

WEB TECHNOLOGY

JavaScript

INDEX

- **Intro to script**
- Types
- JavaScript identifiers
- Operators
- Control & Looping structure
- Functions
- Objects
- Arrays
- Date
- Math
- String
- DOM
- Validations On Forms

INTRODUCTION

- JavaScript is the world's most popular programming language.
- It is the language for HTML, for the web, for servers, PCs, laptops, tablets, phones, and more.
- JavaScript was first known as LiveScript.
- **JavaScript is a Scripting Language**
 - A scripting language is a lightweight programming language.
 - Weakly typed programming language
 - JavaScript is programming code that can be inserted into HTML pages.
 - JavaScript code can be executed by all modern web browsers

WHY STUDY JAVASCRIPT?

1. **HTML** to define the content of web pages
 2. **CSS** to specify the layout of web pages
 3. **JavaScript** to program the behaviour of web pages
- JavaScript is a lightweight, interpreted programming language.
 - Designed for creating network-centric applications.
 - Complementary to and integrated with Java.
 - Complementary to and integrated with HTML.
 - Open and cross-platform.

ADVANTAGES OF JAVASCRIPT

- The merits of using JavaScript are:
 - Less server interaction: You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
 - Increased interactivity: You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
 - Richer interfaces: You can use JavaScript to include such items as drag and-drop components and sliders to give a Rich Interface to your site visitors

LIMITATIONS OF JAVASCRIPT

- **Can not debug:** Although some HTML editors allow for debugging, they are not as effective as editors for C or C++.
- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities
- The main problem or disadvantage in JavaScript is that the code is always visible to everyone anyone can view JavaScript code.
- No matter what proportion fast JavaScript interprets, JavaScript DOM (Document Object Model) is slow and can be a never-fast rendering with HTML.
- If the error occurs in JavaScript, it can stop rendering the whole website.

NOTE: Javascript always runs at client side.

1. Client side script-Script running on browser web browser

Javascript,vbscript,angular JS,JQuery

2. Server side script-Script which is running within the web server

ASP,-IIS

JSP-Tomcat

PHP-Apache

- CGI,perl
- **Programming VS scripting**
- Programming languages are those which use compiler and Scripting languages are those which use Interpreter.
- Programming languages returns executable file where as script not
- Programming languages are fast where as scripting is slow.

- **The <script> Tag**
- To insert a JavaScript into an HTML page, use the <script> tag.
- The <script> and </script> tells where the JavaScript starts and ends.
- The lines between the <script> and </script> contain the JavaScript:

<script language="javascript" type="text/javascript">

JavaScript code

</script>


```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
    document.write ("Hello World!")
//-->
</script>
<noscript>
    Sorry..JavaScript is needed to go ahead.
</noscript>
</body>
</html>
```

An optional HTML comment is added that surrounds JavaScript code. This is to save code from a browser that does not support JavaScript. The comment ends with a "`//>`".

INDEX

- Intro to script
- **Types**
- JavaScript identifiers
- Operators
- Control & Looping structure
- Functions
- Objects
- Arrays
- Date
- Math
- String
- DOM
- Validations On Forms

PLACEMENT

- Script in `<head>...</head>` section.
- Script in `<body>...</body>` section.
- Script in `<body>...</body>` and `<head>...</head>` sections.
- Script in an external file and then include in `<head>...</head>` section.

JAVASCRIPT IN <HEAD>...</HEAD> SECTION

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as function

```
<html>
```

```
<head>
```

```
<script type="text/javascript">
```

```
<!--
```

```
function sayHello() {  
    alert("Hello World")  
}
```

```
//-->
```

```
</script>
```

```
</head>
```

```
<body>
```

Click here for the result

```
<input type="button" onclick="sayHello()" value="Say Hello" />
```

```
</body>
```

```
</html>
```

Click here for the result

JAVASCRIPT IN <BODY>...</BODY> SECTION

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<p>This is web page body </p>
</body>
</html>
```

Hello World

This is web page body

JAVASCRIPT IN <BODY> AND <HEAD> SECTIONS

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

Hello World

JAVASCRIPT IN EXTERNAL FILE

```
<html>
```

```
<head>
```

```
<script type="text/javascript" src="filename.js" >
```

```
</script>
```

```
</head>
```

```
<body>
```

```
.....
```

```
</body>
```

```
</html>
```

```
filename.js  
function sayHello() {  
    alert("Hello World")  
}
```

JAVASCRIPT OUTPUT

- JavaScript can "display" data in different ways:
 - **window.alert()**
 - Writing into the HTML output using **document.write()**
 - Writing into an HTML element, using **getElementById()** with property **innerHTML**
 - Writing into the browser console, using **console.log()**

NOTE: JavaScript does NOT have any built-in print or display functions.

INNERHTML

- To access an HTML element, JavaScript can use the **document.getElementById(id)** method.
- The **id** attribute defines the HTML element. The innerHTML property defines the HTML content:

INNERHTML

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>My First Web Page</h1>
```

```
<p>My First Paragraph</p>
```

My First Web Page

My First Paragraph.

11

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

```
<script>
```

```
document.getElementById("demo").innerHTML = 5 + 6;
```

```
</script>
```

```
</body>
```

```
</html>
```

USING DOCUMENT.WRITE()

- For testing purposes, it is convenient to use **document.write()**:
- Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>My First Web Page</h2>
```

```
<p>My first paragraph.</p>
```

```
<script>
```

```
document.write(5 + 6);
```

```
</script>
```

```
</body>
```

```
</html>
```

My First Web Page

My first paragraph.

11

USING WINDOW.ALERT()

- You can use an alert box to display data/warning messages:
- Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>My First Web Page</h2>
```

```
<p>My first paragraph.</p>
```

```
<script>
```

```
window.alert(5 + 6);
```

```
alert("pls enter number");
```

```
</script>
```

```
</body>
```

```
</html>
```

This page says

11

OK

This page says

pls enter number

OK

USING CONFIRM()

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Click the button to display a confirm box.</p>
```

```
<script>
```

```
var res=confirm("Click yes/no");
```

```
alert("User clicked on " +res);
```

```
</script>
```

```
</body>
```

```
</html>
```

This page says

Click yes/no

OK

Cancel

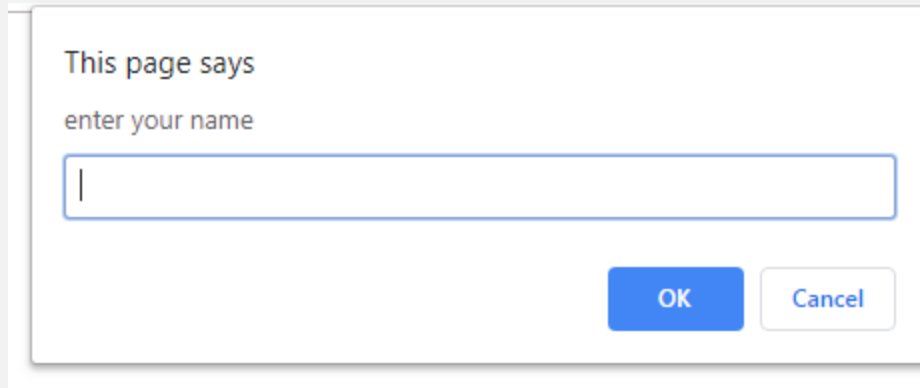
This page says

User ckicked on true

OK

USING PROMPT ()

```
<!DOCTYPE html>  
<html>  
<body>  
<script>  
var name=prompt("enter your name");  
document.write("You entered " +name);  
</script>  
</body>  
</html>
```

A screenshot of a browser's prompt dialog box. The dialog has a title bar and contains the text "This page says" followed by "enter your name". Below this text is a single-line text input field with a blue border and a vertical cursor. At the bottom right of the dialog are two buttons: a blue "OK" button and a white "Cancel" button with a blue border.

This page says
enter your name

OK Cancel

You entered aaaa

USING CONSOLE.LOG()

- For debugging purposes, you can use the **console.log()** method to display data.

- Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>Activate debugging with F12</h2>
```

```
<p>Select "Console" in the debugger menu. Then click Run again.</p>
```

```
<script>
```

```
console.log(5 + 6);
```

```
</script>
```

```
</body>
```

```
</html>
```

INDEX

- Intro to script
- Types
- **JavaScript identifiers**
- Operators
- Control & Looping structure
- Functions
- Objects
- Arrays
- Date
- Math
- String
- DOM
- Validations On Forms

JAVASCRIPT IDENTIFIERS

- The general rules for constructing names for variables (unique identifiers) are:
 - Names can contain letters, digits, underscores, and dollar signs.
 - Names must begin with a letter
 - Names can also begin with \$ and _.
 - Names are case sensitive (y and Y are different variables)
 - Reserved words (like JavaScript keywords) cannot be used as names
 - Example: `var number=10;`

JAVASCRIPT RESERVED WORDS

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

JAVASCRIPT IDENTIFIERS

- Use the **var** keyword only for declaration or initialization.
- JavaScript is untyped language.
- JavaScript variable can hold a value of any data type.
- The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically

```
<script type="text/javascript">
```

```
var name = "Ali";
```

```
var money;
```

```
money = 2000.50;
```

```
</script>
```

JAVASCRIPT DATATYPES

- JavaScript allows you to work with three primitive data types:
 - Numbers, e.g., 123, 120.50 etc.
 - Strings of text, e.g. "This text string" etc.
 - Boolean, e.g. true or false.
- JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value.
- JavaScript supports a composite data type known as object

JAVASCRIPT COMMENTS AND ERRORS

- Single Line Comments only starts with //
- Multi-line comments start with /* and end with */.
- ERRORS:

1. Value = undefined

A variable declared without a value will have the value **undefined**.

2. Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

```
var carName = "Volvo";
```

```
var carName;
```

```
<html> <head> <title> dgdfg </title>
<script>
var x=10;
document.write("Value is "+x+"<br>" );
var x=12.45;
document.write("Value is "+x+"<br>" );
var x="gdfg fhfg";
document.write("Value is "+x+"<br>" );
document.write("Value is "+ (100>102)+"<br>");
var z;
document.write("Value is "+ z +"<br>" );
var x=null;
document.write("Value is "+x+"<br>" );
</script>
</head>
<body>
</body>
</html>
```

Value is 10
Value is 12.45
Value is gdfg fhfg
Value is false
Value is undefined
Value is null

JAVASCRIPT VARIABLE SCOPE

- **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.
- Within the body of a function, a local variable takes precedence over a global variable with the same name.

JAVASCRIPT VARIABLE SCOPE

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


INDEX

- Intro to script
- Types
- JavaScript identifiers
- **Operators**
- Control & Looping structure
- Functions
- Objects
- Arrays
- Date
- Math
- String
- DOM
- Validations On Forms

OPERATORS

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

ARITHMETIC OPERATORS

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var c = "Test";
var linebreak = "<br />";

document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);

document.write("a - b = ");
result = a - b;
document.write(result);
document.write(linebreak);
```

```
document.write("a % b = ");
result = a % b;
document.write(result);
document.write(linebreak);

document.write("a + b + c = ");
result = a + b + c;
document.write(result);
document.write(linebreak);

a = a++;
document.write("a++ = ");
result = a++;
document.write(result);
document.write(linebreak);

b = b--;
document.write("b-- = ");
result = b--;
document.write(result);
document.write(linebreak);
//-->
</script>
```

JAVASCRIPT ASSIGNMENT OPERATORS

Operator	Example	Same As
=	$x = y$	$x = y$
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
/=	$x /= y$	$x = x / y$
%=	$x \% = y$	$x = x \% y$

JAVASCRIPT COMPARISON OPERATORS

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

LOGICAL OPERATORS

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

Conditional (Ternary) Operator

Conditional Operator (? :)

```
var voteable = (age < 18) ? "Too young" : "Old enough";
```

TYPEOF OPERATOR

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"


```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
var a = 10;
```

```
var b = "String";
```

```
var linebreak = "<br />";
```

```
document.write("a=" + a);
```

```
document.write(linebreak);
```

```
document.write("b=" + b);      document.write(linebreak);
```

```
result = (typeof b == "string" ? "B is String" : "B is Numeric");
```

```
document.write("Result => ");
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
result = (typeof a == "string" ? "A is String" : "A is Numeric");
```

```
document.write("Result => ");
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
</script>
```

a=10

b=String

Result => B is String

Result => A is Numeric

STRING CONCATENATION

```
var x = "5" + 2 + 3;
```

If you add a number to a string, the number will be treated as string, and concatenated.

So the result is : 523

```
var x = "John" + " " + "Doe";
```

Result is: John Doe

JAVASCRIPT STRING OPERATORS

- `txt1 = "John";`
`txt2 = "Doe";`
`txt3 = txt1 + " " + txt2;`

Result: John Doe

- `txt1 = "What a very ";`
`txt1 += "nice day";`

Result: What a very nice day

INDEX

- Intro to script
- Types
- JavaScript identifiers
- Operators
- **Control & Looping structure**
- Functions
- Objects
- Arrays
- Date
- Math
- String
- DOM
- Validations On Forms

CONDITIONAL STATEMENTS

- Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.
- In JavaScript we have the following conditional statements:
- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

IF STATEMENT

- If Statement
- Use the if statement to execute some code only if a specified condition is true.
- Syntax
- ```
if (condition)
{
 code to be executed if condition is true
}
```

## IF...ELSE STATEMENT

```
if (expression){
```

```
 Statement(s) to be executed if expression is true
```

```
}
```

```
else
```

```
{
```

```
 Statement(s) to be executed if expression is false
```

```
}
```

## IF...ELSE IF... STATEMENT

- Use the if....else if...else statement to select one of several blocks of code to be executed.
- Syntax
- *if (condition1)*  
    {  
    *code to be executed if condition1 is true*  
    }  
*else if (condition2)*  
    {  
    *code to be executed if condition2 is true*  
    }  
*else*  
    {  
    *code to be executed if neither condition1 nor condition2 is true*  
    }





# SWITCH-CASE

```
switch (expression)
{
 case condition 1: statement(s)
 break;
 case condition 2: statement(s)
 break;
 ...
 case condition n: statement(s)
 break;
 default: statement(s)
}
```

```
<html>

<body>

 <script type="text/javascript">

var grade='A';

document.write("Entering switch block
");

switch (grade)

{

 case 'A': document.write("Good job
");

 break;

 case 'B': document.write("Pretty good
");

 break;

 case 'C': document.write("Passed
");

 break;

 case 'D': document.write("Not so good
");

 break;

 case 'F': document.write("Failed
");

 break;

 default: document.write("Unknown grade
")
```

Entering switch block  
Good job  
Exiting switch block

## THE BREAK KEYWORD

- When the JavaScript code interpreter reaches a **break** keyword, it breaks out of the switch block.
- This will stop the execution of more code and case testing inside the block.



# THE WHILE LOOP

```
while (expression){
 Statement(s) to be executed if expression is true
}
```

```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
var count = 0;
```

```
document.write("Starting Loop ");
```

```
while (count < 5){
```

```
 document.write("Current Count : " + count + "
");
```

```
 count++;
```

```
}
```

```
document.write("Loop stopped!");
```

```
</script>
```

```
</body>
```

```
</html>
```

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Loop stopped!

# THE DO...WHILE LOOP

```
do{
 Statement(s) to be executed;
} while (expression);
```



```
<html>
<body>
<script type="text/javascript">
var count = 0;
document.write("Starting Loop" + "
");
do{
 document.write("Current Count : " + count + "
");
 count++;
}while (count < 5);
document.write ("Loop stopped!");
</script>
</body>
</html>
```

Starting Loop  
Current Count : 0  
Current Count : 1  
Current Count : 2  
Current Count : 3  
Current Count : 4  
Loop stopped!

# THE FOR LOOP

```
for (initialization; test condition; iteration
statement){
 Statement(s) to be executed if test
 condition is true
}
```

```
<html>
<body>
 <script type="text/javascript">

var count;
document.write("Starting Loop" + "
");
for(count = 0; count < 5; count++){
 document.write("Current Count : " + count);
 document.write("
");
}
document.write("Loop stopped!");
</script>

</body>
</html>
```

## Output

Starting Loop  
Current Count : 0  
Current Count : 1  
Current Count : 2  
Current Count : 3  
Current Count : 4  
Loop stopped!

## JAVASCRIPT BREAK AND CONTINUE

- The break statement "jumps out" of a loop.
- The continue statement "jumps over" one iteration in the loop.

```
<html>
<body>
<script type="text/javascript">
var x = 1;
document.write("Entering the loop
 ");
while (x < 20)
{
 if (x == 5){
 break; // breaks out of loop completely
 }
 x = x + 1;
 document.write(x + "
");
}
document.write("Exiting the loop!
 ");
</script>
</body>
</html>
```

OUTPUT:

Entering the loop

1

2

3

4

Exiting the loop

```
<html>
<body>
<script type="text/javascript">
var x = 1;
document.write("Entering the loop
 ");
while (x < 10)
{
 x = x + 1;
 if (x == 5){
 continue; // skip rest of the loop body
 }
 document.write(x + "
");
}
document.write("Exiting the loop!
 ");
</script>
</body>
```

Output:  
Entering the loop  
2  
3  
4  
6  
7  
8  
9  
10  
Exiting the loop

# INDEX

- Intro to script
- Types
- JavaScript identifiers
- Operators
- Control & Looping structure
- **Functions**
- Objects
- Arrays
- Date
- Math
- String
- Boolean
- DOM
- Validations On Forms

# FUNCTIONS

1. Code reusability: We can call a function several times so it save coding.
2. Less coding: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

- JavaScript functions are **defined** with the **function** keyword.
- Syntax:

```
function function_name(parameter-list)
{
 statements
}
```



# FUNCTIONS

- Example

```
<script type="text/javascript">
```

```
function sayHello()
```

```
{
```

```
 alert("Hello there");
```

```
}
```

```
</script>
```

# CALLING A FUNCTION

```
<html>
<head>
<script type="text/javascript">
function sayHello()
{
 document.write ("Hello there!");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello">
</form>
</body>
</html>
```

Click the following button to call the function

Say Hello

# FUNCTION PARAMETERS

```
<html>
<head>
<script type="text/javascript">
function sayHello(name, age)
{
 document.write (name + " is " + age + " years old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>
</body>
</html>
```

Click the following button to call the function

Say Hello

# THE RETURN STATEMENT

```
<html>
<head>
<script type="text/javascript">
function concatenate(first, last)
{
 var full;
 full = first + last;
 return full;
}
function secondFunction()
{
 var result;
 result = concatenate('Zara', 'Ali');
 document.write (result);
}
</script>
</head>
```

```
<body>
<p>Click the following button to call the
function</p>
<form>
<input type="button"
onclick="secondFunction()" value="Call
Function">
</form>
</body>
</html>
```

Click the following button to call the functi

Call Function

# FUNCTION () CONSTRUCTOR

- The function statement is not the only way to define a new function; you can define your function dynamically using Function() constructor along with the new operator.
- Syntax:

```
<script type="text/javascript">
```

```
var variablename = new Function(Arg1,Arg2..., "Function Body");
```

```
</script>
```

```
<html>
```

```
<head>
```

```
<script type="text/javascript">
```

```
var func = new Function("x", "y", "return x*y;");
```

```
function secondFunction(){
```

```
 var result;
```

```
 result = func(10,20);
```

```
document.write (result);
```

```
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<p>Click the following button to call the function</p>
```

```
<form>
```

```
<input type="button" onclick="secondFunction()" value="Call Function">
```

```
</form>
```

```
</body>
```

Click the following button to call the function

Call Function

# FUNCTION LITERALS/EXPRESSIONS

- A JavaScript function can also be defined using an **expression**.
- A function literal is an expression that defines an unnamed function.
- A function expression can be stored in a variable:

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

```
<script>
```

```
var x = function (a, b) {return a * b} ;
```

```
document.getElementById("demo").innerHTML = x;
```

```
</script>
```

```
<html>
<head>
<script type="text/javascript">
var func = function(x,y){ return x*y };
function secondFunction(){
var result;
result = func(10,20);
document.write (result);
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```



# INDEX


- Intro to script
- Types
- JavaScript identifiers
- Operators
- Control & Looping structure
- Functions
- **Objects**
- Arrays
- Date
- Math
- String
- Boolean
- DOM
- Validations On Forms

# OBJECT

- JavaScript is an Object Oriented Programming (OOP) language.
- Four basic capabilities :Encapsulation,Aggregation, Inheritance & Polymorphism
- Encapsulation: the capability to store related information, whether data or methods, together in an object.
- Aggregation: the capability to store one object inside another object.
- Inheritance: the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- Polymorphism: the capability to write one function or method that works in a variety of different ways.
- In JavaScript, almost "everything" is an object.
- In JavaScript, objects are king. If you understand objects, you understand JavaScript.
- Booleans can be objects (if defined with the **new** keyword).
- Numbers can be objects (if defined with the **new** keyword).
- Strings can be objects (if defined with the **new** keyword).
- Dates are always objects.
- Maths are always objects.
- Regular expressions are always objects.
- Arrays are always objects.
- Functions are always objects.
- Objects are always objects.

# OBJECT

- In real life, a car is an **object**.
- A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods
	<p>car.name = Fiat</p> <p>car.model = 500</p> <p>car.weight = 850kg</p> <p>car.color = white</p>	<p>car.start()</p> <p>car.drive()</p> <p>car.brake()</p> <p>car.stop()</p>

# OBJECTS

- All user-defined objects and built-in objects are descendants of an object called **Object**.
- The **new** operator is used to create an instance of an object.
- To create an object, the new operator is followed by the constructor method
  - `var employee = new Object();`
  - `var books = new Array("C++", "Perl", "Java");`
  - `var day = new Date("August 15, 1947");`

# OBJECTS

- Accessing JavaScript Properties
  - *objectName.property*      // employee.fname
  - *objectName["property"]*      // employee["fname"]
  - *objectName[expression]*      // x = "fname"; employee[x]

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
 var book = new Object(); // Create the object
 book.subject = "Operating System"; // Assign properties to the object
 book.author = "Stalling";
</script>
</head>
<body>
<script type="text/javascript">
 document.write("Book name is : " + book.subject + "
");
 document.write("Book author is : " + book.author + "
");
</script>
</body>
</html>
```

Book name is : Operating System  
Book author is : Stalling

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
function book(title, author){
 this.title = title;
 this.author = author;
}
</script>
</head>
<body>
<script type="text/javascript">
var myBook = new book("OS", "Stalling");
document.write("Book title is : " + myBook.title + "
");
document.write("Book author is : " + myBook.author + "
");
</script>
</body>
</html>
```

Output:

Book name is : Operating System

Book author is : Stalling

# DEFINING METHODS FOR AN OBJECT

<title>User-defined objects</title>

<script type="text/javascript">

// Define a function which will work as a method

function addPrice(amount){

  this.price = amount; }

function book(title, author){

  this.title = title;

  this.author = author;

  this.addPrice = addPrice; // Assign that method as property.

}

</script> </head>

<body>

<script type="text/javascript">

var myBook = new book("OS", "Stalling");

myBook.addPrice(100);

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

document.write("Book price is : " + myBook.price + "<br>");

</script>

</body>



**<script>**

```
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,"Sonoo Jaiswal",30000);
```

```
document.write(e.id+" "+e.name+" "+e.salary);
```

```
e.changeSalary(45000);
```

```
document.write("
" + e.id + " " + e.name + " " + e.salary);
```

**</script>**

# INDEX

- Intro to script
- Types
- JavaScript identifiers
- Operators
- Control & Looping structure
- Functions
- Objects
- **Arrays**
- Date
- Math
- String
- DOM
- Event handling
- Validations On Forms

# WHAT IS AN ARRAY?

- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
  - `var car1="Honda";`  
`var car2="Mercedes";`  
`var car3="BMW";`
- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
- The solution is an array!
- An array can hold many values under a single name, and you can access the values by referring to an index number.

# JAVASCRIPT ARRAYS

- `var cars = ["Honda", "Mercedes", "BMW"];`
- `var cars = new Array ("Honda", "Mercedes", "BMW");`
- The two examples above do exactly the same.
- There is no need to use `new Array()`.  
For simplicity, readability and execution speed, use the first one.
- This statement accesses the value of the first element in cars:  
`var name = cars[0];`
- This statement modifies the first element in cars:  
`cars[0] = "Opel";`

# JavaScript Arrays cont...

- You Can Have Different Objects in One Array
- JavaScript variables can be objects. Arrays are special kinds of objects.
- Because of this, you can have variables of different types in the same Array.
- You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:
- `myArray[0] = Date.now;`  
`myArray[1] = myFunction;`  
`myArray[2] = myCars;`



- 1) Literal
- `Var arr=["abc","pqr","xyz"]`

- 2. Regular

- `Var arr=new Array();`

`Arr[0]="abc"`

`Arr[1]="pqr"`

`Arr[2]="xyz"`

- 3. Condensed

- `Var arr=new Array("abc","pqr","xyz");`

# JavaScript Arrays cont...

- Arrays are Objects
- Arrays are a special type of objects.
- The **typeof** operator in JavaScript returns "object" for arrays.

Array:

```
var person = ["John", "Doe", 46]; // Here person[0] returns john
```

Object:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

```
 // Here, person.firstname returns John
```

# ARRAY OBJECT PROPERTIES

- The Array object is used to store multiple values in a single variable.

Property	Description
constructor	Returns a reference to the array function that created the object.
index	The property represents the zero-based index of the match in the string
input	This property is only present in arrays created by regular expression matches.
length	Reflects the number of elements in an array.
prototype	The prototype property allows you to add properties and methods to an object.



# ARRAY PROPERTIES

## I. constructor

- JavaScript array constructor property returns a reference to the array function that created the instance's prototype.
- Syntax :
  - array.constructor
- Return Value: Returns the function that created this object's instance
- Ex:

```
<script type="text/javascript">
```

```
 var arr = new Array(10, 20, 30);
```

```
 document.write("arr.constructor is:" + arr.constructor);
```

```
</script>
```

# ARRAY PROPERTIES

## 2.The length Property

- JavaScript array length property returns an unsigned, 32-bit integer that specifies the number of elements in an array.
- Syntax :
  - `array.length`
- Return Value Returns the length of an array.
- Ex:

```
<script type="text/javascript">
 var arr = new Array(10, 20, 30);
 document.write("arr.length is:" + arr.length);
</script>
```

# ARRAY PROPERTIES

## 3. Prototype

- The prototype property allows you to add properties and methods to any object (Number, Boolean, String, Date, etc.).
- Its syntax is as follows:
  - `object.prototype.name = value`

```
<html>

<head>

<title>Object: prototype</title>

<script type="text/javascript">

function book(title, author){
 this.title = title;
 this.author = author;
}

</script> </head> <body>

<script type="text/javascript">

var myBook = new book("OS", "Stalling");

book.prototype.price = null;

myBook.price = 100;

document.write("Book title is : " + myBook.title + "
");

document.write("Book author is : " + myBook.author + "
");

document.write("Book price is : " + myBook.price + "
");

</script> </body> </html>
```

## OUTPUT:

Book title is : OS

Book author is : Stalling

Book price is : 100

# Array Object Methods

Method	Description
<u>concat()</u>	Joins two or more arrays, and returns a copy of the joined arrays
<u>indexOf()</u>	Search the array for an element and returns its position
<u>join()</u>	Joins all elements of an array into a string
<u>lastIndexOf()</u>	Search the array for an element, starting at the end, and returns its position
<u>pop()</u>	Removes the last element of an array, and returns that element
<u>push()</u>	Adds new elements to the end of an array, and returns the new length
<u>reverse()</u>	Reverses the order of the elements in an array
<u>shift()</u>	Removes the first element of an array, and returns that element
<u>slice()</u>	Selects a part of an array, and returns the new array
<u>sort()</u>	Sorts the elements of an array
<u>splice()</u>	Adds/Removes elements from an array
<u>toString()</u>	Converts an array to a string, and returns the result
<u>unshift()</u>	Adds new elements to the beginning of an array, and returns the new length
<u>valueOf()</u>	Returns the primitive value of an array

## CONCAT ()

```
<html>
```

```
<head>
```

```
<title>JavaScript Array concat Method</title>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
 var alpha = ["a", "b", "c"];
```

```
 var numeric = [1, 2, 3];
```

```
 var alphaNumeric = alpha.concat(numeric);
```

```
 document.write("alphaNumeric : " + alphaNumeric);
```

```
</script>
```

```
</body>
```

alphaNumeric : a,b,c,1,2,3

```
</html>
```

# JOIN ()

```
<html>
```

```
<head>
```

```
<title>JavaScript Array join Method</title>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
var arr = new Array("First","Second","Third");
```

```
var str = arr.join();
```

```
document.write("str : " + str);
```

```
var str = arr.join(", ");
```

```
document.write("
str : " + str);
```

```
var str = arr.join(" + ");
```

```
document.write("
str : " + str);
```

```
</script>
```

```
</body>
```

str : First,Second,Third

str : First, Second,Third

str : First + Second + Third

## Deleting Elements

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0]; // Changes the first element in fruits
to undefined
```

Using **delete** on array elements leaves undefined holes in the array. Use `pop()` or `splice()` instead.



## POPPING AND PUSHING

- The **pop()** method removes the last element from an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop(); // Removes the last element ("Mango") from fruits
```

- The **push()** method adds a new element to an array (at the end):

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi"); // Adds a new element ("Kiwi") to fruits
```

- The **pop()** method returns the string that was "popped out".
- The **push()** method returns the new array length.

## POP()

```
<html>
<head>
<title>JavaScript Array pop Method</title>
</head>
<body>
<script type="text/javascript">
var numbers = [1, 4, 9];
var element = numbers.pop();
document.write("element is : " + element);
var element = numbers.pop();
document.write("
element is : " + element);
</script>
</body>
</html>
```

Output  
element is : 9  
element is : 4

## PUSH()

```
<html>
```

```
<head>
```

```
<title>JavaScript Array push Method</title>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
var numbers = new Array(1, 4, 9);
```

```
var length = numbers.push(10);
```

```
document.write("new numbers is : " + numbers);
```

```
length = numbers.push(20);
```

```
document.write("
new numbers is : " + numbers);
```

```
</script>
```

**Output**

new numbers is : 1,4,9,10

new numbers is : 1,4,9,10,20

## SHIFTING ELEMENTS

- Shifting is equivalent to popping, working on the first element instead of the last.
- The **shift()** method removes the first element of an array, and "shifts" all other elements one place down.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift(); // Removes the first element "Banana" from fruits
```

Output: Orange,Apple,Mango

- The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon"); // Adds a new element "Lemon" to fruits
```

Output: Lemon,Banana,Orange,Apple,Mango

# SHIFT()

```
<html>
<head>
<title>JavaScript Array shift Method</title>
</head>
<body>
<script type="text/javascript">
var element = [105, 1, 2, 3].shift();
document.write("Removed element is : " + element);
</script>
</body>
```

**Output**

**Removed element is : 105**

## Changing Elements

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi"; // Changes the first element of fruits to "Kiwi"
```

## Append Elements

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi"; // Appends "Kiwi" to fruit
```

## SORTING AN ARRAY

The **sort()** method sorts an array alphabetically:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort(); // Sorts the elements of fruits
```

Output: Apple, Banana, Mango, Orange

### Reversing an Array

The **reverse()** method reverses the elements in an array.

You can use it to sort an array in descending order:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort(); // Sorts the elements of fruits
fruits.reverse(); // Reverses the order of the elements
```

Output: Orange, Mango, Banana, Apple

## REVERSE ()

```
<html>
<head>
<title>JavaScript Array reverse Method</title>
</head>
<body>
<script type="text/javascript">
var arr = [0, 1, 2, 3].reverse();
document.write("Reversed array is : " + arr);
</script>
</body>
</html>
```

Output  
Reversed array is : 3,2,1,0



## SORT

```
<html>
<head>
<title>JavaScript Array sort Method</title>
</head>
<body>
<script type="text/javascript">
var arr = new Array("orange", "mango", "banana", "sugar");
var sorted = arr.sort();
document.write("Returned string is : " + sorted);
</script>
</body>
</html>
```

Output

Returned string is :

banana,mango,orange,sugar

# SLICE

- slice ():
- Javascript array slice() method extracts a section of an array and returns a new array.
- Syntax Its syntax is as follows:
  - `array.slice( begin [,end] );`

## SLICE ()

```
<html>
<head>
<title>JavaScript Array slice Method</title>
</head>
<body>
<script type="text/javascript">
var arr = ["orange", "mango", "banana", "sugar", "tea"];
document.write("arr.slice(1, 2) : " + arr.slice(1, 2));
document.write("
arr.slice(1, 3) : " + arr.slice(1, 3));
</script>
</body>
</html>
```

### Output

arr.slice( 1, 2) : mango

arr.slice( 1, 3) : mango,banana

## SPLICING AN ARRAY

The **splice()** method can be used to **add** new items to an array:

```
array.splice(index, howMany, [element1][, ..., elementN]);
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

- The first parameter (2) defines the position **where** new elements should be **added** (spliced in).
- The second parameter (0) defines **how many** elements should be **removed**.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.
- Output: Banana,Orange,Lemon,Kiwi,Apple,Mango

## USING SPLICE() TO REMOVE ELEMENTS

- With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1); // Removes the first element of fruits
```

- The first parameter (0) defines the position where new elements should be **added** (spliced in).
- The second parameter (1) defines **how many** elements should be **removed**.
- The rest of the parameters are omitted. No new elements will be added.
- Output: Orange,Apple,Mango

# SPLICE()

<html>

<head>

<title>JavaScript Array splice Method</title>

</head>

<body>

<script type="text/javascript">

var arr = ["orange", "mango", "banana", "sugar", "tea"];

var removed = arr.splice(2, 0, "water");

document.write("After adding : " + arr );

document.write("<br />removed is: " + removed);

removed = arr.splice(3, 1);

document.write("<br />After adding: " + arr );

document.write("<br />removed is: " + removed);

</script>

</body>

After adding :

orange,mango,water,banana,sugar,tea

removed is:

After adding : orange,mango,water,sugar,tea

removed is: banana

# JAVASCRIPT ARRAY METHODS

- `valueOf()` and `toString()` methods to convert Arrays to Strings

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML =
fruits.valueOf();
```

Output:Banana,Orange,Apple,Mango

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML =
fruits.toString();
```

For JavaScript arrays, `valueOf()` and `toString()` are equal.

# INDEX

- Intro to script
- Types
- JavaScript identifiers
- Operators
- Control & Looping structure
- Functions
- Objects
- Arrays
- **Date**
- Math
- String
- Boolean
- DOM
- Validations On Forms



# JAVASCRIPT DATE OBJECT

- The Date object is a datatype built into the JavaScript language.
- The Date object is used to work with dates and times.
- Date objects are created with the **new Date()** constructor.
- There are four ways of initiating a date:
  1. `new Date()` // current date and time
  2. `new Date(milliseconds)` // milliseconds since 1970/01/01
  3. `new Date(dateString)`
  4. `new Date(year, month, day, hours, minutes, seconds, milliseconds)`
- Most parameters above are optional. Not specifying, causes 0 to be passed in.
- Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object, using either local time or UTC (universal, or GMT) time.

# DATE OBJECT PROPERTIES

## Date Properties

---

Here is a list of the properties of the Date object along with their description.

Property	Description
constructor	Specifies the function that creates an object's prototype.
prototype	The prototype property allows you to add properties and methods to an object.

# DATE METHODS

Method	Description
Date()	Returns today's date and time
getDate()	Returns the day of the month for the specified date according to local time.
getDay()	Returns the day of the week for the specified date according to local time.
getFullYear()	Returns the year of the specified date according to local time.
getHours()	Returns the hour in the specified date according to local time.
getMilliseconds()	Returns the milliseconds in the specified date according to local time.
getMinutes()	Returns the minutes in the specified date according to local time.
getMonth()	Returns the month in the specified date according to local time.
getSeconds()	Returns the seconds in the specified date according to local time.
getTime()	Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970,

getTimezoneOffset()	Returns the time-zone offset in minutes for the current locale.
getUTCDate()	Returns the day (date) of the month in the specified date according to universal time.
getUTCDay()	Returns the day of the week in the specified date according to universal time.
getUTCFullYear()	Returns the year in the specified date according to universal time.
getUTCHours()	Returns the hours in the specified date according to universal time.
getUTCMilliseconds()	Returns the milliseconds in the specified date
	according to universal time.
getUTCMinutes()	Returns the minutes in the specified date according to universal time.
getUTCMonth()	Returns the month in the specified date according to universal time.
getUTCSeconds()	Returns the seconds in the specified date according to universal time.
getYear()	<b>Deprecated</b> - Returns the year in the specified date according to local time. Use getFullYear instead.
setDate()	Sets the day of the month for a specified date according to local time.
setFullYear()	Sets the full year for a specified date according to local time.
setHours()	Sets the hours for a specified date according to local time.
setMilliseconds()	Sets the milliseconds for a specified date according to local time.
setMinutes()	Sets the minutes for a specified date according to local time.
setMonth()	Sets the month for a specified date according to

setSeconds()	Sets the seconds for a specified date according to local time.
setTime()	Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
setUTCDate()	Sets the day of the month for a specified date according to universal time.
setUTCFullYear()	Sets the full year for a specified date according to universal time.
setUTCHours()	Sets the hour for a specified date according to universal time.
setUTCMilliseconds()	Sets the milliseconds for a specified date according to universal time.
setUTCMinutes()	Sets the minutes for a specified date according to
	universal time.
setUTCMonth()	Sets the month for a specified date according to universal time.
setUTCSeconds()	Sets the seconds for a specified date according to universal time.
setYear()	<b>Deprecated</b> - Sets the year for a specified date according to local time. Use setFullYear instead.
toString()	Returns the "date" portion of the Date as a human-readable string.
toGMTString()	<b>Deprecated</b> - Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
toLocaleDateString()	Returns the "date" portion of the Date as a string, using the current locale's conventions.
toLocaleFormat()	Converts a date to a string, using a format string.

# JAVASCRIPT DATE()

```
<html>
<head>
<title>JavaScript Date Method</title>
</head>
<body>
<script type="text/javascript">
 var dt = Date();
 document.write("Date and Time : " + dt);
</script>
</body>
</html>
```

## Output

Date and Time :Wed Sep 20 2017 20:46:12  
GMT+0530 (India Standard Time)

```
<html>
<head>
<title>JavaScript getDate Method</title>
</head>
<body>
<script type="text/javascript">
 var dt = new Date("December 25, 1995 23:15:00");
 document.write("getDate() : " + dt.getDate());
document.write("getDay() : " + dt.getDay());
document.write("getFullYear() : " + dt.getFullYear());
document.write("getMonth() : " + dt.getMonth());
document.write("getHours() : " + dt.getHours());
document.write("getMinutes() : " + dt.getMinutes());
document.write("getSeconds() : " + dt.getSeconds());
document.write("getTime() : " + dt.getTime())
 document.write("getUTCDate() : " + dt.getUTCDate());

</script>
</body>
</html>
```

## Output

Date and Time :Wed Mar 25 2015 15:00:00 GMT+0530 (India Standard Time)  
getDate() : 25  
getDay() : 1  
getFullYear() : 1995  
getMonth() : 1  
getHours() : 23  
getMinutes() : 15  
getSeconds () : 20  
getTime() : 819913520000

Current Time: **<span id="txt"></span>**

**<script>**

var **today**=**new** Date();

var **h**=**today**.getHours();

var **m**=**today**.getMinutes();

var **s**=**today**.getSeconds();

document.getElementById('txt').**innerHTML**=**h**+":"+**m**+":"+**s**;

**</script>**



# INDEX

- Intro to script
- Types
- JavaScript identifiers
- Operators
- Control & Looping structure
- Functions
- Objects
- Arrays
- Date
- **Math**
- String
- DOM
- Event handling
- Validations On Forms

# JAVASCRIPT MATH OBJECT

- The math object provides you properties and methods for mathematical constants and functions.
- Unlike other global objects, Math is not a constructor.
- All the properties and methods of Math are static and can be called by using Math as an object without creating it.
- **Syntax**

`var x = Math.PI; // Returns PI`

`var y = Math.sqrt(16); // Returns the square root of 16`

# MATH PROPERTIES

Property	Description
E	Euler's constant and the base of natural logarithms, approximately 2.718.
LN2	Natural logarithm of 2, approximately 0.693.
LN10	Natural logarithm of 10, approximately 2.302.
LOG2E	Base 2 logarithm of E, approximately 1.442.
LOG10E	Base 10 logarithm of E, approximately 0.434.
PI	Ratio of the circumference of a circle to its diameter, approximately 3.14159.
SQRT1_2	Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
SQRT2	Square root of 2, approximately 1.414.

```
<html>
<head>
<title>JavaScript Math E Property</title>
</head>
<body>
<script type="text/javascript">
 var property_value = Math.E
 document.write("
 Euler's constantis :" + property_value);
var property_value = Math.LN10
 document.write("
 natural logarithm of 10 is :" + property_value);
var property_value = Math.LOG2E
 document.write("
 base 2 logarithm of E is :" + property_value);
var property_value = Math.PI
 document.write("
 Value of Pi is :" + property_value);
var property_value = Math.SQRT1_2
 document.write("
 square root of 1/2 is :" + property_value);
var property_value = Math.SQRT2
 document.write("
 square root of 2 is :" + property_value);
</script>
</body>
</html>
```

Euler's constantis :2.718281828459045  
natural logarithm of 10 is : 2.302585092994  
base 2 logarithm of E is : 1.44269504088896  
Value of Pi is : 3.141592653589793  
square root of 1/2 is : 0.7071067811865476  
square root of 2 is : 1.4142135623730951

Method	Description
<code>abs(x)</code>	Returns the absolute value of x
<code>acos(x)</code>	Returns the arccosine of x, in radians
<code>asin(x)</code>	Returns the arcsine of x, in radians
<code>atan(x)</code>	Returns the arctangent of x as a numeric value between $-\pi/2$ and $\pi/2$ radians
<code>atan2(y, x)</code>	Returns the arctangent of the quotient of its arguments
<code>ceil(x)</code>	Returns the value of x rounded up to its nearest integer
<code>cos(x)</code>	Returns the cosine of x (x is in radians)
<code>exp(x)</code>	Returns the value of $E^x$
<code>floor(x)</code>	Returns the value of x rounded down to its nearest integer
<code>log(x)</code>	Returns the natural logarithm (base E) of x
<code>max(x, y, z, ..., n)</code>	Returns the number with the highest value
<code>min(x, y, z, ..., n)</code>	Returns the number with the lowest value
<code>pow(x, y)</code>	Returns the value of x to the power of y
<code>random()</code>	Returns a random number between 0 and 1
<code>round(x)</code>	Returns the value of x rounded to its nearest integer
<code>sin(x)</code>	Returns the sine of x (x is in radians)

```
<html> <head> <title>JavaScript Math abs() Method</title> </head> <body>
```

```
<script type="text/javascript">
```

```
var value = Math.abs(-1);
```

```
document.write("
 Absolute Value : " + value);
```

```
var value = Math.ceil(45.95);
```

```
document.write("
 ceil Value : " + value);
```

```
var value = Math.exp(1);
```

```
document.write("
 exponential Value : " + value);
```

```
var value = Math.floor(10.3);
```

```
document.write("
 floor Value : " + value);
```

```
var value = Math.log(0);
```

```
document.write("
log Value : " + value);
```

```
var value = Math.max(10, 20, -1, 100);
```

```
document.write("
 max Value : " + value);
```

```
var value = Math.min(10, 20, -1, 100);
```

```
document.write("
 min Value : " + value);
```

```
var value = Math.pow(7, 2);
```

```
document.write("
 power Value : " + value);
```

Absolute Value : 1

ceil Value : 46

exponential Value : 2.718281828459045

floor Value : 10

log Value : -Infinity

max Value : 100

min Value : -1

power Value : 49

random Value : 0.8856235740915916

round Value : 7

square root Value : 9

# INDEX

- Intro to script
- Types
- JavaScript identifiers
- Operators
- Control & Looping structure
- Functions
- Objects
- Arrays
- Date
- Math
- **String**
- Boolean
- DOM
- Validations On Forms

# JAVASCRIPT STRINGS

- JavaScript strings are used for storing and manipulating text.
- A JavaScript string simply stores a series of characters like "John Doe".
- A string can be any text inside quotes. You can use single or double quotes:
- You can use quotes inside a string, as long as they don't match the quotes surrounding the string

```
var answer = "It's alright";
```

```
var answer = "He is called 'Johnny' ";
```

```
var answer = 'He is called "Johnny" ';
```

```
var answer = "He is called "Johnny""; (Won't work)
```

Solution: 

```
var answer = "He is called \"Johnny\"";
```

- The backslash escape character turns special characters into string characters



## STRING PROPERTIES

Property	Description
constructor	Returns a reference to the String function that created the object.
length	Returns the length of the string.
prototype	The prototype property allows you to add properties and methods to an object.

```
<html>

<head>

<title>JavaScript String prototype property</title>

<script type="text/javascript">

function book(title, author){

 this.title = title;

 this.author = author;

}

</head>

<body>

<script type="text/javascript">

 var str = new String("This is string");

 document.write("str.constructor is:" + str.constructor);

 document.write("
str.length is:" + str.length);

 var myBook = new book("Perl", "Mohtashim");

 book.prototype.price = null;

 myBook.price = 100;

 document.write("Book title is : " + myBook.title + "
");

 document.write("Book author is : " + myBook.author + "
");

 document.write("Book price is : " + myBook.price + "
");
```

# STRING METHODS

Method	Description
<code>charAt()</code>	Returns the character at the specified index.
<code>charCodeAt()</code>	Returns a number indicating the Unicode value of the character at the given index.
<code>concat()</code>	Combines the text of two strings and returns a new string.
<code>indexOf()</code>	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
<code>lastIndexOf()</code>	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
<code>localeCompare()</code>	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sorted order.
<code>match()</code>	Used to match a regular expression against a string.
<code>replace()</code>	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
<code>search()</code>	Executes the search for a match between a regular expression and a specified string.
<code>slice()</code>	Extracts a section of a string and returns a new string.

<code>split()</code>	Splits a String object into an array of strings by separating the string into substrings.
<code>substr()</code>	Returns the characters in a string beginning at the specified location through the specified number of characters.
<code>substring()</code>	Returns the characters in a string between two indexes into the string.
<code>toLocaleLowerCase()</code>	The characters within a string are converted to lower case while respecting the current locale.
<code>toLocaleUpperCase()</code>	The characters within a string are converted to upper case while respecting the current locale.
<code>toLowerCase()</code>	Returns the calling string value converted to lower case.
<code>toString()</code>	Returns a string representing the specified object.
<code>toUpperCase()</code>	Returns the calling string value converted to uppercase.
<code>valueOf()</code>	Returns the primitive value of the specified object.

```
<body>
```

```
<script type="text/javascript">
```

```
 var str1 = new String("This is first string");
```

```
 document.write("
 str.charAt(0) is:" + str1.charAt(0));
```

```
document.write("
 str.charCodeAt(0) is:" + str1.charCodeAt(0));
```

```
var str2 = new String("This is second string");
```

```
var str3 = str1.concat(str2);
```

```
 document.write("
 str3 is:" + str3);
```

```
var index = str1.indexOf("string");
```

```
document.write("
 indexOf String : " + index);
```

```
document.write("
");
```

```
document.write(str1.toLowerCase());
```

```
document.write("
");
```

```
document.write(str1.toUpperCase());
```

```
document.write("
");
```

```
</script> </body>
```

str.charAt(0) is:T

str.charCodeAt(0) is:84

str3 is:This is first stringThis is second string

indexOf String :14

this is first string

THIS IS FIRST STRING

# JAVASCRIPT STRINGS CONT...

- Breaking Long Code Lines

```
document.getElementById("demo").innerHTML =
 "Hello Dolly.";
```

```
document.getElementById("demo").innerHTML = "Hello \
Dolly!";
```

```
document.getElementById("demo").innerHTML = "Hello" +
 "Dolly!";
```

```
document.getElementById("demo").innerHTML = \
 "Hello Dolly!"; (Won't work)
```

## STRINGS CAN BE OBJECTS

```
var x = "John"; // typeof x will return string
var y = new String("John"); // typeof y will return object
```

== (check the equality of values)

=== (check the equality of values and type)

(x == y) is true because x and y have equal values

(x === y) is false because x and y have different types

```
var x = new String("John");
var y = new String("John");
```

```
// (x == y) is false because objects cannot be compared
```

# CREATING USER DEFINED OBJECT

```
<html>
```

```
<head>
```

```
<title>User-defined objects</title>
```

```
<script type="text/javascript">
```

```
var book = new Object(); // Create the object
```

```
book.subject = "OS"; // Assign properties to the object
```

```
book.author = "Stalling";
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
document.write("Book name is : " + book.subject + "
");
```

```
document.write("Book author is : " + book.author + "
");
```

```
</script>
```



```
<html>
 <head>
 <title>User-defined objects</title>
 <script type="text/javascript">
 function book(title, author){
 this.title = title;
 this.author = author;
 }
 </script>
 </head>
 <body>
 <script type="text/javascript">
 var myBook = new book("OS", "Stalling");
 document.write("Book title is : " + myBook.title + "
");
 document.write("Book author is : " + myBook.author + "
");
 </script>
 </body>
</html>
```

```
<html>

 <head>

 <title>User-defined objects</title>

 <script type="text/javascript">

 function book(title, author){

 this.title = title;

 this.author = author;

 }

function disp(){

 document.write("Book title is : " + myBook.title + "
");

 document.write("Book author is : " + myBook.author + "
");

}

 </script>

 </head>

 <body>

 <script type="text/javascript">

 var myBook = new book("OS", "Stalling");

 disp();

 </script>
```

```
<html> <head> <title>User-defined objects</title>
```

```
 <script type="text/javascript">
```

```
 // Define a function which will work as a method
```

```
 function addPrice(amount){
```

```
 this.price = amount;
```

```
 }
```

```
 function book(title, author){
```

```
 this.title = title;
```

```
 this.author = author;
```

```
 this.addPrice = addPrice; // Assign that method as property.
```

```
 }
```

```
 </script> </head>
```

```
<body>
```

```
 <script type="text/javascript">
```

```
 var myBook = new book("OS", "Stalling");
```

```
 myBook.addPrice(100);
```

```
 document.write("Book title is : " + myBook.title + "
");
```

```
 document.write("Book author is : " + myBook.author + "
");
```

```
 document.write("Book price is : " + myBook.price + "
");
```

# INPUT FROM USER:PROMPT

```
<!DOCTYPE html>

<html>

<body>

<p>Click the button to display a dialog box which will
ask for your favorite car.</p>

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

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

<script>

function myFunction() {

 var text;

 var car = prompt("What's your favorite car?");

 switch(car) {
```

```
 switch(car) {

 case "BMW":

 text = "Excellent choice."; break;

 case "Honda":

 text = "very good!"; break;

 case "Maruti":

 text = "Good";

 break;

 default:

 text = "I have never heard of that one..";

 }

 document.getElementById("demo").innerHTML
 = text;

 }

</script>

</body>

</html>
```

# INDEX

- Intro to script
- Types
- JavaScript identifiers
- Operators
- Control & Looping structure
- Functions
- Objects
- Arrays
- Date
- Math
- String
- **Boolean**
- DOM
- Validations On Forms

## BOOLEAN OBJECT

- The **Boolean** object represents two values, either "true" or "false".
- If *value* parameter is omitted or is 0, -0, null, false, **NaN**, undefined, or the empty string (""), the object has an initial value of false.

# BOOLEAN OBJECT

<html>

<head>

<title>JavaScript String Methods</title>

</head>

<body>

<script type="text/javascript">

var b1=new Boolean(1);

var b2=new Boolean(0);

var b3= new Boolean(null);

var b4= new Boolean(10>4);

var b5= new Boolean("vit");

var b6= new Boolean(false);

var b7= new Boolean(-0);

var b8= new Boolean("");

document.write(b1+"<br>");

document.write(b2+"<br>");

document.write(b3+"<br>");

document.write(b4+"<br>");

document.write(b5+"<br>");

document.write(b6+"<br>");

document.write(b7+"<br>");

document.write(b8+"<br>");

</script>

</body>

</html>

true

false

false

true

true

false

false

false