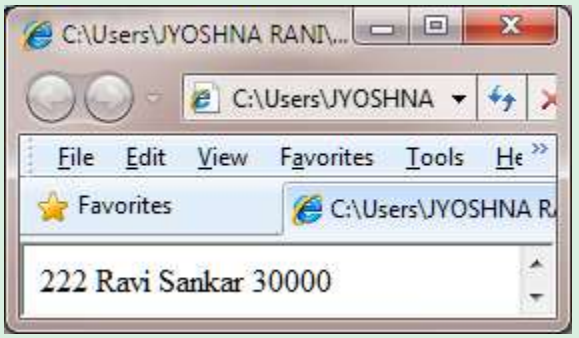
3. By using an Object constructor

We need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The this keyword refers to the current object.

Example of creating object by object constructor

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(222,"Ravi Sankar",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
</body>
</html>
```

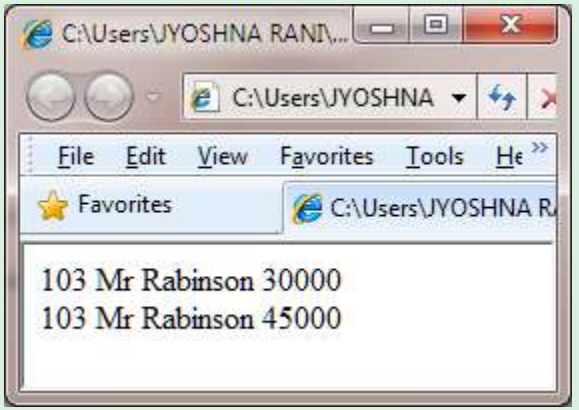


Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

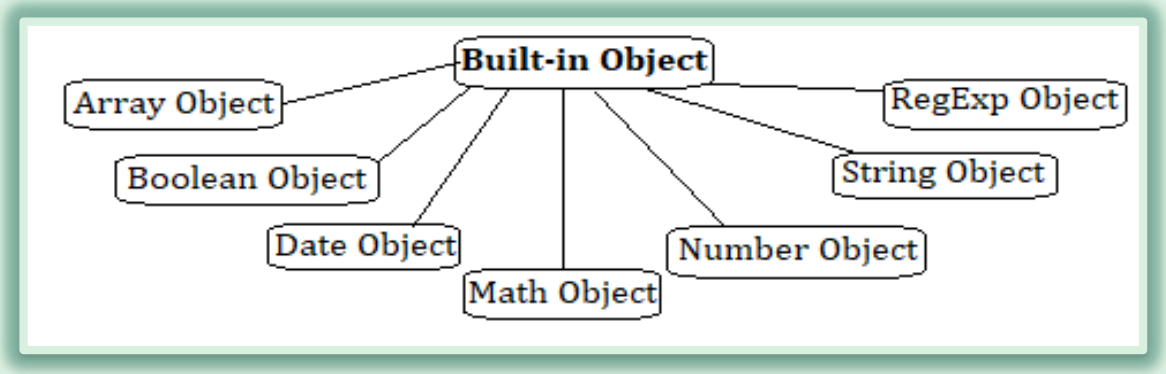
Example of defining method in object

```
<!DOCTYPE html><html><body><script language="Javascript">
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;

this.changeSalary=changeSalary;
function changeSalary(otherSalary){
this.salary=otherSalary;
}
}
e=new emp(103,"Mr Rabinson",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+" "+e.salary);
</script></body></html>
```



JavaScript Built-in Objects

- JavaScript has several built-in or core language objects. These built-in objects are available regardless of window content and operate independently of whatever page your browser has loaded.



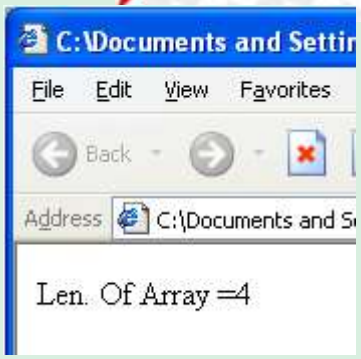
- **Array Object**
  - Multiple values are stored in a single variable using the Array object.
  - In JavaScript, an array can hold different types of data types in a single slot, which implies that an array can have a string, a number or an object in a single slot.
  - An Array object can be created by using following ways:
- **Using the Array Constructor:**
  - To create empty array when don't know the exact number of elements to be inserted in an array
    - var arrayname = new Array();

JavaScript Built-in Objects (conti.)

- To create an array of given size
  - **var arrayname = new Array(size);**
- To create an array with given elements
  - **var arrayname = new Array("element 1","element 2",……..,"element n");**
- **Using the Array Literal Notation:**
- To create empty array
  - **var arrayname =[ ];**
- To create an array when elements are given
  - **var arrayname =["element 1","element 2",……..,"element n"];**
- **Properties of the Array object**
  - **Length - Returns the number of elements in the array.**
  - **Constructor - Returns the function that created the Array object.**
  - **Prototype - Add properties and methods to an object.**
- **Methods of the Array object**
  - **reverse() - Reverses the array elements**
  - **concat() - Joins two or more arrays**
  - **sort() - Sort the elements of an array**
  - **push() - Appends one or more elements at the end of an array**
  - **pop() - Removes and returns the last element**
  - **shift() - Removes and returns the first element**
- **unshift(), join(), indexOf(), lastIndexOf(), slice(startindex, endindex) are some of the methods used in Array object.**

Example : To find the length of Array

output



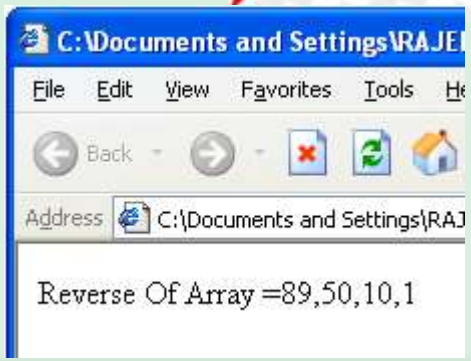
First - Notepad

File Edit Format View Help

```
<html>
        <head>
<script language="JavaScript">
{
        var a;
arr = new Array(1, 10 , 50 ,89);
        a = arr.length;
document.write("Len. Of Array ="+a);
}
</script>
        </head>
        <body></body>
</html>
```

Example Of reverse() method

output



First - Notepad

File Edit Format View Help

```
<html>
        <head>
<script language="JavaScript">
{
        var a;
arr = new Array(1, 10 , 50 ,89);
        a = arr.reverse();
document.write("Reverse Of Array ="+a);
}
</script>
        </head>
        <body></body>
</html>
```

JavaScript String

The JavaScript string is an object that represents a sequence of characters. There are two ways to create string in JavaScript

1. By string literal
2. By string object (using new keyword)

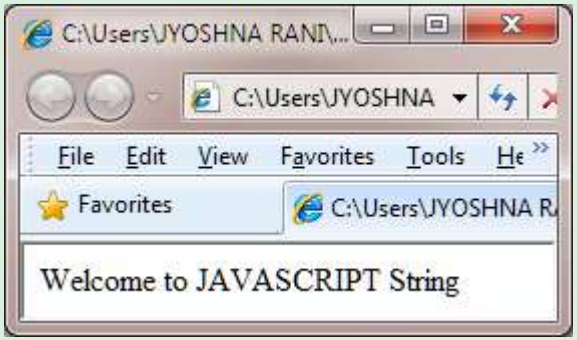
1. By string literal

The string literal is created using double quotes. The syntax of creating string using string literal is given below:

var stringname="string value";

Example of JAVASCRIPT String

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
var str="Welcome to JAVASCRIPT String";
document.write(str);
</script>
</body>
</html>
```



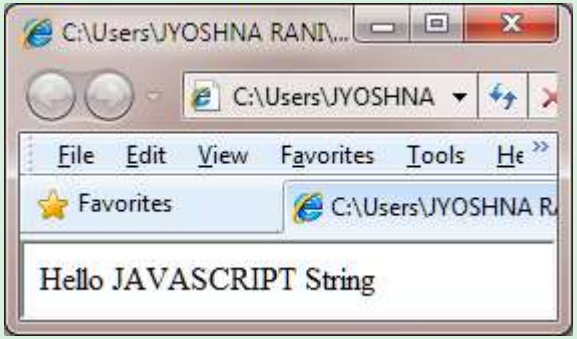
2. By string object (using new keyword)

The syntax of creating string object using new keyword is given by:

var stringname=new String("string literal"); Here, new keyword is used to create instance of string.

Example of creating string in JavaScript by new keyword

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
var sname=new String("Hello JAVASCRIPT String");
document.write(sname);
</script>
</body>
</html>
```

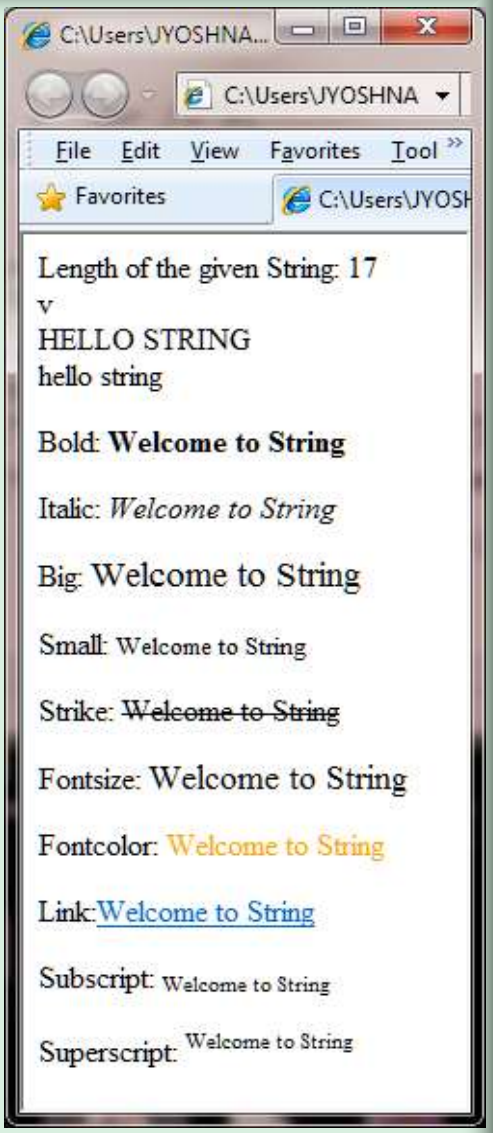


JavaScript String Methods

Methods	Description
charAt()	It provides the char value present at the specified index.
charCodeAt()	It provides the Unicode value of a character present at the specified index.
concat()	It provides a combination of two or more strings.
indexOf()	It provides the position of a char value present in the given string.
lastIndexOf()	It provides the position of a char value present in the given string by searching a character from the last position.
search()	It searches a specified regular expression in a given string and returns its position if a match occurs.
match()	It searches a specified regular expression in a given string and returns that regular expression if a match occurs.
replace()	It replaces a given string with the specified replacement.
substr()	It is used to fetch the part of the given string on the basis of the specified starting position and length.
substring()	It is used to fetch the part of the given string on the basis of the specified index.
slice()	It is used to fetch the part of the given string. It allows us to assign positive as well negative index.
toLowerCase()	It converts the given string into lowercase letter.
toLocaleLowerCase()	It converts the given string into lowercase letter on the basis of host?s current locale.
toUpperCase()	It converts the given string into uppercase letter.
toLocaleUpperCase()	It converts the given string into uppercase letter on the basis of host?s current locale.
toString()	It provides a string representing the particular object.
valueOf()	It provides the primitive value of string object.
split()	It splits a string into substring array, then returns that newly created array.
trim()	It trims the white space from the left and right side of the string.

Example of JavaScript String Methods

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
    var str = "JAVASCRIPT String";
    document.write('Length of the given String: ' + str.length);
    document.write("<br>");

    var str="javascript";
    document.write(str.charAt(2));
    document.write("<br>");

    var str = "Hello String";
    document.write(str.toUpperCase() + "<br />");
    document.write(str.toLowerCase());
    document.write("<br>");

    var str = "Welcome to String";
    document.write("<p> Bold: " + str.bold() + "</p>");
    document.write("<p> Italic: " + str.italics() + "</p>");
    document.write("<p> Big: " + str.big() + "</p>");
    document.write("<p> Small: " + str.small() + "</p>");
    document.write("<p> Strike: " + str.strike() + "</p>");
    document.write("<p> Fontsize: " + str.fontsize(4) + "</p>");
    document.write("<p> Fontcolor: " + str.fontcolor("orange") + "</p>");
    document.write("<p> Link:"+str.link("http://rajeebbal.com")+"</p>");
    document.write("<p> Subscript: " + str.sub() + "</p>");
    document.write("<p> Superscript: " + str.sup() + "</p>");
</script>
</body>
</html>
```



JavaScript Date Object

The JavaScript date object can be used to get year, month and day. We can display a timer on the webpage by the help of JavaScript date object.

We can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

Constructor

We can use four variant of Date constructor to create date object.

1. Date()
2. Date(milliseconds)
3. Date(dateString)
4. Date(year, month, day, hours, minutes, seconds, milliseconds)

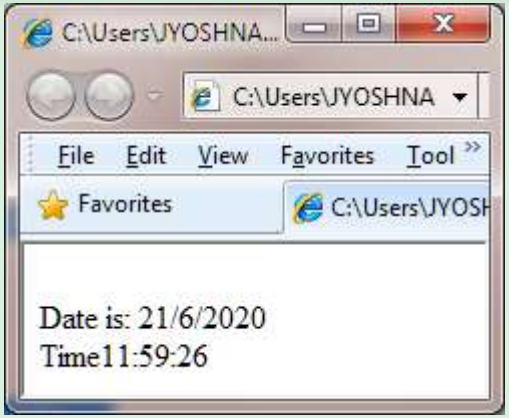
JavaScript Date Methods

Methods	Description
getDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.
getDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.
getFullYears()	It returns the integer value that represents the year on the basis of local time.
getHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.
getMilliseconds()	It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.
getMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.
getMonth()	It returns the integer value between 0 and 11 that represents the month on the basis of local time.
getSeconds()	It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.

getUTCDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of universal time.
getUTCDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of universal time.
getUTCFullYears()	It returns the integer value that represents the year on the basis of universal time.
getUTCHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of universal time.
getUTCMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of universal time.
getUTCMonth()	It returns the integer value between 0 and 11 that represents the month on the basis of universal time.
getUTCSeconds()	It returns the integer value between 0 and 60 that represents the seconds on the basis of universal time.
setDate()	It sets the day value for the specified date on the basis of local time.
setDay()	It sets the particular day of the week on the basis of local time.
setFullYears()	It sets the year value for the specified date on the basis of local time.
setHours()	It sets the hour value for the specified date on the basis of local time.
setMilliseconds()	It sets the millisecond value for the specified date on the basis of local time.
setMinutes()	It sets the minute value for the specified date on the basis of local time.
setMonth()	It sets the month value for the specified date on the basis of local time.
setSeconds()	It sets the second value for the specified date on the basis of local time.
setUTCDate()	It sets the day value for the specified date on the basis of universal time.
setUTCDay()	It sets the particular day of the week on the basis of universal time.
setUTCFullYears()	It sets the year value for the specified date on the basis of universal time.
setUTCHours()	It sets the hour value for the specified date on the basis of universal time.
setUTCMilliseconds()	It sets the millisecond value for the specified date on the basis of universal time.
setUTCMinutes()	It sets the minute value for the specified date on the basis of universal time.
setUTCMonth()	It sets the month value for the specified date on the basis of universal time.
setUTCSeconds()	It sets the second value for the specified date on the basis of universal time.
toDateString()	It returns the date portion of a Date object.
toISOString()	It returns the date in the form ISO format string.
toJSON()	It returns a string representing the Date object. It also serializes the Date object during JSON serialization.
toString()	It returns the date in the form of string.
toTimeString()	It returns the time portion of a Date object.
toUTCString()	It converts the specified date in the form of string using UTC time zone.
valueOf()	It returns the primitive value of a Date object.

JavaScript Date Example

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
    var date=new Date();
var day=date.getDate();
var month=date.getMonth()+1;
var year=date.getFullYear();
document.write("<br>Date is: "+day+"/"+month+"/"+year);

var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
document.write('<br> Time'+h+":"+m+":"+s);
</script>
</body>
</html>
```



JavaScript Math

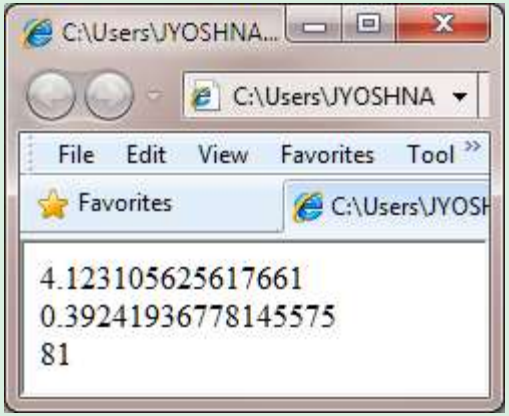
The JavaScript math object provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors.

JavaScript Math Methods

Methods	Description
abs()	It returns the absolute value of the given number.
acos()	It returns the arccosine of the given number in radians.
asin()	It returns the arcsine of the given number in radians.
atan()	It returns the arc-tangent of the given number in radians.
cbrt()	It returns the cube root of the given number.

ceil()	It returns a smallest integer value, greater than or equal to the given number.
cos()	It returns the cosine of the given number.
cosh()	It returns the hyperbolic cosine of the given number.
exp()	It returns the exponential form of the given number.
floor()	It returns largest integer value, lower than or equal to the given number.
hypot()	It returns square root of sum of the squares of given numbers.
log()	It returns natural logarithm of a number.
max()	It returns maximum value of the given numbers.
min()	It returns minimum value of the given numbers.
pow()	It returns value of base to the power of exponent.
random()	It returns random number between 0 (inclusive) and 1 (exclusive).
round()	It returns closest integer value of the given number.
sign()	It returns the sign of the given number
sin()	It returns the sine of the given number.
sinh()	It returns the hyperbolic sine of the given number.
sqrt()	It returns the square root of the given number
tan()	It returns the tangent of the given number.
tanh()	It returns the hyperbolic tangent of the given number.
trunc()	It returns an integer part of the given number.

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
    document.write(Math.sqrt(17)+'<br>');
    document.write(Math.random()+'<br>');
    document.write(Math.pow(3,4));
</script>
</body>
</html>
```



JavaScript Number Object

The JavaScript number object enables to represent a numeric value. It may be integer or floating-point. JavaScript number object follows IEEE standard to represent the floating-point numbers.

By the help of Number() constructor, we can create number object in JavaScript. For example:

```
var n=new Number(value);
```

If value can't be converted to number, it returns NaN(Not a Number) that can be checked by isNaN() method.

JavaScript Number Constants

Constant	Description
MIN_VALUE	returns the largest minimum value.
MAX_VALUE	returns the largest maximum value.
POSITIVE_INFINITY	returns positive infinity, overflow value.
NEGATIVE_INFINITY	returns negative infinity, overflow value.
NaN	represents "Not a Number" value.

JavaScript Number Methods

Methods	Description
isFinite()	It determines whether the given value is a finite number.
isInteger()	It determines whether the given value is an integer.
parseFloat()	It converts the given string into a floating point number.
parseInt()	It converts the given string into an integer number.
toExponential()	It returns the string that represents exponential notation of the given number.
toFixed()	It returns the string that represents a number with exact digits after a decimal point.
toPrecision()	It returns the string representing a number of specified precision.
toString()	It returns the given number in the form of string.

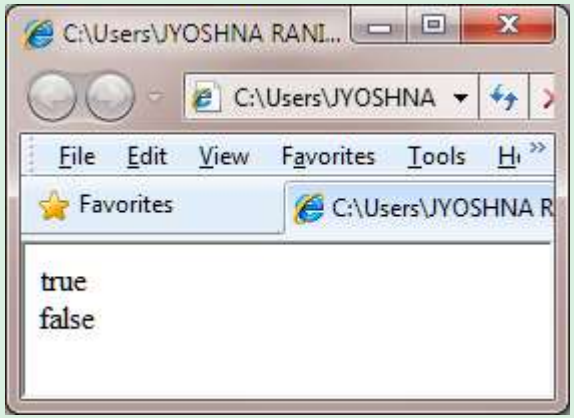
JavaScript Boolean

JavaScript Boolean is an object that represents value in two states: true or false. We can create the JavaScript Boolean object by Boolean() constructor as given below.

```
Boolean b=new Boolean(value);
```

The default value of JavaScript Boolean object is false.

```
<!DOCTYPE html>
<html>
<body>
<script language="Javascript">
    document.write(10<20);
    document.write("<br>");
    document.write(10<5);
</script>
</body>
</html>
```



JavaScript Events

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events :

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

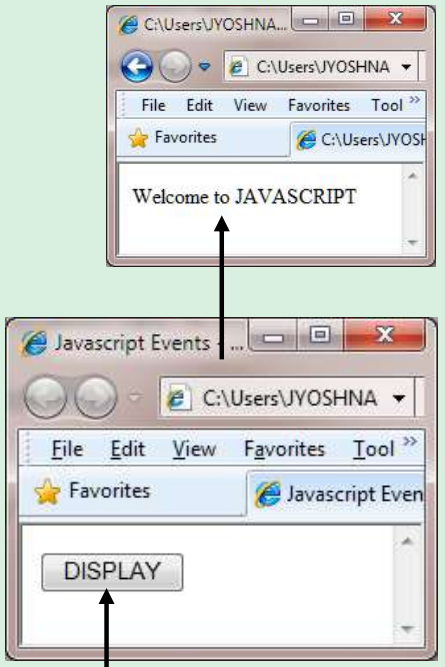
Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events :

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

```
<html>
<head> <title>Javascript Events</title> </head>
<body>
<script language="Javascript" type="text/Javascript">
    function clickevent()
    {
        document.write("Welcome to JAVASCRIPT");
    }
</script>
<form>
<input type="button" onclick="clickevent()" value="DISPLAY"/>
</form>
</body>
</html>
```



Click on Button

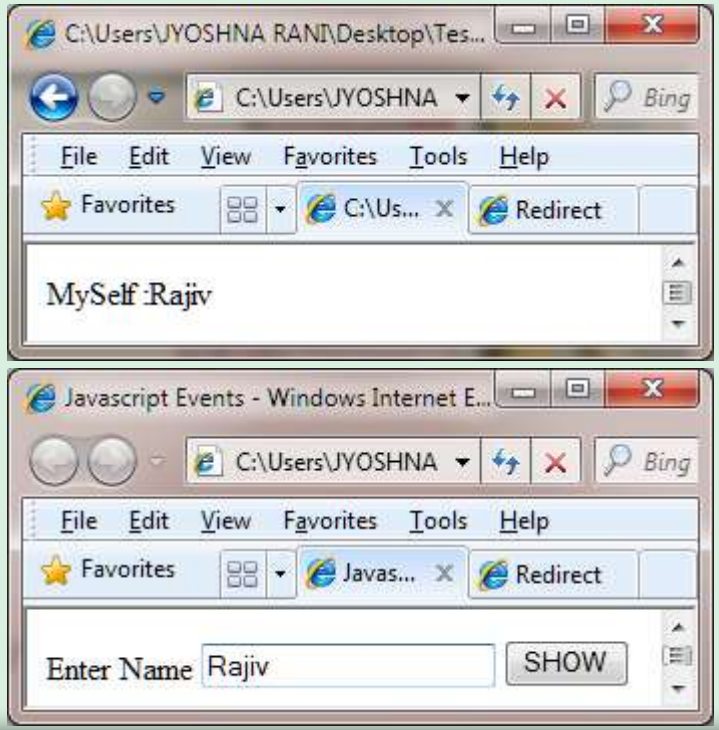
NOTE: We can simply use the value property of the input element to get the value of text input field.

SYNTAX :

formObject.textObject.value

EXAMPLE :

```
<html>
<head> <title>Javascript Events</title> </head>
<body>
<script language="Javascript">
     function clickevent()
    {
     var nm = myForm.tf.value
     document.write("MySelf :"+nm);
    }
</script>
<form name="myForm">
Enter Name <input type="text" name="tf">
<input type="button" onclick="clickevent()"
value="SHOW"/>
</form>
</body>
</html>
```



Document Object Model

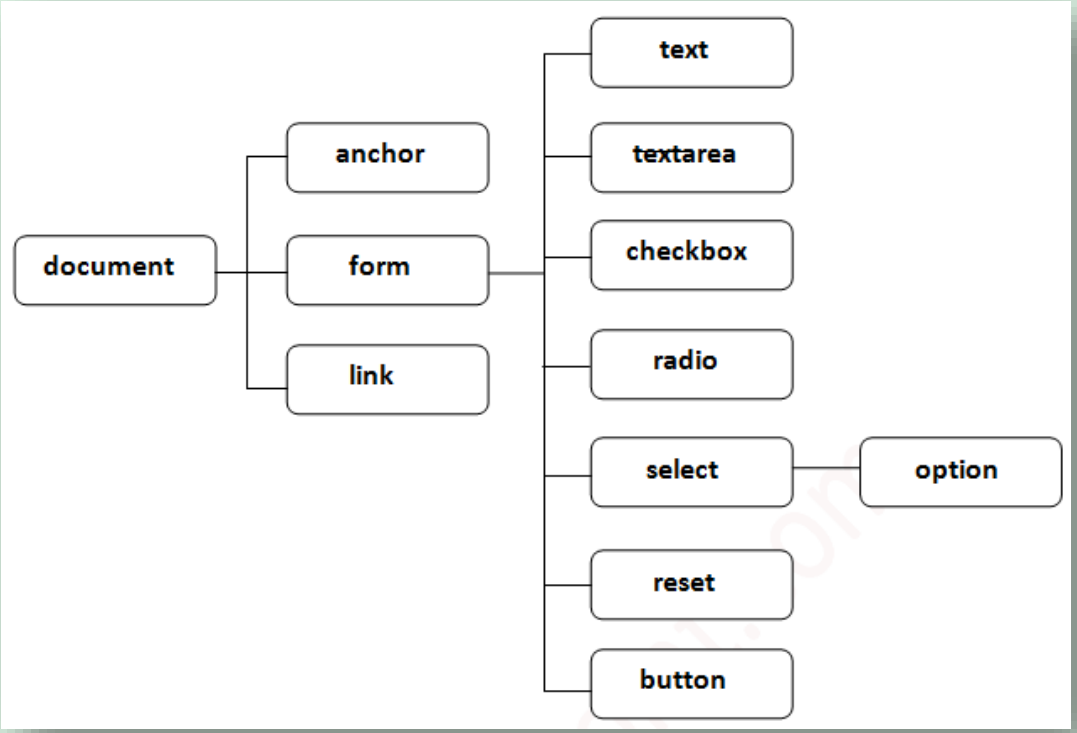
The document object represents the whole html document. When html document is loaded in the browser, it becomes a document object. It is the root element that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page. It is the object of window. So

window.document

Is same as

document

Properties of document object



Methods of document object

We can access and change the contents of document by its methods. The important methods of document object are as follows:

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

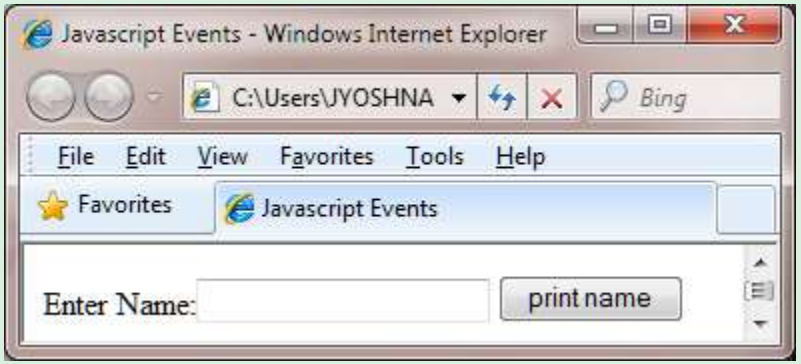
Accessing field value by document object

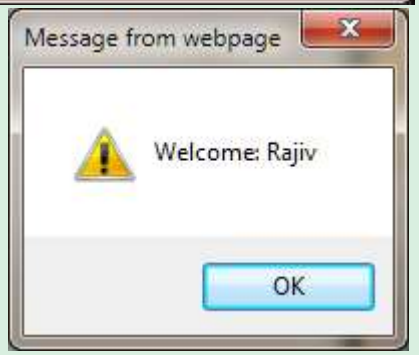
We are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field. Here,

- document is the root element that represents the html document.
- form1 is the name of the form.
- name is the attribute name of the input text.
- value is the property, that returns the value of the input text..

EXAMPLE :

```
<html>
<head> <title>Javascript Events</title> </head>
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>
<body>
<form name="form1">
Enter Name:<input type="text" name="name"/>
<input type="button" onclick="printvalue()" value="print name"/>
</form>
</body>
</html>
```

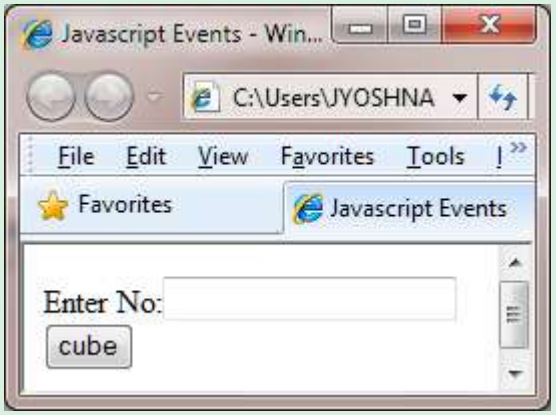


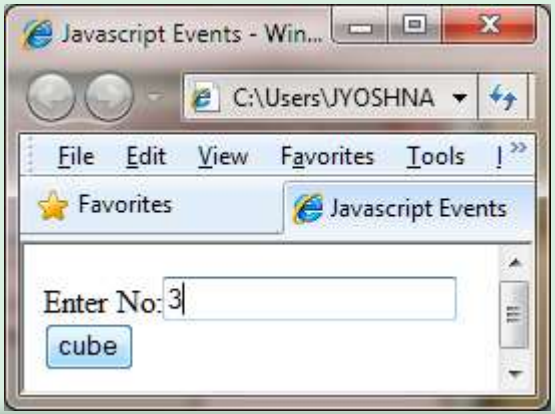


Javascript - document.getElementById() method

The document.getElementById() method returns the element of specified id. In the previous page, we have used document.form1.name.value to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

EXAMPLE :





```
<html>
<head> <title>Javascript Events</title> </head>
<script type="text/javascript">
function getcube(){
var number=document.getElementById("number").value;
alert(number*number*number);
}
</script>
<body>
<form>
Enter No:<input type="text" id="number" name="number"/><br/>
<input type="button" value="cube" onclick="getcube()"/>
</form>
</body>
</html>
```

Browser Object Model

The Browser Object Model (BOM) is used to interact with the browser. The default object of browser is window means we can call all the functions of window by specifying window or directly. For example:

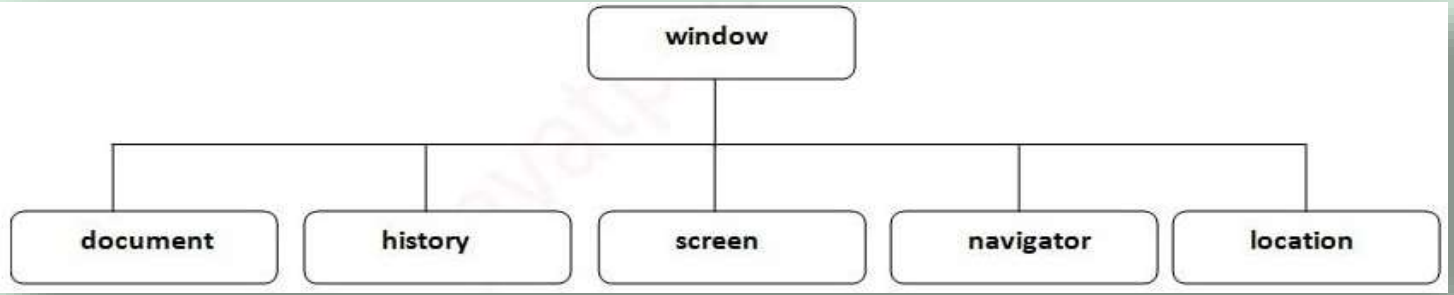
window.alert("hello javatpoint");

is same as

alert("hello javatpoint");

we can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth,

Note: The document object represents an html document. It forms DOM (Document Object Model).



Window Object

The window object represents a window in browser. An object of window is created automatically by the browser. Window is the object of browser, it is not the object of javascript. The javascript objects are string, array, date etc.

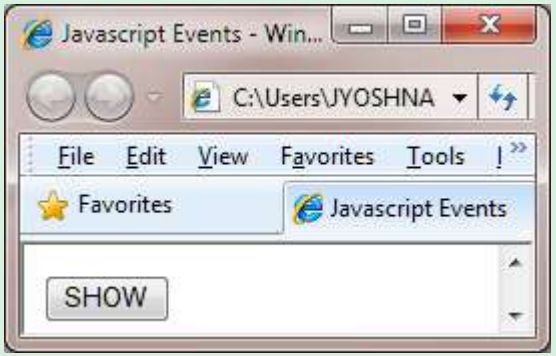
Note: if html document contains frame or iframe, browser creates additional window objects for each frame.

Methods of window object

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

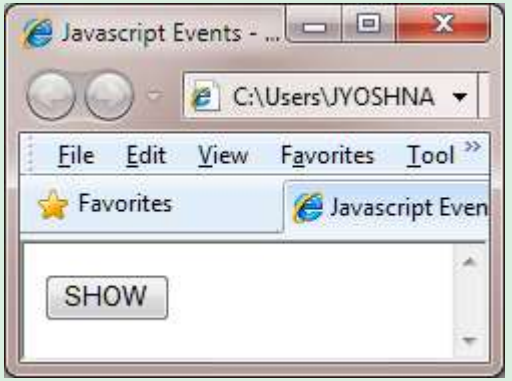
Example of open() in javascript
It displays the content in a new window.

```
<html>
<head> <title>Javascript Events</title> </head>
<script type="text/javascript">
function msg(){
open("http://www.google.com");
}
</script>
<body>
<form>
<input type="button" value="SHOW" onclick="msg()"/>
</form>
</body>
</html>
```



Example of setTimeout() in javascript
It performs its task after the given milliseconds.

```
<html>
<head> <title>Javascript Events</title> </head>
<script type="text/javascript">
function msg(){
setTimeout(
function(){
alert("Welcome to JAVASCRIPT after 1 seconds")
},1000);
}
</script>
<body>
<form>
<input type="button" value="SHOW" onclick="msg()"/>
</form>
</body>
</html>
```



Exception Handling in JavaScript

In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code

Types of Errors

There can be three types of errors in the code:

1. Syntax Error: When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.
2. Runtime Error: When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors are known as Exceptions. Thus, exception handlers are used for handling runtime errors.
3. Logical Error: An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.

Exception Handling Statements

There are following statements that handle if any exception occurs:

- throw statements
- try…catch statements
- try…catch…finally statements.

JavaScript try…catch

A try…catch is a commonly used statement in various programming languages. Basically, it is used to handle the error-prone part of the code. It initially tests the code for all possible errors it may contain, then it implements actions to tackle those errors (if occur). A good programming approach is to keep the complex code within the try…catch statements.

- try{} statement: Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{} block for taking suitable actions and handle the error. Otherwise, it executes the code written within.

- catch{} statement: This block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the built-in handler. This block executes only when any error-prone code needs to be handled in the try block. Otherwise, the catch block is skipped.

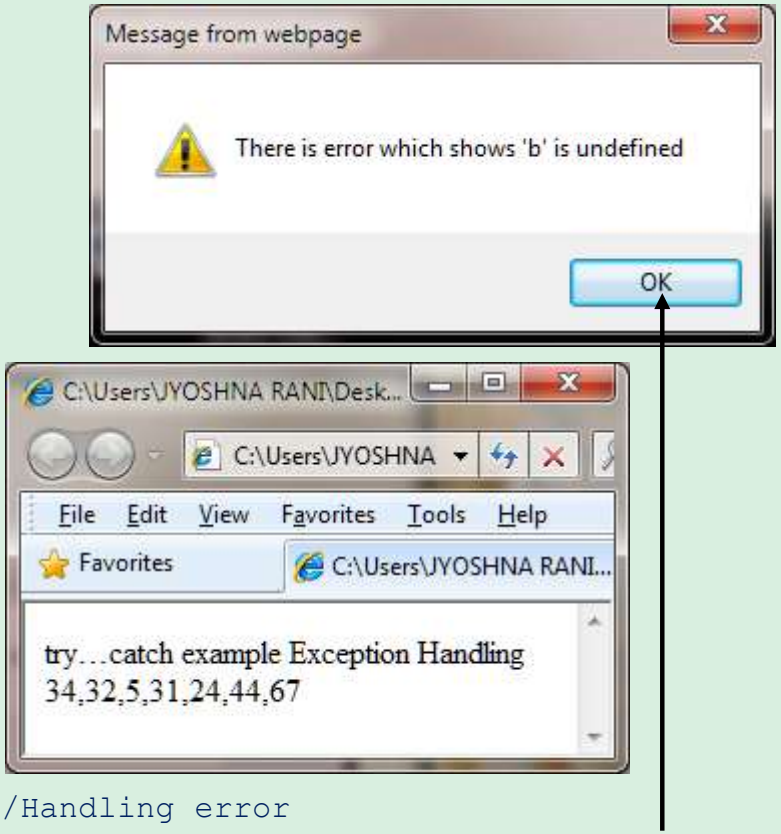
Note: catch {} statement executes only after the execution of the try {} statement. Also, one try block can contain one or more catch blocks.

Syntax:

```
try{
expression; } //code to be written.
catch(error){
expression; } // code for handling the error.
```


try…catch example

```
<html>
<head> Exception Handling</br></head>
<body>
<script>
try{
var a= ["34","32","5","31","24","44","67"];
document.write(a);
document.write(b);
}catch(e){
alert("There is error which shows "+e.message); //Handling error
}
</script>
</body>
</html>
```



Click on Alert message i.e, OK Button

Throw Statement

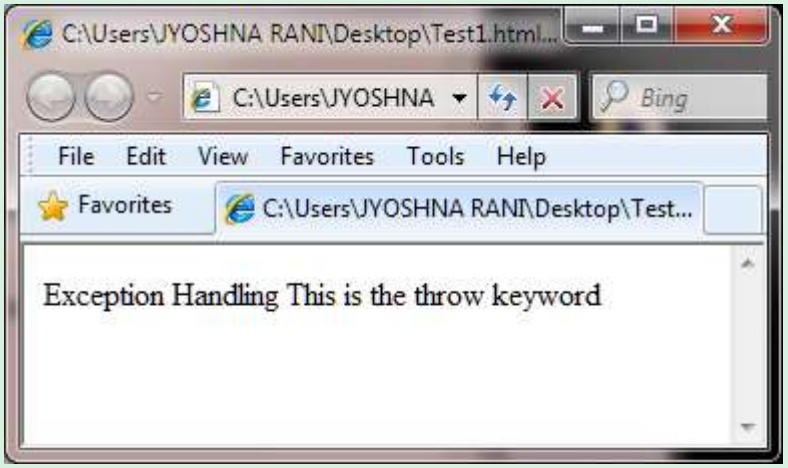
Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.

Syntax:

```
throw exception;
try…catch…throw syntax
try{
throw exception; // user can define their own exception
}
catch(error){
expression; }  // code for handling exception.
```

The exception can be a string, number, object, or boolean value.

throw example with try…catch




```
<html>
<head>Exception Handling</head>
<body>
<script>
try {
   throw new Error('This is the throw keyword'); //user-defined throw statement.
}
catch (e) {
  document.write(e.message); // This will generate an error message
}
</script>
</body>
</html>
```

try…catch…finally statements

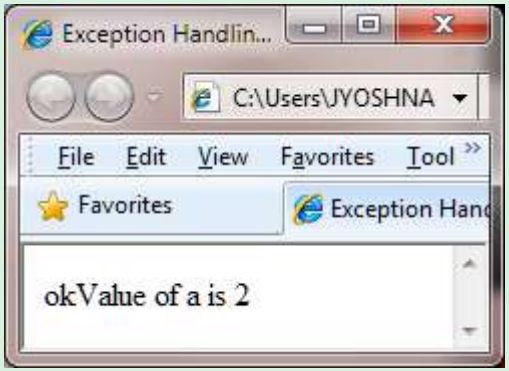
Finally is an optional block of statements which is executed after the execution of try and catch statements. Finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present, will definitely execute. It does not care for the output too.

Syntax:

```
try{
expression;
}
catch(error){
expression;
}
finally{
expression; } //Executable code
```

try…catch…finally example

```
<html>
<head><title>Exception Handling</title></head>
<body>
<script>
try{
var a=2;
if(a==2)
document.write("ok");
}
catch(Error){
document.write("Error found"+e.message);
}
finally{
document.write("Value of a is 2 ");
}
</script>
</body>
</html>
```



THANK YOU!