About the Tutorial

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

Audience

This tutorial has been prepared for JavaScript beginners to help them understand the basic functionality of JavaScript to build dynamic web pages and web applications.

Prerequisites

For this tutorial, it is assumed that the reader have a prior knowledge of HTML coding. It would help if the reader had some prior exposure to object-oriented programming concepts and a general idea on creating online applications.

Copyright and Disclaimer

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com
# Table of Contents

About the Tutorial .................................................................................................................... i  
Audience .................................................................................................................................... i  
Prerequisites ............................................................................................................................... i  
Copyright and Disclaimer ......................................................................................................... i  
Table of Contents ....................................................................................................................... ii  

**PART 1: JAVASCRIPT BASICS .............................................................. 1**  

1. **Overview .......................................................................................................................... 2**  
   What is JavaScript? .................................................................................................................. 2  
   Client-Side JavaScript .......................................................................................................... 2  
   Advantages of JavaScript ..................................................................................................... 3  
   Limitations of JavaScript ..................................................................................................... 3  
   JavaScript Development Tools ............................................................................................ 3  
   Where is JavaScript Today? .................................................................................................. 4  

2. **Syntax ............................................................................................................................. 5**  
   Your First JavaScript Code ................................................................................................... 5  
   Whitespace and Line Breaks ............................................................................................... 6  
   Semicolons are Optional ..................................................................................................... 6  
   Case Sensitivity .................................................................................................................. 7  
   Comments in JavaScript ..................................................................................................... 7  

3. **Enabling ........................................................................................................................... 9**  
   JavaScript in Internet Explorer .......................................................................................... 9  
   JavaScript in Firefox ........................................................................................................... 9  
   JavaScript in Chrome .......................................................................................................... 10  
   JavaScript in Opera ............................................................................................................ 10  
   Warning for Non-JavaScript Browsers .............................................................................. 10  

4. **Placement ......................................................................................................................... 12**  
   JavaScript in <head>...</head> Section ............................................................................... 12  
   JavaScript in <body>...</body> Section .............................................................................. 13  
   JavaScript in <body> and <head> Sections ......................................................................... 13  
   JavaScript in External File ................................................................................................. 14  

5. **Variables ......................................................................................................................... 16**  
   JavaScript Datatypes .......................................................................................................... 16  
   JavaScript Variables .......................................................................................................... 16  
   JavaScript Variable Scope .................................................................................................. 17  
   JavaScript Variable Names ................................................................................................ 18  
   JavaScript Reserved Words ............................................................................................... 19  

6. **Operators ......................................................................................................................... 20**  
   What is an Operator? ........................................................................................................... 20  
   Arithmetic Operators ........................................................................................................ 20  
   Comparison Operators ....................................................................................................... 23  
   Logical Operators .............................................................................................................. 26  

![tutorialspoint](https://example.com/logo.png)
## Arrays

<table>
<thead>
<tr>
<th>Array Properties</th>
<th>177</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>178</td>
</tr>
<tr>
<td>length</td>
<td>179</td>
</tr>
<tr>
<td>Prototype</td>
<td>180</td>
</tr>
<tr>
<td>Array Methods</td>
<td>181</td>
</tr>
<tr>
<td>concat ()</td>
<td>183</td>
</tr>
<tr>
<td>every ()</td>
<td>184</td>
</tr>
<tr>
<td>filter ()</td>
<td>186</td>
</tr>
<tr>
<td>forEach ()</td>
<td>189</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String Properties</th>
<th>136</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>136</td>
</tr>
<tr>
<td>length</td>
<td>137</td>
</tr>
<tr>
<td>Prototype</td>
<td>138</td>
</tr>
<tr>
<td>String Methods</td>
<td>139</td>
</tr>
<tr>
<td>charAt()</td>
<td>141</td>
</tr>
<tr>
<td>charCodeAt ()</td>
<td>142</td>
</tr>
<tr>
<td>contact ()</td>
<td>143</td>
</tr>
<tr>
<td>indexOf ()</td>
<td>144</td>
</tr>
<tr>
<td>lastIndexOf ()</td>
<td>146</td>
</tr>
<tr>
<td>localeCompare ()</td>
<td>147</td>
</tr>
<tr>
<td>match ()</td>
<td>148</td>
</tr>
<tr>
<td>replace ()</td>
<td>149</td>
</tr>
<tr>
<td>search ()</td>
<td>152</td>
</tr>
<tr>
<td>slice ()</td>
<td>153</td>
</tr>
<tr>
<td>split ()</td>
<td>154</td>
</tr>
<tr>
<td>substr ()</td>
<td>155</td>
</tr>
<tr>
<td>substring ()</td>
<td>156</td>
</tr>
<tr>
<td>toLocaleLowerCase ()</td>
<td>157</td>
</tr>
<tr>
<td>toLocaleUpperCase ()</td>
<td>158</td>
</tr>
<tr>
<td>toLowerCase ()</td>
<td>159</td>
</tr>
<tr>
<td>toString ()</td>
<td>160</td>
</tr>
<tr>
<td>toUpperCase ()</td>
<td>161</td>
</tr>
<tr>
<td>valueOf ()</td>
<td>162</td>
</tr>
<tr>
<td>String HTML Wrappers</td>
<td>163</td>
</tr>
<tr>
<td>anchor ()</td>
<td>164</td>
</tr>
<tr>
<td>big ()</td>
<td>165</td>
</tr>
<tr>
<td>blink ()</td>
<td>166</td>
</tr>
<tr>
<td>bold ()</td>
<td>166</td>
</tr>
<tr>
<td>fixed ()</td>
<td>167</td>
</tr>
<tr>
<td>fontColor ()</td>
<td>168</td>
</tr>
<tr>
<td>fontSize ()</td>
<td>169</td>
</tr>
<tr>
<td>italics ()</td>
<td>170</td>
</tr>
<tr>
<td>link ()</td>
<td>171</td>
</tr>
<tr>
<td>small ()</td>
<td>172</td>
</tr>
<tr>
<td>strike ()</td>
<td>173</td>
</tr>
<tr>
<td>sub()</td>
<td>174</td>
</tr>
<tr>
<td>sup ()</td>
<td>175</td>
</tr>
</tbody>
</table>
25. Date.............................................................................................................220

  Date Properties..........................................................................................221
  constructor.................................................................................................221
  Prototype......................................................................................................222
  Date Methods................................................................................................224
  Date()...........................................................................................................224
  getDate().......................................................................................................227
  getDay().........................................................................................................228
  getFullYear()..............................................................................................229
  getHours()......................................................................................................230
  getMilliseconds().........................................................................................231
  getMinutes().................................................................................................232
  getMonth().....................................................................................................233
  getSeconds()..................................................................................................234
  getTime().........................................................................................................234
  getTimezoneOffset()......................................................................................235
  getUTCDate().................................................................................................236
  getUTCDay()..................................................................................................237
  getUTCFullYear()...........................................................................................238
  getUTCHours()...............................................................................................239
  getUTCMilliseconds()....................................................................................240
  getUTCMilliseconds()....................................................................................240
  getUTCMilliseconds()....................................................................................241
  getUTCMilliseconds()....................................................................................241
  getUTCMonth()...............................................................................................242
  getUTCMilliseconds()....................................................................................242
  getUTCMilliseconds()....................................................................................243
  setDate().........................................................................................................243
  setFullYear()...............................................................................................244
  setHours().......................................................................................................245
  setMilliseconds().........................................................................................246
  setMinutes()..................................................................................................247
  setMonth().....................................................................................................248
  setMilliseconds().........................................................................................249
  setSeconds()..................................................................................................250
  setTime().........................................................................................................252
setUTCDate () ................................................................. 252
setUTCFullYear () ............................................................ 253
setUTCHours () ............................................................... 255
setUTCMilliseconds () ...................................................... 256
setUTCMinutes () ............................................................. 257
setUTC Month () ............................................................. 258
setUTCSeconds () .............................................................. 259
setYear () .......................................................... 260
toDateString () .............................................................. 261
toGMTString () ............................................................... 262
toLocaleDateString () ........................................................ 263
toLocaleDateString () ........................................................ 264
toLocaleFormat () ............................................................ 264
toLocaleString () .............................................................. 265
toLocaleTimeString () ....................................................... 266
toSource () ..................................................................... 267
toString () ...................................................................... 268
toTimeString () .................................................................. 269
toUTCString () ................................................................... 270
valueOf () .......................................................... 271
Date Static Methods ......................................................... 272
Date.parse () ............................................................... 272
Date.UTC () ............................................................. 273

26. Math ........................................................................ 275
   Math Properties ............................................................. 275
   Math-E ........................................................................... 276
   Math-LN2 ................................................................. 277
   Math-LN10 ............................................................... 277
   Math-LOG2E ............................................................ 278
   Math-LOG10E ............................................................ 279
   Math-PI ....................................................................... 280
   Math-SQRT1_2 ........................................................... 280
   Math-SQRT2 .............................................................. 281
   Math Methods ............................................................. 282
   abs () .......................................................................... 283
   acos () ......................................................................... 284
   asin () .......................................................................... 286
   atan () ......................................................................... 287
   atan2 () ....................................................................... 288
   ceil () .......................................................................... 290
   cos () ............................................................................ 291
   exp () ............................................................................ 292
   floor () .......................................................................... 294
   log () ............................................................................ 295
   max () ............................................................................ 296
   min () ............................................................................ 298
   pow () ............................................................................ 299
   random () ...................................................................... 300
   round () ....................................................................... 301
### 27. RegExp

- Brackets .................................................................................................................. 308
- Quantifiers ................................................................................................................ 309
- Literal Characters ..................................................................................................... 310
- Metacharacters .......................................................................................................... 311
- Modifiers .................................................................................................................... 311
- RegExp Properties ..................................................................................................... 312
- constructor ................................................................................................................. 312
- global ......................................................................................................................... 313
- ignoreCase .................................................................................................................. 314
- lastIndex .................................................................................................................... 316
- multiline ..................................................................................................................... 317
- source ......................................................................................................................... 318
- RegExp Methods ......................................................................................................... 319
- exec () ....................................................................................................................... 320
- test () ......................................................................................................................... 321
- toSource () ............................................................................................................... 322
- toString () ................................................................................................................ 323

### 28. DOM

- The Legacy DOM .................................................................................................... 325
- The W3C DOM .......................................................................................................... 326
- The IE 4 DOM ............................................................................................................ 332
- DOM Compatibility .................................................................................................. 336
- DOM Compatibili .................................................................................................. 340

### 29. Errors and Exceptions

- Syntax Errors ........................................................................................................... 343
- Runtime Errors ........................................................................................................ 343
- Logical Errors .......................................................................................................... 344
- The try...catch...finally Statement .......................................................................... 344
- The throw Statement ............................................................................................... 348
- The onerror( ) Method ......................................................................................... 349

### 30. Form Validation

- Basic Form Validation ............................................................................................. 352
- Data Format Validation .......................................................................................... 354

### 31. Animation

- Manual Animation .................................................................................................. 357
- Automated Animation ............................................................................................ 358
- Rollover with a Mouse Event .................................................................................. 359

### 32. Multimedia

- Multimedia Rollover with a Mouse Event ................................................................ 363
Part 1: JavaScript Basics
What is JavaScript?

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

- 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.

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.
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.

- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.

- **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

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

- 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.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here:

- **Microsoft FrontPage:** Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.

- **Macromedia Dreamweaver MX:** Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript
components, integrates well with databases, and conforms to new standards such as XHTML and XML.

- **Macromedia HomeSite 5**: HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

Where is JavaScript Today?

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.

The specification for JavaScript 2.0 can be found on the following site: [http://www.ecmascript.org/](http://www.ecmascript.org/)

Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both the languages still support the features that are not a part of the standard.
JavaScript can be implemented using JavaScript statements that are placed within the `<script>...</script>` HTML tags in a web page.

You can place the `<script>` tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the `<head>` tags.

The `<script>` tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```html
<script ...
    JavaScript code
</script>
```

The script tag takes two important attributes:

- **Language**: This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.

- **Type**: This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript syntax will look as follows.

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

### Your First JavaScript Code

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function `document.write` which writes a string into our HTML document.
This function can be used to write text, HTML, or both. Take a look at the following code.

```html
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
    document.write ("Hello World!")
//-->
</script>
</body>
</html>
```

This code will produce the following result:

Hello World!

**Whitespace and Line Breaks**

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

**Semicolons are Optional**

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```javascript
<script language="javascript" type="text/javascript">
<!--
    var1 = 10
    var2 = 20
//-->
</script>
```
But when formatted in a single line as follows, you must use semicolons:

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

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

---

**Case Sensitivity**

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

**NOTE:** Care should be taken while writing variable and function names in JavaScript.

---

**Comments in JavaScript**

JavaScript supports both C-style and C++-style comments. Thus:

- Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.

- Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.

- JavaScript also recognizes the HTML comment opening sequence `<!--`. JavaScript treats this as a single-line comment, just as it does the `//` comment.

- The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.

---

**Example**
The following example shows how to use comments in JavaScript.

```html
<script language="javascript" type="text/javascript">
<!--
// This is a comment. It is similar to comments in C++

/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
//-->
</script>
```
All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, Chrome, and Opera.

### JavaScript in Internet Explorer

Here are the steps to turn on or turn off JavaScript in Internet Explorer:

- Follow **Tools -> Internet Options** from the menu.
- Select **Security** tab from the dialog box.
- Click the **Custom Level** button.
- Scroll down till you find the **Scripting** option.
- Select **Enable** radio button under **Active scripting**.
- Finally click OK and come out.

To disable JavaScript support in your Internet Explorer, you need to select **Disable** radio button under **Active scripting**.

### JavaScript in Firefox

Here are the steps to turn on or turn off JavaScript in Firefox:

- Open a new tab -> type **about: config** in the address bar.
- Then you will find the warning dialog. Select **I’ll be careful, I promise!**
- Then you will find the list of **configure options** in the browser.
- In the search bar, type **javascript.enabled**.
- There you will find the option to enable or disable javascript by right-clicking on the value of that option -> **select toggle**.

If javascript.enabled is true; it converts to false upon clicking toggle. If javascript is disabled; it gets enabled upon clicking toggle.
JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome:

- Click the Chrome menu at the top right hand corner of your browser.
- Select Settings.
- Click Show advanced settings at the end of the page.
- Under the Privacy section, click the Content settings button.
- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

JavaScript in Opera

Here are the steps to turn on or turn off JavaScript in Opera:

- Follow Tools-> Preferences from the menu.
- Select Advanced option from the dialog box.
- Select Content from the listed items.
- Select Enable JavaScript checkbox.
- Finally click OK and come out.

To disable JavaScript support in Opera, you should not select theEnable JavaScript checkbox.

Warning for Non-JavaScript Browsers

If you have to do something important using JavaScript, then you can display a warning message to the user using <noscript> tags.

You can add a noscript block immediately after the script block as follows:

```
<html>
<body>

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

<!--
    document.write ("Hello World!")
//-->  

</script>
```
Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from </noscript> will be displayed on the screen.
There is a flexibility given to include JavaScript code anywhere in an HTML document. However, the most preferred ways to include JavaScript in an HTML file are as follows:

- 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.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

### 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 follows.

```html
<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>
```
This code will produce the following results:

Click here for the result

Say Hello

**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. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

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

This code will produce the following results:

Hello World
This is web page body

**JavaScript in `<body>` and `<head>` Sections**

You can put your JavaScript code in `<head>` and `<body>` section altogether as follows.

```html
<html>
<head>
<script type="text/javascript">
```
function sayHello() {
    alert("Hello World")
}

<script type="text/javascript">
function sayHello() {
    alert("Hello World")
}
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />

This code will produce the following result.

Hello World
Say Hello

JavaScript in External File

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute.
To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in filename.js file and then you can use sayHello function in your HTML file after including the filename.js file.

```javascript
function sayHello() {
    alert("Hello World")
}
```
JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- **Numbers**, 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. In addition to these primitive data types, JavaScript supports a composite data type known as `object`. We will cover objects in detail in a separate chapter.

**Note:** Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the `var` keyword as follows.

```javascript
<script type="text/javascript">
  <!--
  var money;
  var name;
  //-->
</script>
```
You can also declare multiple variables with the same `var` keyword as follows:

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

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named `money` and assign the value `2000.50` to it later. For another variable, you can assign a value at the time of initialization as follows.

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

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

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

### JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.

- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.
Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

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

It will produce the following result:

Local

**JavaScript Variable Names**

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, `break` or `boolean` variable names are not valid.

- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, `123test` is an invalid variable name but `_123test` is a valid one.

- JavaScript variable names are case-sensitive. For example, `Name` and `name` are two different variables.
### JavaScript Reserved Words

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

<table>
<thead>
<tr>
<th>abstract</th>
<th>else</th>
<th>instanceof</th>
<th>switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>enum</td>
<td>int</td>
<td>synchronized</td>
</tr>
<tr>
<td>break</td>
<td>export</td>
<td>interface</td>
<td>this</td>
</tr>
<tr>
<td>byte</td>
<td>extends</td>
<td>long</td>
<td>throw</td>
</tr>
<tr>
<td>case</td>
<td>false</td>
<td>native</td>
<td>throws</td>
</tr>
<tr>
<td>catch</td>
<td>final</td>
<td>new</td>
<td>transient</td>
</tr>
<tr>
<td>char</td>
<td>finally</td>
<td>null</td>
<td>true</td>
</tr>
<tr>
<td>class</td>
<td>float</td>
<td>package</td>
<td>try</td>
</tr>
<tr>
<td>const</td>
<td>for</td>
<td>private</td>
<td>typeof</td>
</tr>
<tr>
<td>continue</td>
<td>function</td>
<td>protected</td>
<td>var</td>
</tr>
<tr>
<td>debugger</td>
<td>goto</td>
<td>public</td>
<td>void</td>
</tr>
<tr>
<td>default</td>
<td>if</td>
<td>return</td>
<td>volatile</td>
</tr>
<tr>
<td>delete</td>
<td>implements</td>
<td>short</td>
<td>while</td>
</tr>
<tr>
<td>do</td>
<td>import</td>
<td>static</td>
<td>with</td>
</tr>
<tr>
<td>double</td>
<td>in</td>
<td>super</td>
<td></td>
</tr>
</tbody>
</table>
What is an Operator?

Let us take a simple expression $4 + 5$ is equal to 9. Here 4 and 5 are called operands and ‘+’ is called the operator. JavaScript supports the following types of operators.

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

Let’s have a look at all the operators one by one.

### Arithmetic Operators

JavaScript supports the following arithmetic operators:

Assume variable A holds 10 and variable B holds 20, then:

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Operator and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+ (Addition)</td>
</tr>
<tr>
<td></td>
<td>Adds two operands</td>
</tr>
<tr>
<td></td>
<td>Ex: A + B will give 30</td>
</tr>
<tr>
<td>2</td>
<td>- (Subtraction)</td>
</tr>
<tr>
<td></td>
<td>Subtracts the second operand from the first</td>
</tr>
<tr>
<td></td>
<td>Ex: A - B will give -10</td>
</tr>
<tr>
<td>3</td>
<td>* (Multiplication)</td>
</tr>
<tr>
<td></td>
<td>Multiply both operands</td>
</tr>
<tr>
<td></td>
<td>Ex: A * B will give 200</td>
</tr>
<tr>
<td>4</td>
<td>/ (Division)</td>
</tr>
<tr>
<td>Arithmetic Operator</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Divide the numerator by the denominator</td>
<td>Ex: B / A will give 2</td>
</tr>
<tr>
<td>5 % (Modulus)</td>
<td>Outputs the remainder of an integer division</td>
</tr>
<tr>
<td>6 ++ (Increment)</td>
<td>Increases an integer value by one</td>
</tr>
<tr>
<td>7 -- (Decrement)</td>
<td>Decreases an integer value by one</td>
</tr>
</tbody>
</table>

**Note:** Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

**Example**

The following code shows how to use arithmetic operators in JavaScript.

```html
<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);

</script>

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

<p>Set the variables to different values and then try...</p>
</body>
</html>

Output

a + b = 43  
a - b = 23  
a / b = 3.3  
a % b = 3  
a + b + c = 43  
a++ = 33  
b-- = 10

Set the variables to different values and then try...

Comparison Operators

JavaScript supports the following comparison operators:

Assume variable A holds 10 and variable B holds 20, then:

<table>
<thead>
<tr>
<th>S.No</th>
<th>Operator and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>== (Equal)</td>
</tr>
<tr>
<td></td>
<td>Checks if the value of two operands are equal or not, if yes, then the condition becomes true.</td>
</tr>
<tr>
<td></td>
<td>Ex: (A == B) is not true.</td>
</tr>
<tr>
<td>2</td>
<td>!= (Not Equal)</td>
</tr>
<tr>
<td></td>
<td>Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.</td>
</tr>
<tr>
<td></td>
<td>Ex: (A != B) is true.</td>
</tr>
<tr>
<td>3</td>
<td>&gt; (Greater than)</td>
</tr>
<tr>
<td></td>
<td>Checks if the value of the left operand is greater than the value of</td>
</tr>
<tr>
<td></td>
<td>Javascript</td>
</tr>
<tr>
<td>---</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>the right operand, if yes, then the condition becomes true.</td>
</tr>
<tr>
<td>Ex</td>
<td>(A &gt; B) is not true.</td>
</tr>
<tr>
<td>4</td>
<td>&lt; (Less than)</td>
</tr>
<tr>
<td></td>
<td>Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.</td>
</tr>
<tr>
<td>Ex</td>
<td>(A &lt; B) is true.</td>
</tr>
<tr>
<td>5</td>
<td>&gt;= (Greater than or Equal to)</td>
</tr>
<tr>
<td></td>
<td>Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.</td>
</tr>
<tr>
<td>Ex</td>
<td>(A &gt;= B) is not true.</td>
</tr>
<tr>
<td>6</td>
<td>&lt;= (Less than or Equal to)</td>
</tr>
<tr>
<td></td>
<td>Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.</td>
</tr>
<tr>
<td>Ex</td>
<td>(A &lt;= B) is true.</td>
</tr>
</tbody>
</table>

**Example**
The following code shows how to use comparison operators in JavaScript.

```html
<html>
<body>

<script type="text/javascript">
  --
  var a = 10;
  var b = 20;
  var linebreak = "<br />";
  
  document.write("(a == b) => ");
  result = (a == b);
  document.write(result);
  document.write(linebreak);
</script>

</body>
</html>
```
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) => ");
result = (a >= b);
document.write(result);
document.write(linebreak);


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

//-->
</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>
</html>
Output

(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true

Set the variables to different values and different operators and then try...

Logical Operators

JavaScript supports the following logical operators:

Assume variable A holds 10 and variable B holds 20, then:

<table>
<thead>
<tr>
<th>S.No</th>
<th>Operator and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&amp;&amp; (Logical AND)</td>
</tr>
<tr>
<td></td>
<td>If both the operands are non-zero, then the condition becomes true.</td>
</tr>
<tr>
<td></td>
<td>Ex: (A &amp;&amp; B) is true.</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If any of the two operands are non-zero, then the condition becomes true.</td>
</tr>
<tr>
<td></td>
<td>Ex: (A</td>
</tr>
<tr>
<td>3</td>
<td>! (Logical NOT)</td>
</tr>
<tr>
<td></td>
<td>Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.</td>
</tr>
<tr>
<td></td>
<td>Ex: ! (A &amp;&amp; B) is false.</td>
</tr>
</tbody>
</table>

Example
Try the following code to learn how to implement Logical Operators in JavaScript.

```html
<html>
<body>
<script type="text/javascript">
<!--
var a = true;
var b = false;
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);

//-->
</script>
<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```
Output

(a && b) => false
(a || b) => true
!(a && b) => true

Set the variables to different values and different operators and then try...

Bitwise Operators

JavaScript supports the following bitwise operators:

Assume variable A holds 2 and variable B holds 3, then:

<table>
<thead>
<tr>
<th>S.No</th>
<th>Operator and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&amp; (Bitwise AND)</td>
</tr>
<tr>
<td></td>
<td>It performs a Boolean AND operation on each bit of its integer arguments.</td>
</tr>
<tr>
<td></td>
<td>Ex: (A &amp; B) is 2.</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>It performs a Boolean OR operation on each bit of its integer arguments.</td>
</tr>
<tr>
<td></td>
<td>Ex: (A</td>
</tr>
<tr>
<td>3</td>
<td>^ (Bitwise XOR)</td>
</tr>
<tr>
<td></td>
<td>It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.</td>
</tr>
<tr>
<td></td>
<td>Ex: (A ^ B) is 1.</td>
</tr>
<tr>
<td>4</td>
<td>~ (Bitwise Not)</td>
</tr>
<tr>
<td></td>
<td>It is a unary operator and operates by reversing all the bits in the operand.</td>
</tr>
</tbody>
</table>
Ex: \((\sim B)\) is -4.

---

**<< (Left Shift)**

It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.

**Ex:** \((A << 1)\) is 4.

---

**>>(Right Shift)**

Binary Right Shift Operator. The left operand’s value is moved right by the number of bits specified by the right operand.

**Ex:** \((A >> 1)\) is 1.

---

**>>> (Right Shift with Zero)**

This operator is just like the >> operator, except that the bits shifted in on the left are always zero.

**Ex:** \((A >>> 1)\) is 1.

---

**Example**

Try the following code to implement Bitwise operator in JavaScript.

```html
<html>
<body>

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

<!--
var a = 2; // Bit presentation 10
var b = 3; // Bit presentation 11
var linebreak = "<br />";

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

</body>
</html>
```
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("(~b) => ");
result = (~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);

//-->
</script>

<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>

Output
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0

Set the variables to different values and different operators and then try...

### Assignment Operators

JavaScript supports the following assignment operators:

<table>
<thead>
<tr>
<th>S.No</th>
<th>Operator and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>= (Simple Assignment)</td>
</tr>
<tr>
<td>2</td>
<td>+= (Add and Assignment)</td>
</tr>
<tr>
<td>3</td>
<td>-= (Subtract and Assignment)</td>
</tr>
<tr>
<td>4</td>
<td>*= (Multiply and Assignment)</td>
</tr>
<tr>
<td>5</td>
<td>/= (Divide and Assignment)</td>
</tr>
<tr>
<td>Ex: C /= A is equivalent to C = C / A</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>%=(Modules and Assignment)</td>
</tr>
<tr>
<td>It takes modulus using two operands and assigns the result to the left operand.</td>
<td></td>
</tr>
<tr>
<td>Ex: C %= A is equivalent to C = C % A</td>
<td></td>
</tr>
</tbody>
</table>

Note: Same logic applies to Bitwise operators, so they will become <<=, >>=, >>=, &=, |= and ^=.

Example
Try the following code to implement assignment operator in JavaScript.

```html
<html>
<body>

<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var linebreak = "<br />";

document.write("Value of a => (a = b) => ");
result = (a = b);
document.write(result);
document.write(linebreak);

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

document.write("Value of a => (a -= b) => ");
result = (a -= b);
document.write(result);
</script>
</body>
</html>
```
document.write(linebreak);

document.write("Value of a => (a *= b) => ");
result = (a *= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a /= b) => ");
result = (a /= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a %= b) => ");
result = (a %= b);
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>

**Output**

Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
Set the variables to different values and different operators and then try...

**Miscellaneous Operators**

We will discuss two operators here that are quite useful in JavaScript: the *conditional operator* (? :) and the *typeof operator*.

**Conditional Operator (? :)**

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Operator and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>? : (Conditional)</td>
</tr>
<tr>
<td></td>
<td>If Condition is true? Then value X : Otherwise value Y</td>
</tr>
</tbody>
</table>

**Example**

Try the following code to understand how the Conditional Operator works in JavaScript.

```html
<html>
<body>

<script type="text/javascript">
<!--
 var a = 10;
 var b = 20;
 var linebreak = "<br />";

document.write("((a > b) ? 100 : 200) => ");
 result = (a > b) ? 100 : 200;
 document.write(result);
 document.write(linebreak);

document.write("((a < b) ? 100 : 200) => ");
</script>
</body>
</html>
```
result = (a < b) ? 100 : 200;
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>

Output

\[
((a > b) \ ? \ 100 \ : \ 200) \Rightarrow 200 \\
((a < b) \ ? \ 100 \ : \ 200) \Rightarrow 100
\]

Set the variables to different values and different operators and then try...

typeof Operator

The `typeof` operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The `typeof` operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the `typeof` Operator.

<table>
<thead>
<tr>
<th>Type</th>
<th>String Returned by typeof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>&quot;number&quot;</td>
</tr>
<tr>
<td>String</td>
<td>&quot;string&quot;</td>
</tr>
<tr>
<td>Boolean</td>
<td>&quot;boolean&quot;</td>
</tr>
<tr>
<td>Object</td>
<td>&quot;object&quot;</td>
</tr>
<tr>
<td>Function</td>
<td>&quot;function&quot;</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>Undefined</td>
<td>&quot;undefined&quot;</td>
</tr>
<tr>
<td>Null</td>
<td>&quot;object&quot;</td>
</tr>
</tbody>
</table>

**Example**
The following code shows how to implement **typeof** operator.

```html
<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = "String";
var linebreak = "<br />

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>

<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```
Output

Result => B is String
Result => A is Numeric

Set the variables to different values and different operators and then try...
While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the **if..else** statement.

**Flow Chart of if-else**

The following flow chart shows how the if-else statement works.

JavaScript supports the following forms of **if..else** statement:

- if statement
- if...else statement
- if...else if... statement
**if Statement**

The `if` statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

**Syntax**

The syntax for a basic if statement is as follows:

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

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

**Example**

Try the following example to understand how the `if` statement works.

```html
<html>
<body>

<script type="text/javascript">
<!--
var age = 20;
if( age > 18 ){
    document.write("<b>Qualifies for driving</b>"se);  
}
//-->
</script>

<p>Set the variable to different value and then try... </p>
</body>
</html>
```
The 'if...else' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax
The syntax of an if-else statement is as follows:

```javascript
if (expression){
    Statement(s) to be executed if expression is true
}else{
    Statement(s) to be executed if expression is false
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example
Try the following code to learn how to implement an if-else statement in JavaScript.

```html
<html>
<body>

<script type="text/javascript">

var age = 15;

if( age > 18 ){
    document.write("<b>Qualifies for driving</b>");
}else{
    document.write("<b>Does not qualify for driving</b>");

</script>

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

<p>Set the variable to different value and then try...</p>
</body>
</html>

Output

Does not qualify for driving
Set the variable to different value and then try...

if...else if... Statement

The ‘if...else if...’ statement is an advanced form of if...else that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows:

```javascript
if (expression 1){
    Statement(s) to be executed if expression 1 is true
} else if (expression 2){
    Statement(s) to be executed if expression 2 is true
} else if (expression 3){
    Statement(s) to be executed if expression 3 is true
} else{
    Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example
Try the following code to learn how to implement an if-else-if statement in JavaScript.

```html
<html>
<body>
<script type="text/javascript">
<!--
var book = "maths";
if( book == "history" ){
    document.write("<b>History Book</b>"IDGE);
}else if( book == "maths" ){
    document.write("<b>Maths Book</b>"IDGE);
}else if( book == "economics" ){
    document.write("<b>Economics Book</b>"IDGE);
}else{
    document.write("<b>Unknown Book</b>"IDGE);
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

**Maths Book**

Set the variable to different value and then try...
You can use multiple `if...else...if` statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a `switch` statement which handles exactly this situation, and it does so more efficiently than repeated `if...else if` statements.

**Flow Chart**

The following flow chart explains a switch-case statement works.
**Syntax**

The objective of a `switch` statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each `case` against the value of the expression until a match is found. If nothing matches, a `default` condition will be used.

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

The `break` statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain `break` statement in [Loop Control](#) chapter.

**Example**

Try the following example to implement switch-case statement.

```html
<html>
<body>

<script type="text/javascript">
<!-
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
    case 'A': document.write("Good job<br />");
```
break;  
case 'B': document.write("Pretty good<br />");  
    break;  
case 'C': document.write("Passed<br />");  
    break;  
case 'D': document.write("Not so good<br />");  
    break;  
case 'F': document.write("Failed<br />");  
    break;  
    default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");  
//-->  
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>

Output

Entering switch block  
Good job  
Exiting switch block  
Set the variable to different value and then try...

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

<html>  
<body>

<script type="text/javascript">  
<!--  
var grade='A';  
document.write("Entering switch block<br "/");
</script>

</body>
</html>
switch (grade)
{
    case 'A': document.write("Good job<br />");
    case 'B': document.write("Pretty good<br />");
    case 'C': document.write("Passed<br />");
    case 'D': document.write("Not so good<br />");
    case 'F': document.write("Failed<br />");
    default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");

<p>Set the variable to different value and then try...</p>
While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

JavaScript supports all the necessary loops to ease down the pressure of programming.

**The while Loop**

The most basic loop in JavaScript is the `while` loop which would be discussed in this chapter. The purpose of a `while` loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes `false`, the loop terminates.

**Flow Chart**

The flow chart of `while loop` looks as follows:
Syntax

The syntax of **while loop** in JavaScript is as follows:

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

Example

Try the following example to implement while loop.

```html
<html>
<body>

<script type="text/javascript">
  <!--
  var count = 0;
  document.write("Starting Loop ");
  while (count < 10){
      document.write("Current Count : " + count + "<br />");
      count++;
  }
  document.write("Loop stopped!" );
  //-->  
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

Starting Loop Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
The do...while Loop

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

Flow Chart

The flow chart of a **do-while** loop would be as follows:

```
do{
    conditional code;
}while(condition);
```

Syntax

The syntax for **do-while** loop in JavaScript is as follows:
do{
    Statement(s) to be executed;
} while (expression);

**Note:** Don’t miss the semicolon used at the end of the do...while loop.

**Example**

Try the following example to learn how to implement a do-while loop in JavaScript.

```html
<html>
<body>

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

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

**Output**

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then try...
The for Loop

The ‘for’ loop is the most compact form of looping. It includes the following three important parts:

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.

- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.

- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Flow Chart

The flow chart of a **for** loop in JavaScript would be as follows:
Syntax
The syntax of for loop is JavaScript is as follows:

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

Example
Try the following example to learn how a for loop works in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++){
    document.write("Current Count : " + count );
    document.write("<br />");
}
document.write("Loop stopped!");
//--> 
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!

Set the variable to different value and then try...
The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

**Syntax**

The syntax of ‘for..in’ loop is:

```javascript
for (variblename in object){
    statement or block to execute
}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

**Example**

Try the following example to implement ‘for-in’ loop. It prints the web browser’s **Navigator** object.

```html
<html>
<body>

<script type="text/javascript">
!---
var aProperty;
document.write("Navigator Object Properties<br /> ");
for (aProperty in navigator) {
    document.write(aProperty);
    document.write("<br />");
}
document.write ("Exiting from the loop!");
//-->
</script>

</body>
</html>
```
<p>Set the variable to different object and then try... </p>

Exiting from the loop! 

Set the variable to different object and then try...
JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

To handle all such situations, JavaScript provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

The break statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart

The flow chart of a break statement would look as follows:

```
Condition

true

false

Break statement

Conditional Code
```

Example

The following example illustrates the use of a break statement with a while loop. Notice how the loop breaks out early once x reaches 5 and reaches to document.write (..) statement just below to the closing curly brace:
<html>
<body>

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

<p>Set the variable to different value and then try...</p>
</body>
</html>

Output

Entering the loop
2
3
4
5
Exiting the loop!

Set the variable to different value and then try...

We have already seen the usage of **break** statement inside a **switch** statement.
The continue Statement

The `continue` statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a `continue` statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example

This example illustrates the use of a `continue` statement with a while loop. Notice how the `continue` statement is used to skip printing when the index held in variable `x` reaches 5.

```html
<html>
<body>
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 10)
{
    x = x + 1;
    if (x == 5){
        continue;  // skill rest of the loop body
    }
    document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output
Using Labels to Control the Flow

Starting from JavaScript 1.2, a label can be used with `break` and `continue` to control the flow more precisely. A `label` is simply an identifier followed by a colon (:) that is applied to a statement or a block of code. We will see two different examples to understand how to use labels with `break` and `continue`.

**Note:** Line breaks are not allowed between the `continue` or `break` statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Try the following two examples for a better understanding of Labels.

**Example 1**
The following example shows how to implement Label with a `break` statement.

```html
<html>
<body>
<script type="text/javascript">
<!--
    document.write("Entering the loop!<br /> ");
    outerloop:   // This is the label name
    for (var i = 0; i < 5; i++)
    {
        document.write("Outerloop: " + i + "<br />");
        innerloop:
        for (var j = 0; j < 5; j++)
        {
            document.write("Innerloop: " + j + "<br />");
            if (j == 3)
            {
                document.write("Breaking out of loop!");
                break outerloop;
            }
        }
    }
</script>
</body>
</html>
```
```javascript
{
    if (j > 3) break; // Quit the innermost loop
    if (i == 2) break innerloop; // Do the same thing
    if (i == 4) break outerloop; // Quit the outer loop
    document.write("Innerloop: " + j + " <br />");
}

document.write("Exiting the loop!<br />");

//-->
</script>
</body>
</html>

Output
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!

Example 2
The following example shows how to implement Label with continue.

<html>
```
<body>
<script type="text/javascript">
<!--
    document.write("Entering the loop!<br /> ");
    outerloop: // This is the label name
    for (var i = 0; i < 3; i++)
    {
        document.write("Outerloop: " + i + "<br /> ");
        for (var j = 0; j < 5; j++)
        {
            if (j == 3){
                continue outerloop;
            }
            document.write("Innerloop: " + j + "<br /> ");
        }
    }
    document.write("Exiting the loop!<br /> ");
//-->
</script>

</body>
</html>

Output

Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 2
Innerloop: 0
Innerloop: 1
| Innerloop: 2 |
| Exiting the loop! |
A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like alert() and write() in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

**Function Definition**

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the `function` keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

**Syntax**

The basic syntax is shown here.

```html
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
    statements
}
//-->
</script>
```

**Example**

Try the following example. It defines a function called sayHello that takes no parameters:
<script type="text/javascript">
<!--[--
function sayHello()
{
    alert("Hello there");
}
//-->
</script>

**Calling a Function**

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```html
<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>

<p>Use different text in write method and then try...</p>
</body>
</html>
```

**Output**
Click the following button to call the function

Say Hello

**Function Parameters**

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

**Example**

Try the following example. We have modified our `sayHello` function here. Now it takes two parameters.

```html
<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>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```
The return Statement

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```html
<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>
```
Click the following button to call the function

Use different parameters inside the function and then try...

There is a lot to learn about JavaScript functions, however we have covered the most important concepts in this tutorial.

Nested Functions

Prior to JavaScript 1.2, function definition was allowed only in top level global code, but JavaScript 1.2 allows function definitions to be nested within other functions as well. Still there is a restriction that function definitions may not appear within loops or conditionals. These restrictions on function definitions apply only to function declarations with the function statement.

As we'll discuss later in the next chapter, function literals (another feature introduced in JavaScript 1.2) may appear within any JavaScript expression, which means that they can appear within if and other statements.

Example

Try the following example to learn how to implement nested functions.

```html
<html>
</html>
```
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) {
    function square(x) { return x*x; }
    return Math.sqrt(square(a) + square(b));
}
function secondFunction(){
    var result;
    result = hypotenuse(1,2);
    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>
Use different parameters inside the function and then try...

**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.

**Note:** Constructor is a terminology from Object Oriented Programming. You may not feel comfortable for the first time, which is OK.

**Syntax**

Following is the syntax to create a function using `Function()` constructor along with the `new` operator.

```javascript
<script type="text/javascript">
<!--
var variablename = new Function(Arg1, Arg2..., "Function Body");
//-->
</script>
```

The `Function()` constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the `Function()` constructor is not passed any argument that specifies a name for the function it creates. The **unnamed** functions created with the `Function()` constructor are called **anonymous** functions.

**Example**

Try the following example.

```html
<html>
<head>
<script type="text/javascript">
<!--
var func = new Function("x", "y", "return x*y;");
//-->
</script>

function secondFunction(){
    var result;
    result = func(10,20);
}
</html>
```
Function Literals

JavaScript 1.2 introduces the concept of function literals which is another new way of defining functions. A function literal is an expression that defines an unnamed function.

Syntax

The syntax for a function literal is much like a function statement, except that it is used as an expression rather than a statement and no function name is required.
Syntactically, you can specify a function name while creating a literal function as follows.

```javascript
<script type="text/javascript">
<!--
var variablename = function FunctionName(Argument List){
    Function Body
};
//-->
</script>

But this name does not have any significance, so it is not worthwhile.

**Example**

Try the following example. It shows the usage of function literals.

```html
<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>

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...
What is an Event?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding HTML Event Reference. Here we will see a few examples to understand the relation between Event and JavaScript.

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

Try the following example.

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

function sayHello() {
    document.write("Hello World")
}

</script>
</head>
</html>
```
onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a validate() function before submitting a form data to the webserver. If validate() function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

```html
<html>
<head>
<script type="text/javascript">
!---
function validation() {
    all validation goes here
    ........
    return either true or false
}
//---
</script>
</head>
<body>
<p>Click the following button and see result</p>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```
onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```html
<html>
<head>
<script type="text/javascript">
!---
function over() {
    document.write ("Mouse Over");
}
function out() {
    document.write ("Mouse Out");
}
//-->
</script>
</head>
<body>
<p>Bring your mouse inside the division to see the result:</p>
<div onmouseover="over()" onmouseout="out()">
<h2>This is inside the division</h2>
</div>
</body>
</html>
```
Output

Bring your mouse inside the division to see the result:

This is inside the division

**HTML 5 Standard Events**

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline</td>
<td>script</td>
<td>Triggers when the document goes offline</td>
</tr>
<tr>
<td>Onabort</td>
<td>script</td>
<td>Triggers on an abort event</td>
</tr>
<tr>
<td>onafterprint</td>
<td>script</td>
<td>Triggers after the document is printed</td>
</tr>
<tr>
<td>onbeforeonload</td>
<td>script</td>
<td>Triggers before the document loads</td>
</tr>
<tr>
<td>onbeforeprint</td>
<td>script</td>
<td>Triggers before the document is printed</td>
</tr>
<tr>
<td>onblur</td>
<td>script</td>
<td>Triggers when the window loses focus</td>
</tr>
<tr>
<td>oncanplay</td>
<td>script</td>
<td>Triggers when media can start play, but might has to stop for buffering</td>
</tr>
<tr>
<td>oncanplaythrough</td>
<td>script</td>
<td>Triggers when media can be played to the end, without stopping for buffering</td>
</tr>
<tr>
<td>onchange</td>
<td>script</td>
<td>Triggers when an element changes</td>
</tr>
<tr>
<td>onclick</td>
<td>script</td>
<td>Triggers on a mouse click</td>
</tr>
<tr>
<td>oncontextmenu</td>
<td>script</td>
<td>Triggers when a context menu is triggered</td>
</tr>
<tr>
<td>ondblclick</td>
<td>script</td>
<td>Triggers on a mouse double-click</td>
</tr>
<tr>
<td>ondrag</td>
<td>script</td>
<td>Triggers when an element is dragged</td>
</tr>
<tr>
<td>ondragend</td>
<td>script</td>
<td>Triggers at the end of a drag operation</td>
</tr>
<tr>
<td>Event</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>ondragenter</td>
<td>script</td>
<td>Triggers when an element has been dragged to a valid drop target</td>
</tr>
<tr>
<td>ondragleave</td>
<td>script</td>
<td>Triggers when an element leaves a valid drop target</td>
</tr>
<tr>
<td>ondragover</td>
<td>script</td>
<td>Triggers when an element is being dragged over a valid drop target</td>
</tr>
<tr>
<td>ondragstart</td>
<td>script</td>
<td>Triggers at the start of a drag operation</td>
</tr>
<tr>
<td>ondrop</td>
<td>script</td>
<td>Triggers when dragged element is being dropped</td>
</tr>
<tr>
<td>ondurationchange</td>
<td>script</td>
<td>Triggers when the length of the media is changed</td>
</tr>
<tr>
<td>onemptied</td>
<td>script</td>
<td>Triggers when a media resource element suddenly becomes empty.</td>
</tr>
<tr>
<td>onended</td>
<td>script</td>
<td>Triggers when media has reach the end</td>
</tr>
<tr>
<td>onerror</td>
<td>script</td>
<td>Triggers when an error occur</td>
</tr>
<tr>
<td>onfocus</td>
<td>script</td>
<td>Triggers when the window gets focus</td>
</tr>
<tr>
<td>onformchange</td>
<td>script</td>
<td>Triggers when a form changes</td>
</tr>
<tr>
<td>onforminput</td>
<td>script</td>
<td>Triggers when a form gets user input</td>
</tr>
<tr>
<td>onhaschange</td>
<td>script</td>
<td>Triggers when the document has change</td>
</tr>
<tr>
<td>oninput</td>
<td>script</td>
<td>Triggers when an element gets user input</td>
</tr>
<tr>
<td>oninvalid</td>
<td>script</td>
<td>Triggers when an element is invalid</td>
</tr>
<tr>
<td>onkeydown</td>
<td>script</td>
<td>Triggers when a key is pressed</td>
</tr>
<tr>
<td>onkeypress</td>
<td>script</td>
<td>Triggers when a key is pressed and released</td>
</tr>
<tr>
<td>onkeyup</td>
<td>script</td>
<td>Triggers when a key is released</td>
</tr>
<tr>
<td>Event</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>onload</td>
<td>script</td>
<td>Triggers when the document loads</td>
</tr>
<tr>
<td>onloadeddata</td>
<td>script</td>
<td>Triggers when media data is loaded</td>
</tr>
<tr>
<td>onloadedmetadata</td>
<td>script</td>
<td>Triggers when the duration and other media data of a media element is loaded</td>
</tr>
<tr>
<td>onloadstart</td>
<td>script</td>
<td>Triggers when the browser starts to load the media data</td>
</tr>
<tr>
<td>onmessage</td>
<td>script</td>
<td>Triggers when the message is triggered</td>
</tr>
<tr>
<td>onmousedown</td>
<td>script</td>
<td>Triggers when a mouse button is pressed</td>
</tr>
<tr>
<td>onmousemove</td>
<td>script</td>
<td>Triggers when the mouse pointer moves</td>
</tr>
<tr>
<td>onmouseout</td>
<td>script</td>
<td>Triggers when the mouse pointer moves out of an element</td>
</tr>
<tr>
<td>onmouseover</td>
<td>script</td>
<td>Triggers when the mouse pointer moves over an element</td>
</tr>
<tr>
<td>onmouseup</td>
<td>script</td>
<td>Triggers when a mouse button is released</td>
</tr>
<tr>
<td>onmousewheel</td>
<td>script</td>
<td>Triggers when the mouse wheel is being rotated</td>
</tr>
<tr>
<td>onoffline</td>
<td>script</td>
<td>Triggers when the document goes offline</td>
</tr>
<tr>
<td>online</td>
<td>script</td>
<td>Triggers when the document comes online</td>
</tr>
<tr>
<td>ononline</td>
<td>script</td>
<td>Triggers when the document comes online</td>
</tr>
<tr>
<td>onpagehide</td>
<td>script</td>
<td>Triggers when the window is hidden</td>
</tr>
<tr>
<td>onpageshow</td>
<td>script</td>
<td>Triggers when the window becomes visible</td>
</tr>
<tr>
<td>onpause</td>
<td>script</td>
<td>Triggers when media data is paused</td>
</tr>
<tr>
<td>onplay</td>
<td>script</td>
<td>Triggers when media data is going to start playing</td>
</tr>
</tbody>
</table>

**JavaScript**
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onplaying</td>
<td>script</td>
<td>Triggers when media data has start playing</td>
</tr>
<tr>
<td>onpopstate</td>
<td>script</td>
<td>Triggers when the window's history changes</td>
</tr>
<tr>
<td>onprogress</td>
<td>script</td>
<td>Triggers when the browser is fetching the media data</td>
</tr>
<tr>
<td>onratechange</td>
<td>script</td>
<td>Triggers when the media data's playing rate has changed</td>
</tr>
<tr>
<td>onreadystatechange</td>
<td>script</td>
<td>Triggers when the ready-state changes</td>
</tr>
<tr>
<td>onredo</td>
<td>script</td>
<td>Triggers when the document performs a redo</td>
</tr>
<tr>
<td>onresize</td>
<td>script</td>
<td>Triggers when the window is resized</td>
</tr>
<tr>
<td>onscroll</td>
<td>script</td>
<td>Triggers when an element's scrollbar is being scrolled</td>
</tr>
<tr>
<td>onseeked</td>
<td>script</td>
<td>Triggers when a media element's seeking attribute is no longer true, and the seeking has ended</td>
</tr>
<tr>
<td>onseeking</td>
<td>script</td>
<td>Triggers when a media element's seeking attribute is true, and the seeking has begun</td>
</tr>
<tr>
<td>onselect</td>
<td>script</td>
<td>Triggers when an element is selected</td>
</tr>
<tr>
<td>onstalled</td>
<td>script</td>
<td>Triggers when there is an error in fetching media data</td>
</tr>
<tr>
<td>onstorage</td>
<td>script</td>
<td>Triggers when a document loads</td>
</tr>
<tr>
<td>onsubmit</td>
<td>script</td>
<td>Triggers when a form is submitted</td>
</tr>
<tr>
<td>onsuspend</td>
<td>script</td>
<td>Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched</td>
</tr>
<tr>
<td>ontimeupdate</td>
<td>script</td>
<td>Triggers when media changes its playing position</td>
</tr>
<tr>
<td>onundo</td>
<td>script</td>
<td>Triggers when a document performs an undo</td>
</tr>
<tr>
<td>Event</td>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>onunload</td>
<td>script</td>
<td>Triggers when the user leaves the document</td>
</tr>
<tr>
<td>onvolumechange</td>
<td>script</td>
<td>Triggers when media changes the volume, also when volume is set to &quot;mute&quot;</td>
</tr>
<tr>
<td>onwaiting</td>
<td>script</td>
<td>Triggers when media has stopped playing, but is expected to resume</td>
</tr>
</tbody>
</table>
What are Cookies?

Web Browsers and Servers use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. But how to maintain users' session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

How It Works?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields:

- **Expires**: The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain**: The domain name of your site.
- **Path**: The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure**: If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value**: Cookies are set and retrieved in the form of key-value pairs.

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.
JavaScript can also manipulate cookies using the `cookie` property of the `Document` object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

### Storing Cookies

The simplest way to create a cookie is to assign a string value to the `document.cookie` object, which looks like this.

```javascript
document.cookie = "key1=value1;key2=value2;expires=date;"
```

Here the `expires` attribute is optional. If you provide this attribute with a valid date or time, then the cookie will expire on a given date or time and thereafter, the cookies' value will not be accessible.

**Note:** Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript `escape()` function to encode the value before storing it in the cookie. If you do this, you will also have to use the corresponding `unescape()` function when you read the cookie value.

### Example

Try the following. It sets a customer name in an input cookie.

```html
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
    if( document.myform.customer.value == "" ){
        alert ("Enter some value!");
        return;
    }
}

cookievalue= escape(document.myform.customer.value) + ";";
document.cookie="name=" + cookievalue;
document.write ("Setting Cookies : " + "name=" + cookievalue);
}
//-->
</script>
</head>
</html>
```
Output

Enter name:  
Set

Now your machine has a cookie called name. You can set multiple cookies using multiple key=value pairs separated by comma.

Reading Cookies

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie. The document.cookie string will keep a list of name=value pairs separated by semicolons, where name is the name of a cookie and value is its string value.

You can use strings' split() function to break a string into key and values as follows:

Example

Try the following example to get all the cookies.

```html
<html>
<head>
<script type="text/javascript">
<!--
function ReadCookie()
{
    var allcookies = document.cookie;
    document.write ("All Cookies : " + allcookies );
</script>
</head>
<body>
<form name="myform" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie();"/>
</form>
</body>
</html>
```
// Get all the cookies pairs in an array
cookiearray = allcookies.split(';;');

// Now take key value pair out of this array
for(var i=0; i<cookiearray.length; i++){
    name = cookiearray[i].split('=')[0];
    value = cookiearray[i].split('=')[1];
    document.write("Key is : " + name + " and Value is : " + value);
}

//-->
<script>
</script>
</head>
<body>
<form name="myform" action="">
<p>click the following button and see the result:</p>
<input type="button" value="Get Cookie" onclick="ReadCookie()"/>
</form>
</body>
</html>

Note: Here **length** is a method of **Array** class which returns the length of an array. We will discuss Arrays in a separate chapter. By that time, please try to digest it.

**Output**

click the following button and see the result:
Get Cookie

Note: There may be some other cookies already set on your machine. The above code will display all the cookies set on your machine.
Setting Cookies Expiry Date

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiry date within the cookie. This can be done by setting the `expires` attribute to a date and time.

Example

Try the following example. It illustrates how to extend the expiry date of a cookie by 1 Month.

```html
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
    var now = new Date();
    now.setMonth( now.getMonth() + 1 );
    cookievalue = escape(document.myform.customer.value) + ";"
    document.cookie = "name=" + cookievalue;
    document.cookie = "expires=" + now.toUTCString() + ";"
    document.write ("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>
```
Deleting a Cookie

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiry date to a time in the past.

Example

Try the following example. It illustrates how to delete a cookie by setting its expiry date to one month behind the current date.

```html
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
  var now = new Date();
  now.setMonth( now.getMonth() - 1 );
  cookievalue = escape(document.myform.customer.value) + ";"
  document.cookie="name=" + cookievalue;
  document.cookie = "expires=" + now.toUTCString() + ";"
  document.write("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
```
Output

Enter Cookie Name: [Input Field] Set Cookie
What is Page Redirection?
You might have encountered a situation where you clicked a URL to reach a page X but internally you were directed to another page Y. It happens due to **page redirection**. This concept is different from JavaScript Page Refresh.

There could be various reasons why you would like to redirect a user from the original page. We are listing down a few of the reasons:

- You did not like the name of your domain and you are moving to a new one. In such a scenario, you may want to direct all your visitors to the new site. Here you can maintain your old domain but put a single page with a page redirection such that all your old domain visitors can come to your new domain.

- You have built-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server-side page redirection, you can use client-side page redirection to land your users on the appropriate page.

- The Search Engines may have already indexed your pages. But while moving to another domain, you would not like to lose your visitors coming through search engines. So you can use client-side page redirection. But keep in mind this should not be done to fool the search engine, it could lead your site to get banned.

### JavaScript Page Refresh
You can refresh a web page using JavaScript `location.reload` method. This code can be called automatically upon an event or simply when the user clicks on a link. If you want to refresh a web page using a mouse click, then you can use the following code:

```html
<a href="javascript:location.reload(true)">Refresh Page</a>
```

### Auto Refresh
You can also use JavaScript to refresh the page automatically after a given time period. Here `setTimeout()` is a built-in JavaScript function which can be used to execute another function after a given time interval.
Example
Try the following example. It shows how to refresh a page after every 5 seconds. You can change this time as per your requirement.

```html
<html>
<head>
<script type="text/JavaScript">
<!--
function AutoRefresh( t ) {
    setTimeout("location.reload(true);", t);
}
// -->
</script>
</head>
<body onload="JavaScript:AutoRefresh(5000);">
<p>This page will refresh every 5 seconds.</p>
</body>
</html>
```

Output
This page will refresh every 5 seconds.

How Page Re-direction Works?

The implementations of Page-Redirection are as follows.

Example 1
It is quite simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows.

```html
<html>
<head>
<script type="text/javascript">
<!--
function Redirect() {
```
window.location="http://www.tutorialspoint.com";

/
//-->
</script>
</head>
<body>
<p>Click the following button, you will be redirected to home page.</p>
<form>
<input type="button" value="Redirect Me" onclick="Redirect()" />
</form>
</body>
</html>

Output

Click the following button, you will be redirected to home page.

Redirect Me

Example 2
You can show an appropriate message to your site visitors before redirecting them to a new page. This would need a bit time delay to load a new page. The following example shows how to implement the same. Here setTimeout() is a built-in JavaScript function which can be used to execute another function after a given time interval.

<html>
<head>
<script type="text/javascript">
/
function Redirect() {
    window.location="http://www.tutorialspoint.com";
}
document.write ("You will be redirected to our main page in 10 seconds!");
</script>
</head>
</html>
setTimeout('Redirect()', 10000);
//-->
</script>
</head>
<body>
</body>
</html>

Output
You will be redirected to tutorialspoint.com main page in 10 seconds!

Example 3
The following example shows how to redirect your site visitors onto a different page based on their browsers.

<html>
<head>
<script type="text/javascript">
<!--
var browsername=navigator.appName;
if( browsername == "Netscape" )
{
    window.location="http://www.location.com/ns.htm";
}
else if ( browsername =="Microsoft Internet Explorer")
{
    window.location="http://www.location.com/ie.htm";
}
else
{
    window.location="http://www.location.com/other.htm";
}
//-->
</script>
</head>
<body>
</body>
</html>
</head>
<body>
</body>
</html>
JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

**Alert Dialog Box**

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

**Example**

```html
<html>
<head>
<script type="text/javascript">
<!--
function Warn() {
    alert ("This is a warning message!");
    document.write ("This is a warning message!");
}
//-->
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form>
<input type="button" value="Click Me" onclick="Warn();" />
</form>
</body>
</html>
```
Confirmation Dialog Box

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method `confirm()` will return true. If the user clicks on the Cancel button, then `confirm()` returns false. You can use a confirmation dialog box as follows.

Example

```html
<html>
<head>
<script type="text/javascript">
!!--
function getConfirmation(){
    var retVal = confirm("Do you want to continue ?");
    if( retVal == true ){
        document.write ("User wants to continue!");
        return true;
    }else{
        Document.write ("User does not want to continue!");
        return false;
    }
}
//-->
</script>
</head>
<body>
<form>
Click Me
</form>
</p>Click the following button to see the result: </p>
```
Prompt Dialog Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called prompt() which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.

Example

The following example shows how to use a prompt dialog box:

```html
<html>
<head>
<script type="text/javascript">
<!--
function getValue(){
  var retVal = prompt("Enter your name : ", "your name here");

  document.write("You have entered : " + retVal);
}
//-->
</script>
</head>
<body>
<input type="button" value="Click Me" onclick="getValue();" />
</form>
</body>
</html>
```
<script>
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form>
<input type="button" value="Click Me" onclick="getValue();" />
</form>
</body>
</html>

Output

Click the following button to see the result:  
Click Me
**void** is an important keyword in JavaScript which can be used as a unary operator that appears before its single operand, which may be of any type. This operator specifies an expression to be evaluated without returning a value.

**Syntax**
The syntax of **void** can be either of the following two:

```html
<head>
<script type="text/javascript">
  <!--
  void func()
  javascript:void func()
  OR
  void(func())
  javascript:void(func())
  //-->
  </script>
</head>
```

**Example 1**
The most common use of this operator is in a client-side **javascript: URL**, where it allows you to evaluate an expression for its side-effects without the browser displaying the value of the evaluated expression.

Here the expression **alert ('Warning!!!')** is evaluated but it is not loaded back into the current document:

```html
<html>
<head>
<script type="text/javascript">
  <!--
  //-->
  </script>
</head>
```
Example 2

Take a look at the following example. The following link does nothing because the expression "0" has no effect in JavaScript. Here the expression "0" is evaluated, but it is not loaded back into the current document.

```html
<html>
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<p>Click the following, This won't react at all...</p>
<a href="javascript:void(0)">Click me!</a>
</body>
</html>
```

Output

Click the following, This won't react at all...

Click me!

Example 3

Another use of void is to purposely generate the undefined value as follows.

```html
<html>
<body>
<p>Click the following, This won't react at all...</p>
<a href="javascript:undefined">Click me!</a>
</body>
</html>
```
<head>
<script type="text/javascript">
<!--
function getValue(){
    var a,b,c;

    a = void ( b = 5, c = 7 );
    document.write('a = ' + a + ' b = ' + b + ' c = ' + c );
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="getValue();" />
</form>
</body>
</html>

Output

Click the following button to see the result:

Click Me
Many times you would like to place a button on your webpage to print the content of that web page via an actual printer. JavaScript helps you to implement this functionality using the `print` function of `window` object.

The JavaScript print function `window.print()` prints the current web page when executed. You can call this function directly using the `onclick` event as shown in the following example.

**Example**

Try the following example.

```html
<head>
  <script type="text/javascript">
    <!--
    //-->
  </script>
</head>
<body>
<form>
  <input type="button" value="Print" onclick="window.print()" />
</form>
</body>
```

**Output**

![Print button]

Although it serves the purpose of getting a printout, it is not a recommended way. A printer friendly page is really just a page with text, no images, graphics, or advertising.

You can make a page printer friendly in the following ways:

1. Make a copy of the page and leave out unwanted text and graphics, then link to that printer friendly page from the original. Check Example.
2. If you do not want to keep an extra copy of a page, then you can mark your printable text using proper comments like <!-- PRINT STARTS HERE -->..... <!-- PRINT ENDS HERE --> and then you can use PERL or any other script in the background to purge printable text and display for final printing. We at Tutorialspoint use this method to provide print facility to our site visitors. Check Example.

**How to Print a Page?**

If you don’t find the above facilities on a web page, then you can use the browser’s standard toolbar to get print the web page. Follow the link as follows.

| File --> Print --> Click OK button. |
Part 2: JavaScript Objects
JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers:

- **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.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

### Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is:

```
objectName.objectProperty = propertyValue;
```

**For example**: The following code gets the document title using the "title" property of the `document` object.

```
var str = document.title;
```

### Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the `this` keyword.
Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example: Following is a simple example to show how to use the `write()` method of document object to write any content on the document.

```javascript
document.write("This is test");
```

**User-Defined Objects**

All user-defined objects and built-in objects are descendants of an object called `Object`.

**The new Operator**

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.

In the following example, the constructor methods are `Object()`, `Array()`, and `Date()`. These constructors are built-in JavaScript functions.

```javascript
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

**The Object () Constructor**

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called `Object()` to build the object. The return value of the `Object()` constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the `var` keyword.

**Example 1**

Try the following example; it demonstrates how to create an Object.

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

Output

Book name is : Perl
Book author is : Mohtashim

Example 2
This example demonstrates how to create an object with a User-Defined Function. Here `this` keyword is used to refer to the object that has been passed to a function.

```javascript
<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("Perl", "Mohtashim");
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
</script>
</body>
</html>
```
Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

Example

Try the following example; it shows how to add a function along with an object.

```html
<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>
</body>
</html>
```
The 'with' Keyword

The 'with' keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to with becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax

The syntax for with object is as follows:

```javascript
with (object){
    properties used without the object name and dot
}
```

Example

Try the following example.

```html
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    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>
</html>
```

Output

Book title is : Perl
Book author is : Mohtashim
Book price is : 100
// Define a function which will work as a method
function addPrice(amount)
{
    with(this){
        price = amount;
    }
}

function book(title, author){
    this.title = title;
    this.author = author;
    this.price = 0;
    this.addPrice = addPrice; // Assign that method as property.
}
</script>
</head>
<body>
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    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>
</html>

Output

Book title is : Perl
Book author is : Mohtashim
Book price is : 100
The `Number` object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about `Number` objects because the browser automatically converts number literals to instances of the number class.

**Syntax**
The syntax for creating a `number` object is as follows:

```javascript
var val = new Number(number);
```

In the place of `number`, if you provide any non-number argument, then the argument cannot be converted into a number, it returns NaN (Not-a-Number).

**Number Properties**

Here is a list of each property and their description.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_VALUE</td>
<td>The largest possible value a number in JavaScript can have 1.7976931348623157E+308</td>
</tr>
<tr>
<td>MIN_VALUE</td>
<td>The smallest possible value a number in JavaScript can have 5E-324</td>
</tr>
<tr>
<td>NaN</td>
<td>Equal to a value that is not a number.</td>
</tr>
<tr>
<td>NEGATIVE_INFINITY</td>
<td>A value that is less than MIN_VALUE.</td>
</tr>
<tr>
<td>POSITIVE_INFINITY</td>
<td>A value that is greater than MAX.VALUE</td>
</tr>
<tr>
<td>prototype</td>
<td>A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document</td>
</tr>
<tr>
<td>constructor</td>
<td>Returns the function that created this object’s instance. By default this is the Number object.</td>
</tr>
</tbody>
</table>
In the following sections, we will take a few examples to demonstrate the properties of Number.

**MAX_VALUE**

The `Number.MAX_VALUE` property belongs to the static `Number` object. It represents constants for the largest possible positive numbers that JavaScript can work with.

The actual value of this constant is $1.7976931348623157 \times 10^{308}$.

**Syntax**

The syntax to use MAX_VALUE is:

```javascript
var val = Number.MAX_VALUE;
```

**Example**

Try the following example to learn how to use MAX_VALUE.

```html
<html>
<head>
<script type="text/javascript">
!---
function showValue()
{
    var val = Number.MAX_VALUE;

    document.write ("Value of Number.MAX_VALUE : " + val);
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="showValue();" />
</form>
</body>
</html>
```
MIN_VALUE

The `Number.MIN_VALUE` property belongs to the static `Number` object. It represents constants for the smallest possible positive numbers that JavaScript can work with.

The actual value of this constant is $5 \times 10^{-324}$.

**Syntax**

The syntax to use MIN_VALUE is:

```javascript
var val = Number.MIN_VALUE;
```

**Example**

Try the following example.

```html
<html>
<head>
<script type="text/javascript">
<!--
function showValue()
{
    var val = Number.MIN_VALUE;
    alert("Value of Number.MIN_VALUE : " + val);
}
//--> 
</script>
</head>
<body>
<form>
</form>
</body>
</html>
```

Value of Number.MAX_VALUE : $1.7976931348623157 \times 10^{308}$
Click the following to see the result:

Value of Number.MIN_VALUE : 5e-324

**NaN**

Unquoted literal constant NaN is a special value representing Not-a-Number. Since NaN always compares unequal to any number, including NaN, it is usually used to indicate an error condition for a function that should return a valid number.

**Note:** Use the `isNaN()` global function to see if a value is an NaN value.

**Syntax**
The syntax to use NaN is:

```javascript
var val = Number.NaN;
```

**Example**

Try the following example to learn how to use NaN.
```javascript
<head>
<script type="text/javascript">
<!--
function showValue()
{
    var dayOfMonth = 50;
    if (dayOfMonth < 1 || dayOfMonth > 31)
    {
        dayOfMonth = Number.NaN
        alert("Day of Month must be between 1 and 31.");
    }
    Document.write("Value of dayOfMonth : " + dayOfMonth );
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="showValue();" />
</form>
</body>
</html>
```

**Output**

Click the following to see the result:

[Click Me]
NEGATIVE_INFINITY

This is a special numeric value representing a value less than Number.MIN_VALUE. This value is represented as "-Infinity". It resembles an infinity in its mathematical behavior. For example, anything multiplied by NEGATIVE_INFINITY is NEGATIVE_INFINITY, and anything divided by NEGATIVE_INFINITY is zero.

Because NEGATIVE_INFINITY is a constant, it is a read-only property of Number.

Syntax
The syntax to use NEGATIVE_INFINITY is as follows:

```javascript
var val = Number.NEGATIVE_INFINITY;
```

Example
Try the following example.

```html
<html>
<head>
<script type="text/javascript">
<!--
function showValue()
{
    var smallNumber = (-Number.MAX_VALUE) * 2
    if (smallNumber == Number.NEGATIVE_INFINITY) {
        alert("Value of smallNumber : " + smallNumber);
    }
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
</form>
</html>
```
POSITIVE_INFINITY

This is a special numeric value representing any value greater than Number.MAX_VALUE. This value is represented as "Infinity". It resembles an infinity in its mathematical behavior. For example, anything multiplied by POSITIVE_INFINITY is POSITIVE_INFINITY, and anything divided by POSITIVE_INFINITY is zero.

As POSITIVE_INFINITY is a constant, it is a read-only property of Number.

Syntax

Use the following syntax to use POSITIVE_INFINITY.

```javascript
var val = Number.POSITIVE_INFINITY;
```

Example

Try the following example to learn how use POSITIVE_INFINITY.

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

function showValue()
{
    var bigNumber = Number.MAX_VALUE * 2
    if (bigNumber == Number.POSITIVE_INFINITY) {
```
javascript

```javascript
alert("Value of bigNumber : " + bigNumber );

}
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="showValue();" />
</form>
</body>
</html>

Output

Click the following to see the result:

Click Me

Value of val : Infinity

Prototype

The prototype property allows you to add properties and methods to any object (Number, Boolean, String and Date etc.).

Note: Prototype is a global property which is available with almost all the objects.

Syntax

Use the following syntax to use Prototype.

```javascript
object.prototype.name = value
```

Example
Try the following example to use the prototype property to add a property to an object.

```html
<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("Perl", "Mohtashim");
book.prototype.price = null;
myBook.price = 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>
</html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```
**constructor**

It returns a reference to the Number function that created the instance's prototype.

**Syntax**

Its syntax is as follows:

```
number.constructor()
```

**Return value**

Returns the function that created this object's instance.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript constructor() Method</title>
</head>
<body>
<script type="text/javascript">
  var num = new Number( 177.1234 );
  document.write("num.constructor() is : " + num.constructor);
</script>
</body>
</html>
```

**Output**

```
num.constructor() is : function Number() { [native code] }
```

**Number Methods**

The Number object contains only the default methods that are a part of every object's definition.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toExponential()</td>
<td>Forces a number to display in exponential notation, even if</td>
</tr>
</tbody>
</table>
the number is in the range in which JavaScript normally uses standard notation.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toFixed()</td>
<td>Formats a number with a specific number of digits to the right of the decimal.</td>
</tr>
<tr>
<td>toLocaleString()</td>
<td>Returns a string value version of the current number in a format that may vary according to a browser's local settings.</td>
</tr>
<tr>
<td>toPrecision()</td>
<td>Defines how many total digits (including digits to the left and right of the decimal) to display of a number.</td>
</tr>
<tr>
<td>toString()</td>
<td>Returns the string representation of the number's value.</td>
</tr>
<tr>
<td>valueOf()</td>
<td>Returns the number's value.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to explain the methods of Number.

**toExponential()**

This method returns a string representing the number object in exponential notation.

**Syntax**

Its syntax is as follows:

```javascript
number.toExponential([fractionDigits])
```

**Parameter Details**

**fractionDigits**: An integer specifying the number of digits after the decimal point. Defaults to as many digits as necessary to specify the number.

**Return Value**

A string representing a Number object in exponential notation with one digit before the decimal point, rounded to fractionDigits digits after the decimal point. If the fractionDigits argument is omitted, the number of digits after the decimal point defaults to the number of digits necessary to represent the value uniquely.

**Example**
Try the following example.

```html
<html>
<head>
<title>Javascript Method toExponential()</title>
</head>
<body>
<script type="text/javascript">
    var num=77.1234;
    var val = num.toExponential();
    document.write("num.toExponential() is : " + val + "\n");
    document.write("<br />");

    val = num.toExponential(4);
    document.write("num.toExponential(4) is : " + val + "\n");
    document.write("<br />");

    val = num.toExponential(2);
    document.write("num.toExponential(2) is : " + val + "\n");
    document.write("<br />");

    val = 77.1234.toExponential();
    document.write("77.1234.toExponential() is : " + val + "\n");
    document.write("<br />");

    val = 77.1234.toExponential();
    document.write("77.1234.toExponential() is : " + val + "\n");
</script>
</body>
</html>

Output

num.toExponential() is : 7.71234e+1
num.toExponential(4) is : 7.7123e+1
num.toExponential(2) is : 7.71e+1
77.1234.toExponential() is : 7.71234e+1
77 .toExponential() is : 7.71234e+1

tofixed ()
This method formats a number with a specific number of digits to the right of the decimal.

Syntax
Its syntax is as follows:

number.toFixed( [digits] )

Parameter Details
digits: The number of digits to appear after the decimal point.

Return Value
A string representation of number that does not use exponential notation and has the exact number of digits after the decimal place.

Example
Try the following example.

<html>
<head>
<title>JavaScript toFixed() Method</title>
</head>
<body>
<script type="text/javascript">
var num=177.1234;
document.write("num.toFixed() is : " + num.toFixed());
document.write("<br />");
document.write("num.toFixed(6) is : " + num.toFixed(6));
document.write("<br />");
document.write("num.toFixed(1) is : " + num.toFixed(1));
document.write("<br />");
document.write("(1.23e+20).toFixed(2) is:" +
(1.23e+20).toFixed(2));
</script>
</body>
</html>
```
document.write("<br />");
document.write("(1.23e-10).toFixed(2) is : " +
    (1.23e-10).toFixed(2));
</script>
</body>
</html>
```

### Output

```
num.toFixed() is : 177
num.toFixed(6) is : 177.123400
num.toFixed(1) is : 177.1
(1.23e+20).toFixed(2) is:1230000000000000000.00
(1.23e-10).toFixed(2) is : 0.00
```

### toLocaleString()

This method converts a `number` object into a human readable string representing the number using the locale of the environment.

#### Syntax

Its syntax is as follows:

```
number.toLocaleString()
```

#### Return Value

Returns a human readable string representing the number using the locale of the environment.

#### Example

Try the following example.

```
<html>
<head>
<title>JavaScript toLocaleString() Method </title>
</head>
<body>
<script type="text/javascript">
```
```javascript
var num = new Number(177.1234);
document.write( num.toLocaleString());
</script>
</body>
</html>
```

Output

177.123

toPrecision()

This method returns a string representing the number object to the specified precision.

Syntax

Its syntax is as follows:

```javascript
number.toPrecision( [ precision ] )
```

Parameter Details

**precision**: An integer specifying the number of significant digits.

Return Value

Returns a string representing a Number object in fixed-point or exponential notation rounded toPrecision significant digits.

Example

Try the following example.

```html
<html>
<head>
<title>JavaScript toPrecision() Method </title>
</head>
<body>
<script type="text/javascript">
    var num = new Number(7.123456);
</script>
</body>
</html>
```
document.write("num.toPrecision() is " + num.toPrecision());
document.write("<br/>");

document.write("num.toPrecision(4) is " + num.toPrecision(4));
document.write("<br/>");

document.write("num.toPrecision(2) is " + num.toPrecision(2));
document.write("<br/>");

document.write("num.toPrecision(1) is " + num.toPrecision(1));
</script>
</body>
</html>

**Output**

<table>
<thead>
<tr>
<th>num.toPrecision() is 7.123456</th>
</tr>
</thead>
<tbody>
<tr>
<td>num.toPrecision(4) is 7.123</td>
</tr>
<tr>
<td>num.toPrecision(2) is 7.1</td>
</tr>
<tr>
<td>num.toPrecision(1) is 7</td>
</tr>
</tbody>
</table>

**toString()**

This method returns a string representing the specified object. The toString() method parses its first argument, and attempts to return a string representation in the specified radix (base).

**Syntax**

Its syntax is as follows:

```
number.toString( [radix] )
```

**Parameter Details**

- **radix**: An integer between 2 and 36 specifying the base to use for representing numeric values.

**Return Value**

Returns a string representing the specified Number object.
Try the following example.

```html
<html>
<head>
<title>JavaScript toString() Method</title>
</head>
<body>
<script type="text/javascript">
    var num = new Number(15);
    document.write("num.toString() is " + num.toString());
    document.write("<br />");
    document.write("num.toString(2) is " + num.toString(2));
    document.write("<br />");
    document.write("num.toString(4) is " + num.toString(4));
    document.write("<br />");
</script>
</body>
</html>
```

**Output**

```
num.toString() is 15
num.toString(2) is 1111
num.toString(4) is 33
```

**valueOf ()**

This method returns the primitive value of the specified `number` object.

**Syntax**

Its syntax is as follows:

```
number.valueOf()
```
Returns the primitive value of the specified \texttt{number} object.

\textbf{Example}

Try the following example.

\begin{verbatim}
<html>
<head>
<title>JavaScript valueOf() Method</title>
</head>
<body>
<script type="text/javascript">
    var num = new Number(15.11234);
    document.write("num.valueOf() is "+num.valueOf());
</script>
</body>
</html>
\end{verbatim}

\textbf{Output}

\begin{verbatim}
num.valueOf() is 15.11234
\end{verbatim}
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.

**Syntax**

Use the following syntax to create a **boolean** object.

```javascript
var val = new Boolean(value);
```

**Boolean Properties**

Here is a list of the properties of Boolean object:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>Returns a reference to the Boolean function that created the object.</td>
</tr>
<tr>
<td>prototype</td>
<td>The prototype property allows you to add properties and methods to an object.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to illustrate the properties of Boolean object.

**constructor ()**

Javascript boolean **constructor()** method returns a reference to the Boolean function that created the instance's prototype.

**Syntax**

Use the following syntax to create a Boolean constructor() method.

```javascript
boolean.constructor()
```

**Return Value**
Returns the function that created this object's instance.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript constructor() Method</title>
</head>
<body>
<script type="text/javascript">
  var bool = new Boolean( );
  document.write("bool.constructor() is : " + bool.constructor);
</script>
</body>
</html>
```

**Output**

```
bool.constructor() is : function Boolean() { [native code] }
```

**Prototype**

The prototype property allows you to add properties and methods to any object (Number, Boolean, String and Date, etc.).

**Note:** Prototype is a global property which is available with almost all the objects.

**Syntax**

Use the following syntax to create a Boolean prototype.

```
object.prototype.name = value
```

**Example**
Try the following example; it shows how to use the prototype property to add a property to an object.

```html
<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("Perl", "Mohtashim");
book.prototype.price = null;
myBook.price = 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>
</html>
```

Output

<table>
<thead>
<tr>
<th>Book title is</th>
<th>Perl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book author is</td>
<td>Mohtashim</td>
</tr>
<tr>
<td>Book price is</td>
<td>100</td>
</tr>
</tbody>
</table>

Boolean Methods

Here is a list of the methods of Boolean object and their description.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toSource()</td>
<td>Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.</td>
</tr>
<tr>
<td>toString()</td>
<td>Returns a string of either &quot;true&quot; or &quot;false&quot; depending upon the value of the object.</td>
</tr>
<tr>
<td>valueOf()</td>
<td>Returns the primitive value of the Boolean object.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of the Boolean methods.

**toSource()**

Javascript boolean **toSource()** method returns a string representing the source code of the object.

**Note:** This method is not compatible with all the browsers.

**Syntax**

Its syntax is as follows:

```javascript
boolean.toSource()
```

**Return Value**

Returns a string representing the source code of the object.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript toSource() Method</title>
</head>
<body>
<script type="text/javascript">
function book(title, publisher, price)
{  
```
this.title = title;
this.publisher = publisher;
this.price = price;
}
var newBook = new book("Perl","Leo Inc",200);
document.write("newBook.toSource() is : "+ newBook.toSource());
</script>
</body>
</html>

Output

({title:"Perl", publisher:"Leo Inc", price:200})

toString ()

This method returns a string of either "true" or "false" depending upon the value of the object.

Syntax

Its syntax is as follows:

boolean.toString()

Return Value

Returns a string representing the specified Boolean object.

Example

Try the following example.

<html>
<head>
<title>JavaScript toString() Method</title>
</head>
<body>
<script type="text/javascript">
var flag = new Boolean(false);
document.write( "flag.toString is : " + flag.toString() );
</script>
</body>
</html>
Javascript boolean `valueOf()` method returns the primitive value of the specified boolean object.

Syntax
Its syntax is as follows:

```javascript
boolean.valueOf()
```

Return Value
Returns the primitive value of the specified boolean object.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript toString() Method</title>
</head>
<body>
<script type="text/javascript">
var flag = new Boolean(false);
document.write( "flag.valueOf is : " + flag.valueOf() );
</script>
</body>
</html>
```

Output

`flag.valueOf is : false`
Javascript
The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

**Syntax**

Use the following syntax to create a String object:

```javascript
var val = new String(string);
```

The `string` parameter is a series of characters that has been properly encoded.

**String Properties**

Here is a list of the properties of String object and their description.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>Returns a reference to the String function that created the object.</td>
</tr>
<tr>
<td>length</td>
<td>Returns the length of the string.</td>
</tr>
<tr>
<td>prototype</td>
<td>The prototype property allows you to add properties and methods to an object.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of String properties.

**constructor**

A constructor returns a reference to the string function that created the instance's prototype.
Syntax
Its syntax is as follows:

```javascript
string.constructor
```

Return Value
Returns the function that created this object's instance.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript String constructor property</title>
</head>
<body>
<script type="text/javascript">
    var str = new String( "This is string" );
    document.write("str.constructor is:" + str.constructor);
</script>
</body>
</html>
```

Output

```
str.constructor is:function String() { [native code] }
```

Length
This property returns the number of characters in a string.

Syntax
Use the following syntax to find the length of a string:

```javascript
string.length
```

Return Value
Returns the number of characters in the string.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript String length Property</title>
</head>
<body>
<script type="text/javascript">
    var str = new String( "This is string" );
    document.write("str.length is:" + str.length);
</script>
</body>
</html>
```

Output

```
str.length is:14
```

**Prototype**

The prototype property allows you to add properties and methods to any object (Number, Boolean, String, Date, etc.).

**Note:** Prototype is a global property which is available with almost all the objects.

**Syntax**

Its syntax is as follows:

```
object.prototype.name = value
```

**Example**

Try the following example.

```html
<html>
<head>
<title>User-defined objects</title>
</head>
<script type="text/javascript">
</script>
</html>
```
function book(title, author){
    this.title = title;
    this.author  = author;
}
</script>
</head>
<body>
<script type="text/javascript">
    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 + ";
</script>
</body>
</html>

Output

Book title is: Perl
Book author is: Mohtashim
Book price is: 100

String Methods

Here is a list of the methods available in String object along with their description.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charAt()</td>
<td>Returns the character at the specified index.</td>
</tr>
<tr>
<td>charCodeAt()</td>
<td>Returns a number indicating the Unicode value of the character at the given index.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>concat()</td>
<td>Combines the text of two strings and returns a new string.</td>
</tr>
<tr>
<td>indexOf()</td>
<td>Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.</td>
</tr>
<tr>
<td>lastIndexOf()</td>
<td>Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.</td>
</tr>
<tr>
<td>localeCompare()</td>
<td>Returns a number indicating whether a reference string comes before or after or is the same as the given string in sorted order.</td>
</tr>
<tr>
<td>match()</td>
<td>Used to match a regular expression against a string.</td>
</tr>
<tr>
<td>replace()</td>
<td>Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.</td>
</tr>
<tr>
<td>search()</td>
<td>Executes the search for a match between a regular expression and a specified string.</td>
</tr>
<tr>
<td>slice()</td>
<td>Extracts a section of a string and returns a new string.</td>
</tr>
<tr>
<td>split()</td>
<td>Splits a String object into an array of strings by separating the string into substrings.</td>
</tr>
<tr>
<td>substr()</td>
<td>Returns the characters in a string beginning at the specified location through the specified number of characters.</td>
</tr>
<tr>
<td>substring()</td>
<td>Returns the characters in a string between two indexes into the string.</td>
</tr>
<tr>
<td>toLocaleLowerCase()</td>
<td>The characters within a string are converted to lower case while respecting the current locale.</td>
</tr>
<tr>
<td>toLocaleUpperCase()</td>
<td>The characters within a string are converted to upper case while respecting the current locale.</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>Returns the calling string value converted to lower case.</td>
</tr>
<tr>
<td>toString()</td>
<td>Returns a string representing the specified object.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>Returns the calling string value converted to uppercase.</td>
</tr>
<tr>
<td>valueOf()</td>
<td>Returns the primitive value of the specified object.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of String methods.

**charAt()**

charAt() is a method that returns the character from the specified index.

Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character in a string, called `stringName`, is `stringName.length - 1`.

**Syntax**

Use the following syntax to find the character at a particular index.

```
string.charAt(index)
```

**Argument Details**

- **index**: An integer between 0 and 1 less than the length of the string.

**Return Value**

Returns the character from the specified index.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript String charAt() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String( "This is string" );
    document.writeln("str.charAt(0) is:" + str.charAt(0));
    document.writeln("<br />str.charAt(1) is:" + str.charAt(1));
    document.writeln("<br />str.charAt(2) is:" + str.charAt(2));
</script>
</body>
</html>
```
Output

str.charAt(0) is:T
str.charAt(1) is:h
str.charAt(2) is:i
str.charAt(3) is:s
str.charAt(4) is:
str.charAt(5) is:i

charCodeAt ()

This method returns a number indicating the Unicode value of the character at the given index.

Unicode code points range from 0 to 1,114,111. The first 128 Unicode code points are a direct match of the ASCII character encoding. charCodeAt() always returns a value that is less than 65,536.

Syntax
Use the following syntax to find the character code at a particular index.

string.charCodeAt(index)

Argument Details
index: An integer between 0 and 1 less than the length of the string; if unspecified, defaults to 0.

Return Value
Returns a number indicating the Unicode value of the character at the given index. It returns NaN if the given index is not between 0 and 1 less than the length of the string.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript String charCodeAt() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String( "This is string" );
    document.write("str.charCodeAt(0) is:" + str.charCodeAt(0));
    document.write("<br />str.charCodeAt(1) is:" + str.charCodeAt(1));
    document.write("<br />str.charCodeAt(2) is:" + str.charCodeAt(2));
    document.write("<br />str.charCodeAt(3) is:" + str.charCodeAt(3));
    document.write("<br />str.charCodeAt(4) is:" + str.charCodeAt(4));
    document.write("<br />str.charCodeAt(5) is:" + str.charCodeAt(5));
</script>
</body>
</html>
```

**Output**

```
str.charCodeAt(0) is:84
str.charCodeAt(1) is:104
str.charCodeAt(2) is:105
str.charCodeAt(3) is:115
str.charCodeAt(4) is:32
str.charCodeAt(5) is:105
```

**contact ()**

This method adds two or more strings and returns a new single string.

**Syntax**

Its syntax is as follows:

```javascript
string.concat(string2, string3[, ..., stringN]);
```
**Argument Details**

string2...stringN: These are the strings to be concatenated.

**Return Value**

Returns a single concatenated string.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript String concat() Method</title>
</head>
<body>
<script type="text/javascript">
    var str1 = new String( "This is string one" );
    var str2 = new String( "This is string two" );
    var str3 = str1.concat( str2 );

    document.write("Concatenated String :" + str3);
</script>
</body>
</html>
```

**Output**

Concatenated String :This is string one This is string two

**indexOf()**

This method returns the index within the calling String object of the first occurrence of the specified value, starting the search at `fromIndex` or -1 if the value is not found.

**Syntax**

Use the following syntax to use the `indexOf()` method.

```
string.indexOf(searchValue[, fromIndex])
```
Argument Details

- **searchValue**: A string representing the value to search for.
- **fromIndex**: The location within the calling string to start the search from. It can be any integer between 0 and the length of the string. The default value is 0.

Return Value

Returns the index of the found occurrence, otherwise -1 if not found.

Example

Try the following example.

```html
<html>
<head>
<title>JavaScript String indexOf() Method</title>
</head>
<body>
<script type="text/javascript">
    var str1 = new String( "This is string one" );
    var index = str1.indexOf( "string" );
    document.write("indexOf found String :" + index );
    document.write("<br />");

    var index = str1.indexOf( "one" );
    document.write("indexOf found String :" + index );
</script>
</body>
</html>
```

Output

indexOf found String :8
indexOf found String :15
lastIndexOf ()

This method returns the index within the calling String object of the last occurrence of the specified value, starting the search at fromIndex or -1 if the value is not found.

Syntax
Its syntax is as follows:

```javascript
string.lastIndexOf(searchValue[, fromIndex])
```

Argument Details
- **searchValue**: A string representing the value to search for.
- **fromIndex**: The location within the calling string to start the search from. It can be any integer between 0 and the length of the string. The default value is 0.

Return Value
Returns the index of the last found occurrence, otherwise -1 if not found.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript String lastIndexOf() Method</title>
</head>
<body>
<script type="text/javascript">
  var str1 = new String( "This is string one and again string" );
  var index = str1.lastIndexOf( "string" );
  document.write("lastIndexOf found String :" + index);
  document.write("<br />");

  var index = str1.lastIndexOf( "one" );
  document.write("lastIndexOf found String :" + index);
</script>
</body>
</html>
```
localeCompare ()

This method returns a number indicating whether a reference string comes before or after or is the same as the given string in sorted order.

Syntax

The syntax of localeCompare() method is:

```
string.localeCompare( param )
```

Argument Details

**param** : A string to be compared with *string* object.

Return Value

- **0** : If the string matches 100%.
- **1** : no match, and the parameter value comes before the *string* object's value in the locale sort order
- **-1** : no match, and the parameter value comes after the *string* object's value in the local sort order

Example

Try the following example.

```
<html>
<head>
<title>JavaScript String localeCompare() Method</title>
</head>
<body>
<script type="text/javascript">
    var str1 = new String( "This is beautiful string" );
</script>
</body>
</html>
```
var index = str1.localeCompare( "XYZ" );
document.write("localeCompare first :" + index );

document.write("<br />");

var index = str1.localeCompare( "AbCD ?" );
document.write("localeCompare second :" + index );

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

Output
localeCompare first :-1
localeCompare second :1

match ()
This method is used to retrieve the matches when matching a string against a regular expression.

Syntax
Use the following syntax to use the match() method.

string.match ( param )

Argument Details
param : A regular expression object.

Return Value
- If the regular expression does not include the g flag, it returns the same result as regexp.exec(string).
- If the regular expression includes the g flag, the method returns an Array containing all the matches.

Example
Try the following example.
This method finds a match between a regular expression and a string, and replaces the matched substring with a new substring.

The replacement string can include the following special replacement patterns:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Inserts</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>Inserts a &quot;$&quot;.</td>
</tr>
<tr>
<td>$&amp;</td>
<td>Inserts the matched substring.</td>
</tr>
<tr>
<td>$´</td>
<td>Inserts the portion of the string that precedes the matched substring.</td>
</tr>
<tr>
<td>$´</td>
<td>Inserts the portion of the string that follows the matched substring.</td>
</tr>
</tbody>
</table>
$n$ or $nn$

Where $n$ or $nn$ are decimal digits, inserts the $n$th parenthesized submatch string, provided the first argument was a RegExp object.

### Syntax

The syntax to use the `replace()` method is as follows:

```
string.replace(regexp/substr, newSubStr/function[, flags]);
```

### Argument Details

- **regexp**: A `RegExp` object. The match is replaced by the return value of parameter #2.
- **substr**: A String that is to be replaced by `newSubStr`.
- **newSubStr**: The String that replaces the substring received from parameter #1.
- **function**: A function to be invoked to create the new substring.
- **flags**: A String containing any combination of the RegExp flags: `g` - global match, `i` - ignore case, `m` - match over multiple lines. This parameter is only used if the first parameter is a string.

### Return Value

It simply returns a new changed string.

### Example

Try the following example.

```html
<html>
<head>
<title>JavaScript String replace() Method</title>
</head>
<body>
<script type="text/javascript">
    var re = /apples/gi;
</script>
</body>
</html>
```
```javascript
var str = "Apples are round, and apples are juicy.";
var newstr = str.replace(re, "oranges");

document.write(newstr);
```

Output

oranges are round, and oranges are juicy.

Example

Try the following example; it shows how to switch words in a string.

```html
<html>
<head>
<title>JavaScript String replace() Method</title>
</head>
<body>
<script type="text/javascript">
var re = /\w+\s\w+/;
var str = "zara ali";
var newstr = str.replace(re, "$2, $1");
document.write(newstr);

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

Output

ali, zara
**Search ()**

This method executes the search for a match between a regular expression and this String object.

**Syntax**

Its syntax is as follows:

```javascript
string.search(regexp);
```

**Argument Details**

- **regexp**: A regular expression object. If a non-RegExp object `obj` is passed, it is implicitly converted to a RegExp by using `new RegExp(obj)`.

**Return Value**

If successful, the search returns the index of the regular expression inside the string. Otherwise, it returns `-1`.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript String search() Method</title>
</head>
<body>
<script type="text/javascript">
    var re = /apples/gi;
    var str = "Apples are round, and apples are juicy.";

    if ( str.search(re) == -1 ){
        document.write("Does not contain Apples ");
    }else{
        document.write("Contains Apples ");
    }
</script>
</body>
</html>
```
slice ()
This method extracts a section of a string and returns a new string.

Syntax
The syntax for slice() method is:

```
string.slice( beginslice [, endSlice] );
```

Argument Details
- **beginSlice**: The zero-based index at which to begin extraction.
- **endSlice**: The zero-based index at which to end extraction. If omitted, slice extracts to the end of the string.

Return Value
If successful, slice returns the index of the regular expression inside the string. Otherwise, it returns -1.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript String slice() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Apples are round, and apples are juicy.";

    var sliced = str.slice(3, -2);
</script>
</body>
</html>
```
Output

```
les are round, and apples are juic
```

split ()
This method splits a String object into an array of strings by separating the string into substrings.

Syntax
Its syntax is as follows:

```
string.split([separator][, limit]);
```

Argument Details
- **separator** : Specifies the character to use for separating the string. If `separator` is omitted, the array returned contains one element consisting of the entire string.
- **limit** : Integer specifying a limit on the number of splits to be found.

Return Value
The split method returns the new array. Also, when the string is empty, split returns an array containing one empty string, rather than an empty array.

Example
Try the following example.

```
<html>
<head>
```
<title>JavaScript String split() Method</title>

```javascript
var str = "Apples are round, and apples are juicy.";
var splitted = str.split(" ", 3);
document.write(splitted);
```

**Output**

Apples, are, round,

---

**substr()**

This method returns the characters in a string beginning at the specified location through the specified number of characters.

**Syntax**

The syntax to use substr() is as follows:

```javascript
string.substr(start[, length]);
```

**Argument Details**

- **start**: Location at which to start extracting characters (an integer between 0 and one less than the length of the string).
- **length**: The number of characters to extract.

**Note**: If start is negative, substr uses it as a character index from the end of the string.

**Return Value**

The substr() method returns the new sub-string based on given parameters.

**Example**
Try the following example.

```
<html>
<head>
<title>JavaScript String substr() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Apples are round, and apples are juicy.";
    document.write("(1,2): " + str.substr(1,2));
    document.write("<br />(-2,2): " + str.substr(-2,2));
    document.write("<br />(1): " + str.substr(1));
    document.write("<br />(-20, 2): " + str.substr(-20,2));
    document.write("<br />(20, 2): " + str.substr(20,2));
</script>
</body>
</html>
```

**Output**

(1,2): pp
(-2,2): y.
(1): pples are round, and apples are juicy.
(-20, 2): nd
(20, 2): d

**substring ()**

This method returns a subset of a String object.

**Syntax**

The syntax to use substr() is as follows:

```
string.substring(indexA, [indexB])
```

**Argument Details**
- **indexA**: An integer between 0 and one less than the length of the string.
- **indexB**: (optional) An integer between 0 and the length of the string.

**Return Value**
The substring method returns the new sub-string based on given parameters.

**Example**
Try the following example.

```html
<html>
<head>
<title>JavaScript String substring() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Apples are round, and apples are juicy.";
    document.write("(1,2): " + str.substring(1,2));
    document.write("(0,10): " + str.substring(0, 10));
    document.write("(5): " + str.substring(5));
</script>
</body>
</html>
```

**Output**

(1,2): p
(0,10): Apples are
(5): s are round, and apples are juicy.

**toLocaleLowerCase()**
This method is used to convert the characters within a string to lowercase while respecting the current locale. For most languages, it returns the same output as `toLowerCase`.

**Syntax**
Its syntax is as follows:

```javascript
string.toLocaleLowerCase()
```
**Return Value**

Returns a string in lowercase with the current locale.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript String toLocaleLowerCase() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Apples are round, and Apples are Juicy.";
    document.write(str.toLocaleLowerCase( ));
</script>
</body>
</html>
```

**Output**

```
apples are round, and apples are juicy.
```

### toLocaleUpperCase()

This method is used to convert the characters within a string to uppercase while respecting the current locale. For most languages, it returns the same output as **toUpperCase**.

**Syntax**

Its syntax is as follows:

```
string.toLocaleUpperCase( )
```

**Return Value**

Returns a string in uppercase with the current locale.
Example
Try the following example.

```html
<html>
<head>
<title>JavaScript String toLocaleUpperCase() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Apples are round, and Apples are Juicy.";
    document.write(str.toLocaleUpperCase( ));
</script>
</body>
</html>
```

Output
APPLES ARE ROUND, AND APPLES ARE JUICY.

toLowerCase ()
This method returns the calling string value converted to lowercase.

Syntax
Its syntax is as follows:

```
string.toLocaleLowerCase( )
```

Return Value
Returns the calling string value converted to lowercase.

Example
Try the following example.

```html
<html>
```
<head>
<title>JavaScript String toLowerCase() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Apples are round, and Apples are Juicy.";
    document.write(str.toLowerCase( ));
</script>
</body>
</html>

Output
apples are round, and apples are juicy.

toString ()
This method returns a string representing the specified object.

Syntax
Its syntax is as follows:

string.toString( )

Return Value
Returns a string representing the specified object.

Example
Try the following example.

<html>
<head>
<title>JavaScript String toString() Method</title>
</head>
Output

Apples are round, and Apples are Juicy.

toUpperCase()

This method returns the calling string value converted to uppercase.

Syntax

Its syntax is as follows:

```
string.toUpperCase()
```

Return Value

Returns a string representing the specified object.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript String toUpperCase() Method</title>
</head>
<body>

```
<script type="text/javascript">
    var str = "Apples are round, and Apples are Juicy.";
    document.write(str.toUpperCase( ));
</script>
</body>
</html>

Output

APPLES ARE ROUND, AND APPLES ARE JUICY.

valueOf ()

This method returns the primitive value of a String object.

Syntax

Its syntax is as follows:

```
string.valueOf()
```

Return Value

Returns the primitive value of a String object.

Example

Try the following example.

```html
<html>
<head>
<title>JavaScript String valueOf() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    document.write(str.valueOf( ));
</script>
</body>
</html>
```
### String HTML Wrappers

Here is a list of the methods that return a copy of the string wrapped inside an appropriate HTML tag.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anchor()</td>
<td>Creates an HTML anchor that is used as a hypertext target.</td>
</tr>
<tr>
<td>big()</td>
<td>Creates a string to be displayed in a big font as if it were in a <code>&lt;big&gt;</code> tag.</td>
</tr>
<tr>
<td>blink()</td>
<td>Creates a string to blink as if it were in a <code>&lt;blink&gt;</code> tag.</td>
</tr>
<tr>
<td>bold()</td>
<td>Creates a string to be displayed as bold as if it were in a <code>&lt;b&gt;</code> tag.</td>
</tr>
<tr>
<td>fixed()</td>
<td>Causes a string to be displayed in fixed-pitch font as if it were in a <code>&lt;tt&gt;</code> tag.</td>
</tr>
<tr>
<td>fontcolor()</td>
<td>Causes a string to be displayed in the specified color as if it were in a <code>&lt;font color=&quot;color&quot;&gt;</code> tag.</td>
</tr>
<tr>
<td>fontsize()</td>
<td>Causes a string to be displayed in the specified font size as if it were in a <code>&lt;font size=&quot;size&quot;&gt;</code> tag.</td>
</tr>
<tr>
<td>italics()</td>
<td>Causes a string to be italic, as if it were in an <code>&lt;i&gt;</code> tag.</td>
</tr>
<tr>
<td>link()</td>
<td>Creates an HTML hypertext link that requests another URL.</td>
</tr>
<tr>
<td>small()</td>
<td>Causes a string to be displayed in a small font, as if it were in a <code>&lt;small&gt;</code> tag.</td>
</tr>
<tr>
<td>strike()</td>
<td>Causes a string to be displayed as struck-out text, as if it were in a <code>&lt;strike&gt;</code> tag.</td>
</tr>
<tr>
<td>sub()</td>
<td>Causes a string to be displayed as a subscript, as if it were in a <code>&lt;sub&gt;</code> tag.</td>
</tr>
</tbody>
</table>
javascript

<table>
<thead>
<tr>
<th>were in a <code>&lt;sub&gt;</code> tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>sup()</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of string HTML wrappers.

**anchor()**

This method creates an HTML anchor that is used as a hypertext target.

**Syntax**

Its syntax is as follows:

```
string.anchor( anchornam e )
```

**Attribute details**

**anchornam e**: Defines a name for the anchor.

**Return Value**

Returns the string having the anchor tag.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript String anchor() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    alert (str.anchor( "myanchor" ));
</script>
</body>
</html>
```
### big()

This method causes a string to be displayed in a big font as if it were in a BIG tag.

#### Syntax

The syntax to use `big()` is as follows:

```javascript
string.big()
```

#### Return Value

Returns the string having `<big>` tag.

#### Example

Try the following example.

```html
<html>
<head>
<title>JavaScript String big() Method</title>
</head>
<body>
<script type="text/javascript">
var str = new String("Hello world");
alert(str.big());
</script>
</body>
</html>
```

**Output**

`<big>Hello world</big>`
**blink()**

This method causes a string to blink as if it were in a `<blink>` tag.

**Syntax**

The syntax for `blink()` method is as follows:

```javascript
string.blink()
```

**Return Value**

Returns the string having `<blink>` tag.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript String blink() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    alert(str.blink());
</script>
</body>
</html>
```

**Output**

`<blink>Hello world</blink>`

---

**bold()**

This method causes a string to be displayed as bold as if it were in a `<b>` tag.

**Syntax**

The syntax for `bold()` method is as follows:
### Javascript

#### string.bold( )

**Return Value**

Returns the string having `<bold>` tag.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript String bold() Method</title>
</head>
<body>
<script type="text/javascript">
var str = new String("Hello world");
alert(str.bold());
</script>
</body>
</html>
```

**Output**

`<b>Hello world</b>`

---

#### fixed ()

This method causes a string to be displayed in fixed-pitch font as if it were in a `<tt>` tag.

**Syntax**

Its syntax is as follows:

```
string.fixed( )
```

**Return Value**

Returns the string having `<tt>` tag.

**Example**
Try the following example.

```html
<html>
<head>
<title>JavaScript String fixed() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    alert(str.fixed());
</script>
</body>
</html>
```

**Output**

```tt
Hello world
```

### fontColor()

This method causes a string to be displayed in the specified color as if it were in a `<font color="color">` tag.

**Syntax**

Its syntax is as follows:

```javascript
string.fontColor( color)
```

**Attribute Details**

- **color**: A string expressing the color as a hexadecimal RGB triplet or as a string literal.

**Return Value**

Returns the string with `<font color="color">` tag.

**Example**

Try the following example.
<head>
<title>JavaScript String fontcolor() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    alert(str.fontcolor( "red" ));
</script>
</body>
</html>

Output

<font color="red">Hello world</font>

**fontsize()**

This method causes a string to be displayed in the specified size as if it were in a `<font size="size">` tag.

**Syntax**

Its syntax is as follows:

```
string.fontsize( size )
```

**Attribute Details**

**size:** An integer between 1 and 7, a string representing a signed integer between 1 and 7.

**Return Value**

Returns the string with `<font size="size">` tag.

**Example**

Try the following example.
<html>
<head>
<title>JavaScript String fontsize() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    alert(str.fontsize( 3 ));
</script>
</body>
</html>

Output

<font size="3">Hello world</font>

**italics ()**

This method causes a string to be italic, as if it were in an `<i>` tag.

**Syntax**

Its syntax is as follows:

```
string.italics ( )
```

**Return Value**

Returns the string with `<i>` tag.

**Example**

Try the following example.

```html
```
<head>
<title>JavaScript String italics() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    alert(str.italics( ));
</script>
</body>
</html>

Output

<i>Hello world</i>

**link()**

This method creates an HTML hypertext link that requests another URL.

**Syntax**

The syntax for link() method is as follows:

```
string.link ( hrefname )
```

**Attribute Details**

**hrefname**: Any string that specifies the HREF of the A tag; it should be a valid URL.

**Return Value**

Returns the string with `<a>` tag.

**Example**

Try the following example.

```
<html>
```
<head>
<title>JavaScript String link() Method</title>
</head>
<body>
<script type="text/javascript">
var str = new String("Hello world");
var URL = "http://www.tutorialspoint.com";

alert(str.link( URL ));
</script>
</body>
</html>

Output

<a href = “http://www.tutorialspoint.com”>Hello world</a>

**small ()**

This method causes a string to be displayed in a small font, as if it were in a `<small>` tag.

**Syntax**

Its syntax is as follows:

```
string.small ( )
```

**Return Value**

Returns the string with `<small>` tag.

**Example**

Try the following example.

```html

```
<script type="text/javascript">
    var str = new String("Hello world");
    alert(str.small());
</script>

**Output**

<small>Hello world</small>

### strike ()

This method causes a string to be displayed as struck-out text, as if it were in a `<strike>` tag.

**Syntax**

Its syntax is as follows:

```
string.strike ( )
```

**Return Value**

Returns the string with `<strike>` tag.

**Example**

Try the following example.

```html
<html>
```
<head>
<title>JavaScript String strike() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    alert(str.strike());
</script>
</body>
</html>

Output

<strike>Hello world</strike>

sub()

This method causes a string to be displayed as a subscript, as if it were in a <sub> tag.

Syntax

Its syntax is as follows:

```
string.sub()
```

Return Value

Returns the string with <sub> tag.

Example

Try the following example.
<html>
<head>
<title>JavaScript String sub() Method</title>
</head>
<body>
<script type="text/javascript">
  var str = new String("Hello world");
  alert(str.sub());
</script>
</body>
</html>

Output

<sub>Hello world</sub>

sup() 

This method causes a string to be displayed as a superscript, as if it were in a <sup> tag.

Syntax

Its syntax is as follows:

string.sup()

Return Value

 Returns the string with <sup> tag.

Example

Try the following example.
<html>
<head>
<title>JavaScript String sup() Method</title>
</head>
<body>
<script type="text/javascript">
    var str = new String("Hello world");
    alert(str.sup());
</script>
</body>
</html>

Output

<sup>Hello world</sup>
The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

**Syntax**

Use the following syntax to create an **Array** Object.

```javascript
var fruits = new Array( "apple", "orange", "mango" );
```

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create an array by simply assigning values as follows:

```javascript
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

- `fruits[0]` is the first element
- `fruits[1]` is the second element
- `fruits[2]` is the third element

**Array Properties**

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>Returns a reference to the array function that created the object.</td>
</tr>
<tr>
<td>index</td>
<td>The property represents the zero-based index of the match in the string</td>
</tr>
</tbody>
</table>
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.

In the following sections, we will have a few examples to illustrate the usage of Array properties.

**constructor**

Javascript array **constructor** property returns a reference to the array function that created the instance's prototype.

**Syntax**

Its syntax is as follows:

```javascript
array.constructor
```

**Return Value**

Returns the function that created this object's instance.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript Array constructor Property</title>
</head>
<body>
<script type="text/javascript">
  var arr = new Array( 10, 20, 30 );
  document.write("arr.constructor is:" + arr.constructor);
</script>
</body>
</html>
```
length
Javascript array length property returns an unsigned, 32-bit integer that specifies the number of elements in an array.

Syntax
Its syntax is as follows:

array.length

Return Value
Returns the length of an array.

Example
Try the following example.

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

Output

arr.length is:3
Prototype

The prototype property allows you to add properties and methods to any object (Number, Boolean, String, Date, etc.).

**Note:** Prototype is a global property which is available with almost all the objects.

**Syntax**
Its syntax is as follows:

```javascript
object.prototype.name = value
```

**Example**
Try the following example.

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

<script type="text/javascript">
function book(title, author){
    this.title = title;
    this.author  = author;
}
</script>

<body>
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    book.prototype.price = null;
    myBook.price = 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>
</html>
```
Array Methods

Here is a list of the methods of the Array object along with their description.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>concat()</td>
<td>Returns a new array comprised of this array joined with other array(s) and/or value(s).</td>
</tr>
<tr>
<td>every()</td>
<td>Returns true if every element in this array satisfies the provided testing function.</td>
</tr>
<tr>
<td>filter()</td>
<td>Creates a new array with all of the elements of this array for which the provided filtering function returns true.</td>
</tr>
<tr>
<td>forEach()</td>
<td>Calls a function for each element in the array.</td>
</tr>
<tr>
<td>indexOf()</td>
<td>Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.</td>
</tr>
<tr>
<td>join()</td>
<td>Joins all elements of an array into a string.</td>
</tr>
<tr>
<td>lastIndexOf()</td>
<td>Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.</td>
</tr>
<tr>
<td>map()</td>
<td>Creates a new array with the results of calling a provided function on every element in this array.</td>
</tr>
<tr>
<td>pop()</td>
<td>Removes the last element from an array and returns that element.</td>
</tr>
</tbody>
</table>
In the following sections, we will have a few examples to demonstrate the usage of Array methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>push()</td>
<td>Adds one or more elements to the end of an array and returns the new length of the array.</td>
</tr>
<tr>
<td>reduce()</td>
<td>Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.</td>
</tr>
<tr>
<td>reduceRight()</td>
<td>Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.</td>
</tr>
<tr>
<td>reverse()</td>
<td>Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.</td>
</tr>
<tr>
<td>shift()</td>
<td>Removes the first element from an array and returns that element.</td>
</tr>
<tr>
<td>slice()</td>
<td>Extracts a section of an array and returns a new array.</td>
</tr>
<tr>
<td>some()</td>
<td>Returns true if at least one element in this array satisfies the provided testing function.</td>
</tr>
<tr>
<td>toSource()</td>
<td>Represents the source code of an object</td>
</tr>
<tr>
<td>sort()</td>
<td>Sorts the elements of an array.</td>
</tr>
<tr>
<td>splice()</td>
<td>Adds and/or removes elements from an array.</td>
</tr>
<tr>
<td>toString()</td>
<td>Returns a string representing the array and its elements.</td>
</tr>
<tr>
<td>unshift()</td>
<td>Adds one or more elements to the front of an array and returns the new length of the array.</td>
</tr>
</tbody>
</table>
### concat()

Javascript array `concat()` method returns a new array comprised of this array joined with two or more arrays.

#### Syntax

The syntax of `concat()` method is as follows:

```javascript
array.concat(value1, value2, ..., valueN);
```

#### Parameter Details

- `valueN`: Arrays and/or values to concatenate to the resulting array.

#### Return Value

Returns the length of the array.

#### Example

Try the following example.

```html
<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>
</html>
```

#### Output

```
alphaNumeric : a,b,c,1,2,3
```
every()

Javascript array every method tests whether all the elements in an array passes the test implemented by the provided function.

Syntax
Its syntax is as follows:

```javascript
array.every(callback[, thisObject]);
```

Parameter Details
- **callback**: Function to test for each element.
- **thisObject**: Object to use as this when executing callback.

Return Value
Returns true if every element in this array satisfies the provided testing function.

Compatibility
This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add the following code at the top of your script.

```javascript
if (!Array.prototype.every)
{
    Array.prototype.every = function(fun /*, thisp*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        var thisp = arguments[1];
        for (var i = 0; i < len; i++)
        {
            if (i in this &&
                !fun.call(thisp, this[i], i, this))
                return false;
        }
    }
}
Example
Try the following example.

```html
<html>
<head>
<title>JavaScript Array every Method</title>
</head>
<body>
<script type="text/javascript">
if (!Array.prototype.every)
{
    Array.prototype.every = function(fun /*, thisp*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        var thisp = arguments[1];
        for (var i = 0; i < len; i++)
        {
            if (i in this &&
                !fun.call(thisp, this[i], i, this))
                return false;
        }

        return true;
    }
}

function isBigEnough(element, index, array) {
    return (element >= 10);
}
```
```javascript
var passed = [12, 5, 8, 130, 44].every(isBigEnough);
document.write("First Test Value : " + passed);

passed = [12, 54, 18, 130, 44].every(isBigEnough);
document.write("Second Test Value : " + passed);
</script>
</body>
</html>

Output

First Test Value : false
Second Test Value : true

filter()

Javascript array `filter()` method creates a new array with all elements that pass the test implemented by the provided function.

Syntax

Its syntax is as follows:

```javascript
array.filter(callback[, thisObject]);
```

Parameter Details

- `callback` : Function to test for each element of an array.
- `thisObject` : Object to use as `this` when executing callback.

Return Value

Returns created array.

Compatibility

This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add the following code at the top of your script.

```javascript
if (!Array.prototype.filter)
```
```javascript
Array.prototype.filter = function(fun /*, thisp*/)
{
    var len = this.length;
    if (typeof fun != "function")
        throw new TypeError();

    var res = new Array();
    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
    {
        if (i in this)
        {
            var val = this[i]; // in case fun mutates this
            if (fun.call(thisp, val, i, this))
                res.push(val);
        }
    }

    return res;
};
```

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript Array filter Method</title>
</head>
<body>
<script type="text/javascript">
if (!Array.prototype.filter)
{
```
Array.prototype.filter = function(fun /*, thisp*/) {
    var len = this.length;
    if (typeof fun != "function")
        throw new TypeError();

    var res = new Array();
    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
    {
        if (i in this)
        {
            var val = this[i]; // in case fun mutates this
            if (fun.call(thisp, val, i, this))
                res.push(val);
        }
    }

    return res;
};

function isBigEnough(element, index, array) {
    return (element >= 10);
}

var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
document.write("Filtered Value : " + filtered);

</script>
</body>
</html>
ForEach ()

Javascript array `forEach()` method calls a function for each element in the array.

Syntax

Its syntax is as follows:

```
array.forEach(callback[, thisObject]);
```

Parameter Details

- **callback**: Function to test for each element of an array.
- **thisObject**: Object to use as this when executing callback.

Return Value

Returns the created array.

Compatibility

This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add following code at the top of your script.

```javascript
if (!Array.prototype.forEach)
{
    Array.prototype.forEach = function(fun /*, thisp*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        var thisp = arguments[1];
        for (var i = 0; i < len; i++)
        {
            if (i in this)
                fun.call(thisp, this[i], i, this);
        }
    }
}
```
Example
Try the following example.

```html
<html>
<head>
<title>JavaScript Array forEach Method</title>
</head>
<body>
<script type="text/javascript">
if (!Array.prototype.forEach)
{
    Array.prototype.forEach = function(fun /*, thisp*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        var thisp = arguments[1];
        for (var i = 0; i < len; i++)
        {
            if (i in this)
                fun.call(thisp, this[i], i, this);
        }
    }
}

function printBr(element, index, array) {
    document.write("<br />
[" + index + "] is " + element);
}

[12, 5, 8, 130, 44].forEach(printBr);
```
indexOf ()

Javascript array `indexOf()` method returns the first index at which a given element can be found in the array, or -1 if it is not present.

Syntax

Its syntax is as follows:

```
array.indexOf(searchElement[, fromIndex]);
```

Parameter Details

- `searchElement` : Element to locate in the array.
- `fromIndex` : The index at which to begin the search. Defaults to 0, i.e. the whole array will be searched. If the index is greater than or equal to the length of the array, -1 is returned.

Return Value

Returns the index of the found element.

Compatibility

This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add the following code at the top of your script.

```
if (!Array.prototype.indexOf)
{
    Array.prototype.indexOf = function(elt /*, from*/) {

```
```javascript
var len = this.length;

var from = Number(arguments[1]) || 0;
from = (from < 0) ? Math.ceil(from) : Math.floor(from);
if (from < 0)
  from += len;

for (; from < len; from++)
{
  if (from in this &&
      this[from] === elt)
    return from;
}
return -1;
```

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript Array indexOf Method</title>
</head>
<body>
<script type="text/javascript">
if (!Array.prototype.indexOf)
{
  Array.prototype.indexOf = function(elt /*, from*/) {
  var len = this.length;
```
```javascript
var from = Number(arguments[1]) || 0;
from = (from < 0)
    ? Math.ceil(from)
    : Math.floor(from);
if (from < 0)
    from += len;

for (; from < len; from++)
{
    if (from in this &&
        this[from] === elt)
        return from;
}
return -1;
}

function printBr(element, index, array) {
    document.write("<br />[" + index + "] is " + element );
}

var index = [12, 5, 8, 130, 44].indexOf(8);
document.write("index is : " + index );

var index = [12, 5, 8, 130, 44].indexOf(13);
document.write("<br />index is : " + index );
</script>
</body>
</html>

Output

index is : 2
index is : -1
**join()**

Javascript array `join()` method joins all the elements of an array into a string.

**Syntax**

Its syntax is as follows:

```javascript
array.join(separator);
```

**Parameter Details**

- **separator**: Specifies a string to separate each element of the array. If omitted, the array elements are separated with a comma.

**Return Value**

Returns a string after joining all the array elements.

**Example**

Try the following example.

```html
<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("<br />str : " + str);

var str = arr.join(" + ");
document.write("<br />str : " + str);
</script>
</body>
</html>
```
Output

```javascript
str : First,Second,Third
str : First, Second, Third
str : First + Second + Third
```

**lastIndexOf()**

Javascript array `lastIndexOf()` method returns the last index at which a given element can be found in the array, or -1 if it is not present. The array is searched backwards, starting at `fromIndex`.

**Syntax**

Its syntax is as follows:

```javascript
array.join(separator);
```

**Parameter Details**

- `searchElement`: Element to locate in the array.
- `fromIndex`: The index at which to start searching backwards. Defaults to the array's length, i.e., the whole array will be searched. If the index is greater than or equal to the length of the array, the whole array will be searched. If negative, it is taken as the offset from the end of the array.

**Return Value**

Returns the index of the found element from the last.

**Compatibility**

This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add the following code at the top of your script.

```javascript
if (!Array.prototype.lastIndexOf)
{
    Array.prototype.lastIndexOf = function(elt /*, from*/) {
    var len = this.length;
```
var from = Number(arguments[1]);
if (isNaN(from))
{
    from = len - 1;
}
else
{
    from = (from < 0)
        ? Math.ceil(from)
        : Math.floor(from);
    if (from < 0)
        from += len;
    else if (from >= len)
        from = len - 1;
}

for (; from > -1; from--)
{
    if (from in this &&
        this[from] === elt)
        return from;
}
return -1;

Example
Try the following example.
<html>
<head>
<title>JavaScript Array lastIndexOf Method</title>
</head>
<body>
if (!Array.prototype.lastIndexOf)
{
    Array.prototype.lastIndexOf = function(elt /*, from*/)
    {
        var len = this.length;

        var from = Number(arguments[1]);
        if (isNaN(from))
        {
            from = len - 1;
        }
        else
        {
            from = (from < 0)
                ? Math.ceil(from)
                : Math.floor(from);
            if (from < 0)
                from += len;
            else if (from >= len)
                from = len - 1;
        }

        for (; from > -1; from--)
        {
            if (from in this &&
                this[from] === elt)
                return from;
        }
        return -1;
    };
```javascript
var index = [12, 5, 8, 130, 44].lastIndexOf(8);
document.write("index is : " + index);

var index = [12, 5, 8, 130, 44, 5].lastIndexOf(5);
document.write("<br />index is : " + index);
</script>
</body>
</html>

Output

index is : 2
index is : 5

map()

Javascript array **map()** method creates a new array with the results of calling a provided function on every element in this array.

Syntax

Its syntax is as follows:

```javascript
array.map(callback[, thisObject]);
```

Parameter Details

- **callback**: Function that produces an element of the new Array from an element of the current one.
- **thisObject**: Object to use as **this** when executing callback.

Return Value

Returns the created array.

Compatibility
This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add the following code at the top of your script.

```javascript
if (!Array.prototype.map)
{
    Array.prototype.map = function(fun /*, thisp*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        var res = new Array(len);
        var thisp = arguments[1];
        for (var i = 0; i < len; i++)
        {
            if (i in this)
                res[i] = fun.call(thisp, this[i], i, this);
        }

        return res;
    }
}
```

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript Array map Method</title>
</head>
<body>
<script type="text/javascript">
if (!Array.prototype.map)
{
```
Array.prototype.map = function(fun /*, thisp*/)
{
    var len = this.length;
    if (typeof fun != "function")
        throw new TypeError();

    var res = new Array(len);
    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
    {
        if (i in this)
            res[i] = fun.call(thisp, this[i], i, this);
    }

    return res;
};

var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);

document.write("roots is : " + roots );

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

Output

roots is : 1,2,3

pop ()

Javascript array pop() method removes the last element from an array and returns that element.
**Syntax**
Its syntax is as follows:

```javascript
Array.pop();
```

**Return Value**
Returns the removed element from the array.

**Example**
Try the following example.

```html
<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("<br />element is : " + element);
</script>
</body>
</html>
```

**Output**

```
element is : 9
element is : 4
```

**push()**
Javascript array **push()** method appends the given element(s) in the last of the array and returns the length of the new array.

**Syntax**
Its syntax is as follows:

```
Array.push();
```

**Parameter Details**

*element1, ..., elementN:* The elements to add to the end of the array.

**Return Value**

Returns the length of the new array.

**Example**

Try the following example.

```html
<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(<br />new numbers is : " + numbers);
</script>
</body>
</html>
```

**Output**

```
new numbers is : 1,4,9,10
new numbers is : 1,4,9,10,20
```
reduce ()

Javascript array **reduce()** method applies a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.

**Syntax**

Its syntax is as follows:

```
array.reduce(callback[, initialValue]);
```

**Parameter Details**

- **callback** : Function to execute on each value in the array.
- **initialValue** : Object to use as the first argument to the first call of the callback.

**Return Value**

Returns the reduced single value of the array.

**Compatibility**

This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add the following code at the top of your script.

```javascript
if (!Array.prototype.reduce)
{
    Array.prototype.reduce = function(fun /*, initial*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        // no value to return if no initial value and an empty array
        if (len == 0 && arguments.length == 1)
            throw new TypeError();

        var i = 0;
        if (arguments.length >= 2)
        {
```
```javascript
var rv = arguments[1];
}
else
{
    do
    {
        if (i in this)
        {
            rv = this[i++];
            break;
        }
    // if array contains no values, no initial value to return
    if (++i >= len)
        throw new TypeError();
    } while (true);
}
for (; i < len; i++)
{
    if (i in this)
        rv = fun.call(null, rv, this[i], i, this);
}
return rv;
}

Example
Try the following example.

<html>
<head>
```
if (!Array.prototype.reduce)
{
    Array.prototype.reduce = function(fun /*, initial*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        // no value to return if no initial value and an empty array
        if (len == 0 && arguments.length == 1)
            throw new TypeError();

        var i = 0;
        if (arguments.length >= 2)
        {
            var rv = arguments[1];
        }
        else
        {
            do
            {
                if (i in this)
                {
                    rv = this[i++];
                    break;
                }

            // if array contains no values, no initial value to return
            if (++i >= len)
throw new TypeError();
}
while (true);

for (; i < len; i++)
{
    if (i in this)
        rv = fun.call(null, rv, this[i], i, this);
}

return rv;
};

var total = [0, 1, 2, 3].reduce(function(a, b){ return a + b; });
document.write("total is : " + total);
</script>
</body>
</html>

Output

| total is : 6 |

reduceRight ()

Javascript array **reduceRight()** method applies a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

**Syntax**

Its syntax is as follows:

```javascript
array.reduceRight(callback[, initialValue]);
```
• **callback** : Function to execute on each value in the array.

• **initialValue** : Object to use as the first argument to the first call of the callback.

**Return Value**
Reutrns the reduced right single value of the array.

**Compatibility**
This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add the following code at the top of your script.

```javascript
if (!Array.prototype.reduceRight)
{
    Array.prototype.reduceRight = function(fun /*, initial*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        // no value to return if no initial value, empty array
        if (len == 0 && arguments.length == 1)
            throw new TypeError();

        var i = len - 1;
        if (arguments.length >= 2)
        {
            var rv = arguments[1];
        }
        else
        {
            do
            {
                if (i in this)
                {
```
rv = this[i--];
break;
}

// if array contains no values, no initial value to return
if (--i < 0)
    throw new TypeError();
}
while (true);

for (; i >= 0; i--)
{
    if (i in this)
        rv = fun.call(null, rv, this[i], i, this);
}

return rv;
}

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript Array reduceRight Method</title>
</head>
<body>
<script type="text/javascript">
if (!Array.prototype.reduceRight)
{
    Array.prototype.reduceRight = function(fun /*, initial*/)
    {
```
```javascript
var len = this.length;
if (typeof fun != "function")
    throw new TypeError();

// no value to return if no initial value, empty array
if (len == 0 && arguments.length == 1)
    throw new TypeError();

var i = len - 1;
if (arguments.length >= 2)
{
    var rv = arguments[1];
}
else
{
    do
    {
        if (i in this)
        {
            rv = this[i--];
            break;
        }
    }
    while (true);
}

for (; i >= 0; i--)
{
    if (i in this)
```
rv = fun.call(null, rv, this[i], i, this);
}

return rv;
};
}

var total = [0, 1, 2, 3].reduceRight(function(a, b){ return a + b; });
document.write("total is : " + total);
</script>
</body>
</html>

Output
total is : 6

reverse ()
Javascript array reverse() method reverses the element of an array. The first array element becomes the last and the last becomes the first.

Syntax
Its syntax is as follows:

array.reverse();

Return Value
Returns the reversed single value of the array.

Example
Try the following example.

<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

**shift()**

JavaScipt array **shift()** method removes the first element from an array and returns that element.

**Syntax**

Its syntax is as follows:

array.shift();

**Return Value**

Returns the removed single value of the array.

**Example**

Try the following example.

```html
<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>
```
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] );
```

**Parameter Details**

- **begin**: Zero-based index at which to begin extraction. As a negative index, start indicates an offset from the end of the sequence.
- **end**: Zero-based index at which to end extraction.

**Return Value**

Returns the extracted array based on the passed parameters.

**Example**

Try the following example.

```html
<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("<br />
    arr.slice( 1, 2) : " + arr.slice( 1, 3));
</script>
</body>
</html>
```
Output

arr.slice(1, 2) : mango
arr.slice(1, 2) : mango,banana

some()

Javascript array some() method tests whether some element in the array passes the test implemented by the provided function.

Syntax

Its syntax is as follows:

```javascript
array.some(callback[, thisObject]);
```

Parameter Details

- **callback**: Function to test for each element.
- **thisObject**: Object to use as this when executing callback.

Return Value

If some element pass the test, then it returns true, otherwise false.

Compatibility

This method is a JavaScript extension to the ECMA-262 standard; as such it may not be present in other implementations of the standard. To make it work, you need to add the following code at the top of your script.

```javascript
if (!Array.prototype.some)
{
  Array.prototype.some = function(fun /*, thisp*/)
  {
    var len = this.length;
    if (typeof fun != "function")
      throw new TypeError();

    var thisp = arguments[1];
    for (var i = 0; i < len; i++)
      ```
Example
Try the following example.

```html
<html>
<head>
<title>JavaScript Array some Method</title>
</head>
<body>
<script type="text/javascript">
if (!Array.prototype.some)
{
    Array.prototype.some = function(fun /*, thisp*/)
    {
        var len = this.length;
        if (typeof fun != "function")
            throw new TypeError();

        var thisp = arguments[1];
        for (var i = 0; i < len; i++)
        {
            if (i in this &&
                fun.call(thisp, this[i], i, this))
                return true;
        }
        return false;
    }
}
```
return false;
};

function isBigEnough(element, index, array) {
    return (element >= 10);
}

var retval = [2, 5, 8, 1, 4].some(isBigEnough);
document.write("Returned value is : " + retval);

var retval = [12, 5, 8, 1, 4].some(isBigEnough);
document.write("<br />Returned value is : " + retval);
</script>
</body>
</html>

Output

Returned value is : false
Returned value is : true

**sort()**

Javascript array `sort()` method sorts the elements of an array.

**Syntax**

Its syntax is as follows:

```
array.sort( compareFunction );
```

**Parameter Details**

**compareFunction**: Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically.
Return Value
Returns a sorted array.

Example
Try the following example.

```html
<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

splice ()
Javascript array splice() method changes the content of an array, adding new elements while removing old elements.

Syntax
Its syntax is as follows:

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

Parameter Details
- `index`: Index at which to start changing the array.
- **howMany**: An integer indicating the number of old array elements to remove. If **howMany** is 0, no elements are removed.

- **element1, ..., elementN**: The elements to add to the array. If you don't specify any elements, splice simply removes the elements from the array.

**Return Value**

Returns the extracted array based on the passed parameters.

**Example**

Try the following example.

```html
<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 1: " + arr);
    document.write("<br />removed is: " + removed);

    removed = arr.splice(3, 1);
    document.write("<br />After adding 1: " + arr);
    document.write("<br />removed is: " + removed);

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

**Output**

After adding 1: orange,mango,water,banana,sugar,tea
removed is:
After adding 1: orange,mango,water,sugar,tea
removed is: banana
**toString ()**

Javascript array **toString()** method returns a string representing the source code of the specified array and its elements.

**Syntax**

Its syntax is as follows:

```javascript
array.toString();
```

**Return Value**

Returns a string representing the array.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript Array toString Method</title>
</head>
<body>
<script type="text/javascript">
    var arr = new Array("orange", "mango", "banana", "sugar");

    var str = arr.toString();
    document.write("Returned string is : " + str);
</script>
</body>
</html>
```

**Output**

```
Returned string is : orange,mango,banana,sugar
```

**unshift ()**

Javascript array **unshift()** method adds one or more elements to the beginning of an array and returns the new length of the array.
Syntax
Its syntax is as follows:
array.unshift( element1, ..., elementN );

Parameter Details
**element1, ..., elementN** : The elements to add to the front of the array.

Return Value
Returns the length of the new array. It returns **undefined** in IE browser.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript Array unshift Method</title>
</head>
<body>
<script type="text/javascript">
    var arr = new Array("orange", "mango", "banana", "sugar");

    var length = arr.unshift("water");
    document.write("Returned array is : " + arr);
    document.write("<br /> Length of the array is : " + length);
</script>
</body>
</html>
```

Output
Returned array is : water, orange, mango, banana, sugar
Length of the array is : 5
The Date object is a datatype built into the JavaScript language. Date objects are created with the `new Date()` as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

The ECMAScript standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so JavaScript can represent date and time till the year 275755.

**Syntax**

You can use any of the following syntaxes to create a Date object using `Date()` constructor.

```plaintext
new Date( )
new Date(milliseconds)
new Date(datestring)
new Date(year,month,date[,hour,minute,second,millisecond ])
```

**Note:** Parameters in the brackets are always optional.

Here is a description of the parameters:

- **No Argument:** With no arguments, the `Date()` constructor creates a Date object set to the current date and time.

- **milliseconds:** When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the `getTime()` method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.

- **datestring:** When one string argument is passed, it is a string representation of a date, in the format accepted by the `Date.parse()` method.

- **7 arguments:** To use the last form of the constructor shown above. Here is a description of each argument:
- **year**: Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.

- **month**: Integer value representing the month, beginning with 0 for January to 11 for December.

- **date**: Integer value representing the day of the month.

- **hour**: Integer value representing the hour of the day (24-hour scale).

- **minute**: Integer value representing the minute segment of a time reading.

- **second**: Integer value representing the second segment of a time reading.

- **millisecond**: Integer value representing the millisecond segment of a time reading.

### Date Properties

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>Specifies the function that creates an object's prototype.</td>
</tr>
<tr>
<td>prototype</td>
<td>The prototype property allows you to add properties and methods to an object.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of different Date properties.

### constructor

Javascript date **constructor** property returns a reference to the array function that created the instance's prototype.

**Syntax**

Its syntax is as follows:

```javascript
date.constructor
```
**Return Value**

Returns the function that created this object’s instance.

**Example**

Try the following example.

```html
<html>
  <head>
    <title>JavaScript Date constructor Property</title>
  </head>
  <body>
    <script type="text/javascript">
      var dt = new Date();
      document.write("dt.constructor is : " + dt.constructor);
    </script>
  </body>
</html>
```

**Output**

```
dt.constructor is : function Date() { [native code] }
```

**Prototype**

The prototype property allows you to add properties and methods to any object (Number, Boolean, String, Date, etc.).

**Note:** Prototype is a global property which is available with almost all the objects.

**Syntax**

Its syntax is as follows:

```
object.prototype.name = value
```
Example
Try the following example.

```html
<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("Perl", "Mohtashim");
    book.prototype.price = null;
    myBook.price = 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>
</html>
```

Output
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
# Date Methods

Here is a list of the methods used with `Date` and their description.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date()</td>
<td>Returns today's date and time</td>
</tr>
<tr>
<td>getDate()</td>
<td>Returns the day of the month for the specified date according to local time.</td>
</tr>
<tr>
<td>getDay()</td>
<td>Returns the day of the week for the specified date according to local time.</td>
</tr>
<tr>
<td>getFullYear()</td>
<td>Returns the year of the specified date according to local time.</td>
</tr>
<tr>
<td>getHours()</td>
<td>Returns the hour in the specified date according to local time.</td>
</tr>
<tr>
<td>getMilliseconds()</td>
<td>Returns the milliseconds in the specified date according to local time.</td>
</tr>
<tr>
<td>getMinutes()</td>
<td>Returns the minutes in the specified date according to local time.</td>
</tr>
<tr>
<td>getMonth()</td>
<td>Returns the month in the specified date according to local time.</td>
</tr>
<tr>
<td>getSeconds()</td>
<td>Returns the seconds in the specified date according to local time.</td>
</tr>
<tr>
<td>getTime()</td>
<td>Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.</td>
</tr>
<tr>
<td>getTimezoneOffset()</td>
<td>Returns the time-zone offset in minutes for the current locale.</td>
</tr>
<tr>
<td>getUTCDate()</td>
<td>Returns the day (date) of the month in the specified date according to universal time.</td>
</tr>
<tr>
<td>getUTCDay()</td>
<td>Returns the day of the week in the specified date according to universal time.</td>
</tr>
<tr>
<td>getUTCFullYear()</td>
<td>Returns the year in the specified date according to universal time.</td>
</tr>
<tr>
<td>getUTCHours()</td>
<td>Returns the hours in the specified date according to universal time.</td>
</tr>
<tr>
<td>getUTCMilliseconds()</td>
<td>Returns the milliseconds in the specified date</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getUTCMinutes()</td>
<td>Returns the minutes in the specified date according to universal time.</td>
</tr>
<tr>
<td>getUTCMonth()</td>
<td>Returns the month in the specified date according to universal time.</td>
</tr>
<tr>
<td>getUTCSeconds()</td>
<td>Returns the seconds in the specified date according to universal time.</td>
</tr>
<tr>
<td>getYear()</td>
<td><strong>Deprecated</strong> - Returns the year in the specified date according to local time. Use getFullYear instead.</td>
</tr>
<tr>
<td>setDate()</td>
<td>Sets the day of the month for a specified date according to local time.</td>
</tr>
<tr>
<td>setFullYear()</td>
<td>Sets the full year for a specified date according to local time.</td>
</tr>
<tr>
<td>setHours()</td>
<td>Sets the hours for a specified date according to local time.</td>
</tr>
<tr>
<td>setMilliseconds()</td>
<td>Sets the milliseconds for a specified date according to local time.</td>
</tr>
<tr>
<td>setMinutes()</td>
<td>Sets the minutes for a specified date according to local time.</td>
</tr>
<tr>
<td>setMonth()</td>
<td>Sets the month for a specified date according to local time.</td>
</tr>
<tr>
<td>setSeconds()</td>
<td>Sets the seconds for a specified date according to local time.</td>
</tr>
<tr>
<td>setTime()</td>
<td>Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.</td>
</tr>
<tr>
<td>setUTCDate()</td>
<td>Sets the day of the month for a specified date according to universal time.</td>
</tr>
<tr>
<td>setUTCFullYear()</td>
<td>Sets the full year for a specified date according to universal time.</td>
</tr>
<tr>
<td>setUTCHours()</td>
<td>Sets the hour for a specified date according to universal time.</td>
</tr>
<tr>
<td>setUTCMilliseconds()</td>
<td>Sets the milliseconds for a specified date according to universal time.</td>
</tr>
<tr>
<td>setUTCMinutes()</td>
<td>Sets the minutes for a specified date according to universal time.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>setUTCMonth()</td>
<td>Sets the month for a specified date according to universal time.</td>
</tr>
<tr>
<td>setUTCSeconds()</td>
<td>Sets the seconds for a specified date according to universal time.</td>
</tr>
<tr>
<td>setYear()</td>
<td><strong>Deprecated</strong> - Sets the year for a specified date according to local time. Use setFullYear instead.</td>
</tr>
<tr>
<td>toDateString()</td>
<td>Returns the &quot;date&quot; portion of the Date as a human-readable string.</td>
</tr>
<tr>
<td>toGMTString()</td>
<td><strong>Deprecated</strong> - Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.</td>
</tr>
<tr>
<td>toLocaleDateString()</td>
<td>Returns the &quot;date&quot; portion of the Date as a string, using the current locale's conventions.</td>
</tr>
<tr>
<td>toLocaleFormat()</td>
<td>Converts a date to a string, using a format string.</td>
</tr>
<tr>
<td>toLocaleString()</td>
<td>Converts a date to a string, using the current locale's conventions.</td>
</tr>
<tr>
<td>toLocaleTimeString()</td>
<td>Returns the &quot;time&quot; portion of the Date as a string, using the current locale's conventions.</td>
</tr>
<tr>
<td>toSource()</td>
<td>Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.</td>
</tr>
<tr>
<td>toString()</td>
<td>Returns a string representing the specified Date object.</td>
</tr>
<tr>
<td>toTimeString()</td>
<td>Returns the &quot;time&quot; portion of the Date as a human-readable string.</td>
</tr>
<tr>
<td>toUTCString()</td>
<td>Converts a date to a string, using the universal time convention.</td>
</tr>
<tr>
<td>valueOf()</td>
<td>Returns the primitive value of a Date object.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of Date methods.
Date()

Javascript Date() method returns today's date and time and does not need any object to be called.

Syntax

Its syntax is as follows:

```
Date()
```

Return Value

Returns today's date and time.

Example

Try the following example.

```html
<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 Mar 25 2015 15:00:57 GMT+0530 (India Standard Time)
```

gDate()

Javascript date gDate() method returns the day of the month for the specified date according to local time. The value returned by gDate is an integer between 1 and 31.
Syntax
Its syntax is as follows:

\[ \text{Date.getDate()} \]

Return Value
Returns today's date and time.

Example
Try the following example.

```html
<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());
</script>
</body>
</html>
```

Output

getDate() : 25

getDay()

Javascript date \texttt{getDay()} method returns the day of the week for the specified date according to local time. The value returned by \texttt{getDay} is an integer corresponding to the day of the week: 0 for Sunday, 1 for Monday, 2 for Tuesday, and so on.

Syntax
Its syntax is as follows:

\[ \text{Date.getDay()} \]
Return Value
Returns the day of the week for the specified date according to local time.

Example
Try the following example.

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

Output

```
getDay() : 1
```

getFullYear()

Javascript date `getFullYear()` method returns the year of the specified date according to local time. The value returned by `getFullYear` is an absolute number. For dates between the years 1000 and 9999, `getFullYear` returns a four-digit number, for example, 2008.

Syntax

Its syntax is as follows:

```
Date.getFullYear()
```

Return Value

Returns the year of the specified date according to local time.
Example
Try the following example.

```html
<html>
<head>
<title>JavaScript getFullYear Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(“December 25, 1995 23:15:00”);
    document.write(“getFullYear() : “ + dt.getFullYear() + “);”);
</script>
</body>
</html>
```

Output
getFullYear() : 1995

gETCHOURS()

Javascript Date getHours() method returns the hour in the specified date according to local time. The value returned by getHours is an integer between 0 and 23.

Syntax
Its syntax is as follows:

```javascript
Date.getHours()
```

Return Value
Returns the hour in the specified date according to local time.

Example
Try the following example.

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

**Output**

getHours() : 23

---

**getMilliseconds()**

Javascript date **getMilliseconds()** method returns the milliseconds in the specified date according to local time. The value returned by **getMilliseconds** is a number between 0 and 999.

**Syntax**

Its syntax is as follows:

```javascript
Date.getMilliseconds()
```

**Return Value**

Returns the milliseconds in the specified date according to local time.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript getMilliseconds Method</title>
</head>
</html>
```
getMilliseconds() method returns the milliseconds in the specified date according to local time. The value returned by `getMilliseconds` is a number between 0 and 29,999,999.

Syntax

```
Date.getMinutes()
```

Return Value

Returns the milliseconds in the specified date according to local time.

Example

Try the following example.

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

Javascript date `getMonth()` method returns the month in the specified date according to local time. The value returned by `getMonth` is an integer between 0 and 11. 0 corresponds to January, 1 to February, and so on.

**Syntax**

Its syntax is as follows:

```
Date.getMonth()
```

**Return Value**

Returns the Month in the specified date according to local time.

**Example**

Try the following example.

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

**Output**

```
getMinutes() : 15
```
getSeconds ()

Javascript date getSeconds() method returns the seconds in the specified date according to local time. The value returned by getSeconds is an integer between 0 and 59.

Syntax

Its syntax is as follows:

```
Date.getSeconds()
```

Return Value

Returns the seconds in the specified date according to local time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getSeconds Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "December 25, 1995 23:15:20" );
    document.write("getSeconds() : "+ dt.getSeconds());
</script>
</body>
</html>
```

Output

```
getSeconds () : 20
```

getTime ()

Javascript date getTime() method returns the numeric value corresponding to the time for the specified date according to universal time. The value returned
by the `getTime` method is the number of milliseconds since 1 January 1970 00:00:00.

You can use this method to help assign a date and time to another Date object.

**Syntax**

Its syntax is as follows:

```javascript
Date.getTime()
```

**Return Value**

Returns the numeric value corresponding to the time for the specified date according to universal time.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript getTime Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "December 25, 1995 23:15:20" );
    document.write("getTime() : " + dt.getTime() );
</script>
</body>
</html>
```

**Output**

```
getTime() : 819913520000
```

---

### getTimezoneOffset ()

Javascript date `getTimezoneOffset()` method returns the time-zone offset in minutes for the current locale. The time-zone offset is the minutes in difference, the Greenwich Mean Time (GMT) is relative to your local time.

For example, if your time zone is GMT+10, -600 will be returned. Daylight savings time prevents this value from being a constant.
**Syntax**

Its syntax is as follows:

```
Date.getTimezoneOffset()
```

**Return Value**

Returns the time-zone offset in minutes for the current locale.

**Example**

Try the following example.

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

**Output**

```
getTimezoneOffset() : -330
```

---

**getUTCDate()**

Javascript date `getUTCDate()` method returns the day of the month in the specified date according to universal time. The value returned by `getUTCDate` is an integer between 1 and 31.

**Syntax**

Its syntax is as follows:

```
Date.getUTCDate()
```
Return Value
Returns the day of the month in the specified date according to universal time.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript getUTCDate Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "December 25, 1995 23:15:20" );
    document.write("getUTCDate() : " + dt.getUTCDate() );
</script>
</body>
</html>
```

Output

```
getUTCDate() : 25
```

getUTCDay ()
Javascript date getUTCDay() method returns the day of the week in the specified date according to universal time. The value returned by getUTCDay is an integer corresponding to the day of the week: 0 for Sunday, 1 for Monday, 2 for Tuesday, and so on.

Syntax
Its syntax is as follows:

```
Date.getUTCDay ()
```

Return Value
Returns the day of the week in the specified date according to universal time.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript getUTCDay Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "December 25, 1995 23:15:20" );
    document.write("getUTCDay() : " + dt.getUTCDay() );
</script>
</body>
</html>
```

**Output**

getUTCDay() : 1

---

getUTCFullYear()

Javascript date `getUTCFullYear()` method returns the year in the specified date according to universal time. The value returned by `getUTCFullYear` is an absolute number that is compliant with year-2000, for example, 2008.

**Syntax**

Its syntax is as follows:

```javascript
Date.getUTCFullYear ()
```

**Return Value**

Returns the year in the specified date according to universal time.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript getUTCFullYear Method</title>
</head>
```
<body>
<script type="text/javascript">
    var dt = new Date("December 25, 1995 23:15:20");
    document.write("getUTCFullYear() : " + dt.getUTCFullYear());
</script>
</body>
</html>

Output

getUTCFullYear() : 1995

**getUTCHours()**

Javascript date **getUTCHours()** method returns the hours in the specified date according to universal time. The value returned by **getUTCHours** is an integer between 0 and 23.

**Syntax**

Its syntax is as follows:

Date.getUTCHours ()

**Return Value**

Returns the hours in the specified date according to universal time.

**Example**

Try the following example.

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

Javascript date **getUTCMilliseconds()** method returns the milliseconds in the specified date according to universal time. The value returned by **getUTCMilliseconds** is an integer between 0 and 999.

**Syntax**

Its syntax is as follows:

```javascript
Date.getUTCMilliseconds()
```

**Return Value**

Returns the milliseconds in the specified date according to universal time.

**Example**

Try the following example.

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

**Output**

getUTCMilliseconds() : 206
getUTCMinutes()  
Javascript date getUTCMinutes() method returns the minutes in the specified date according to universal time. The value returned by getUTCMinutes is an integer between 0 and 59.

Syntax  
Its syntax is as follows:

```
Date.getUTCMinutes()
```

Return Value  
Returns the minutes in the specified date according to universal time.

Example  
Try the following example.

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

Output  
getUTCMinutes() : 18

getUTCMonth()  
Javascript date getUTCMonth() method returns the month in the specified date according to universal time. The value returned by getUTCMonth is an integer between 0 and 11 corresponding to the month. 0 for January, 1 for February, 2 for March, and so on.
**Syntax**
Its syntax is as follows:

```
Date.getUTCMonth()
```

**Return Value**
Returns the month in the specified date according to universal time.

**Example**
Try the following example.

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

**Output**
```
getUTCMonth() : 2
```

---

**getUTCSeconds()**
Javascript date `getUTCSeconds()` method returns the seconds in the specified date according to universal time. The value returned by `getUTCSeconds` is an integer between 0 and 59.

**Syntax**
Its syntax is as follows:

```
Date.getUTCSeconds()
```

**Return Value**
Returns the month in the specified date according to universal time.
Example
Try the following example.

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

Output

getUTCSeconds() : 24

**getYear()**

Javascript date **getYear()** method returns the year in the specified date according to universal time. The **getYear** is no longer used and has been replaced by the **getFullYear** method.

The value returned by **getYear** is the current year minus 1900. JavaScript 1.2 and earlier versions return either a 2-digit or 4-digit year. For example, if the year is 2026, the value returned is 2026. So before testing this function, you need to be sure of the javascript version you are using.

**Syntax**

Its syntax is as follows:

```javascript
Date.getFullYear()
```

**Return Value**

Returns the year in the specified date according to universal time.

**Example**

Try the following example.
**JavaScript**

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

**Output**

```
getYear() : 115
```

**setDate ()**

JavaScript date **setDate()** method sets the day of the month for a specified date according to local time.

**Syntax**

Its syntax is as follows:

```
Date.setDate( dayValue )
```

**Parameter Detail**

**dayValue** : An integer from 1 to 31, representing the day of the month.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript setDate Method</title>
</head>
<body>
<script type="text/javascript">
</script>
</body>
</html>
```
var dt = new Date( "Aug 28, 2008 23:30:00" );
dt.setDate( 24 );
document.write( dt );
</script>
</body>
</html>

Output

Sun Aug 24 2008 23:30:00 GMT+0530 (India Standard Time)

**setFullYear()**

Javascript date **setFullYear()** method sets the full year for a specified date according to local time.

**Syntax**

Its syntax is as follows:

```
Date.setFullYear(yearValue[, monthValue[, dayValue]])
```

**Parameter Detail**

- **yearValue**: An integer specifying the numeric value of the year, for example, 2008.
- **monthValue**: An integer between 0 and 11 representing the months January through December.
- **dayValue**: An integer between 1 and 31 representing the day of the month. If you specify the dayValue parameter, you must also specify the monthValue.

If you do not specify the monthValue and dayValue parameters, the values returned from the getMonth and getDate methods are used.

**Example**

Try the following example.

```
<html>
```
Javascript

```html
<head>
<title>JavaScript setFullYear Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setFullYear( 2000 );
    document.write( dt );
</script>
</body>
</html>

Output

Mon Aug 28 2000 23:30:00 GMT+0530 (India Standard Time)

setHours()

Javascript date `setHours()` method sets the hours for a specified date according to local time.

Syntax

Its syntax is as follows:

```javascript
Date.setHours(hoursValue[, minutesValue[, secondsValue[, msValue]]])
```

**Note:** Parameters in the bracket are always optional.

Parameter Detail

- `hoursValue` : An integer between 0 and 23, representing the hour.
- `minutesValue` : An integer between 0 and 59, representing the minutes.
- `secondsValue` : An integer between 0 and 59, representing the seconds. If you specify the `secondsValue` parameter, you must also specify the `minutesValue`.
- `msValue` : A number between 0 and 999, representing the milliseconds. If you specify the `msValue` parameter, you must also specify the `minutesValue` and `secondsValue`.
If you do not specify the minutesValue, secondsValue, and msValue parameters, the values returned from the getUTCMinutes, getUTCSeconds, and getMilliseconds methods are used.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript setHours Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setHours( 02 );
    document.write( dt );
</script>
</body>
</html>
```

**Output**

Thu Aug 28 2008 02:30:00 GMT+0530 (India Standard Time)

**setMilliseconds()**

Javascript date **setMilliseconds()** method sets the milliseconds for a specified date according to local time.

**Syntax**

Its syntax is as follows:

```
Date.setMilliseconds(millisecondsValue)
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**

**millisecondsValue**: A number between 0 and 999, representing the milliseconds.
If you specify a number outside the expected range, the date information in the Date object is updated accordingly. For example, if you specify 1010, the number of seconds is incremented by 1, and 10 is used for the milliseconds.

**Example**
Try the following example.

```html
<html>
<head>
<title>JavaScript setMilliseconds Method</title>
</head>
<body>
<script type="text/javascript">
  var dt = new Date( "Aug 28, 2008 23:30:00" );
  dt.setMilliseconds( 1010 );
  document.write( dt );
</script>
</body>
</html>
```

**Output**

Thu Aug 28 2008 23:30:01 GMT+0530 (India Standard Time)

---

**setMinutes()**

Javascript date setMinutes() method sets the minutes for a specified date according to local time.

**Syntax**
Its syntax is as follows:

```
Date.setMinutes(minutesValue[, secondsValue[, msValue]])
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**
- **minutesValue**: An integer between 0 and 59, representing the minutes.
- **secondsValue**: An integer between 0 and 59, representing the seconds. If you specify the secondsValue parameter, you must also specify the minutesValue.

- **msValue**: A number between 0 and 999, representing the milliseconds. If you specify the msValue parameter, you must also specify the minutesValue and secondsValue.

If you do not specify the secondsValue and msValue parameters, the values returned from getSeconds and getMilliseconds methods are used.

Try the following example.

```html
<html>
<head>
<title>JavaScript setMinutes Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setMinutes( 45 );
    document.write( dt );
</script>
</body>
</html>
```

**Output**

```
Thu Aug 28 2008 23:45:00 GMT+0530 (India Standard Time)
```

**setMonth()**

Javascript date setMonth() method sets the month for a specified date according to local time.

**Syntax**

The following syntax for setMonth() Method.

```javascript
Date.setMonth(monthValue[, dayValue])
```

**Note**: Parameters in the bracket are always optional.
- **monthValue**: An integer between 0 and 11 (representing the months January through December).
- **dayValue**: An integer from 1 to 31, representing the day of the month.
- **msValue**: A number between 0 and 999, representing the milliseconds. If you specify the msValue parameter, you must also specify the minutesValue and secondsValue.

If you do not specify the dayValue parameter, the value returned from the getDate method is used. If a parameter you specify is outside of the expected range, setMonth attempts to update the date information in the Date object accordingly. For example, if you use 15 for monthValue, the year will be incremented by 1 (year + 1), and 3 will be used for month.

**Example**
Try the following example.

```html
<html>
<head>
<title>JavaScript setMonth Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setMonth( 2 );
    document.write( dt );
</script>
</body>
</html>
```

**Output**
Fri Mar 28 2008 23:30:00 GMT+0530 (India Standard Time)

---

**setSeconds()**

Javascript date **setSeconds()** method sets the seconds for a specified date according to local time.

**Syntax**
Its syntax is as follows:

```
Date.setSeconds(secondsValue[, msValue])
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**
- **secondsValue**: An integer between 0 and 59.
- **msValue**: A number between 0 and 999, representing the milliseconds.

If you do not specify the `msValue` parameter, the value returned from the `getMilliseconds` method is used. If a parameter you specify is outside of the expected range, `setSeconds` attempts to update the date information in the `Date` object accordingly. For example, if you use 100 for `secondsValue`, the minutes stored in the `Date` object will be incremented by 1, and 40 will be used for seconds.

**Example**

Try the following example.

```
<html>
<head>
<title>JavaScript setSeconds Method</title>
</head>
<body>
<script type="text/javascript">
   var dt = new Date( "Aug 28, 2008 23:30:00" );
   dt.setSeconds( 80 );
   document.write( dt );
</script>
</body>
</html>
```

**Output**

**setTime()**

Javascript date `setTime()` method sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.

**Syntax**

Its syntax is as follows:

```
Date.setTime(timeValue)
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**

`timeValue`: An integer representing the number of milliseconds since 1 January 1970, 00:00:00 UTC.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript setTime Method</title>
</head>
<body>
<script type="text/javascript">
var dt = new Date( "Aug 28, 2008 23:30:00" );
dt.setTime( 5000000 );
document.write( dt );
</script>
</body>
</html>
```

**Output**

```
Thu Jan 01 1970 06:53:20 GMT+0530 (India Standard Time)
```

**setUTCDate()**

Javascript date `setUTCDate()` method sets the day of the month for a specified date according to universal time.
Syntax
Its syntax is as follows:

```
Date.setUTCDate(dayValue)
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**

**dayValue**: An integer from 1 to 31, representing the day of the month.

If a parameter you specify is outside the expected range, setUTCDate attempts to update the date information in the Date object accordingly.

**Example**

Try the following example.

```
<html>
<head>
<title>JavaScript setUTCDate Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setUTCDate( 20 );
    document.write( dt );
</script>
</body>
</html>
```

**Output**

```
Wed Aug 20 2008 23:30:00 GMT+0530 (India Standard Time)
```

**setUTCFullYear()**

Javascript date **setUTCFullYear()** method sets the full year for a specified date according to universal time.

**Syntax**

Its syntax is as follows:
Date.setUTCFullYear(yearValue[, monthValue[, dayValue]])

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**

- **yearValue**: An integer specifying the numeric value of the year, for example, 2008.
- **monthValue**: An integer between 0 and 11 representing the months January through December.
- **dayValue**: An integer between 1 and 31 representing the day of the month. If you specify the dayValue parameter, you must also specify the monthValue.

If you do not specify the monthValue and dayValue parameters, the values returned from the getMonth and getDate methods are used. If a parameter you specify is outside of the expected range, setUTCFullYear attempts to update the other parameters and the date information in the Date object accordingly. For example, if you specify 15 for monthValue, the year is incremented by 1 (year + 1), and 3 is used for the month.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript setUTCFullYear Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setUTCFullYear( 2006 );
    document.write( dt );
</script>
</body>
</html>
```

**Output**

Mon Aug 28 2006 23:30:00 GMT+0530 (India Standard Time)
setUTCHours ()

Javascript date setUTCHours() method sets the hour for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setUTCHours(hoursValue[, minutesValue[, secondsValue[, msValue]]])
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**

- **hoursValue**: An integer between 0 and 23, representing the hour.
- **minutesValue**: An integer between 0 and 59, representing the minutes.
- **secondsValue**: An integer between 0 and 59, representing the seconds. If you specify the secondsValue parameter, you must also specify the minutesValue.
- **msValue**: A number between 0 and 999, representing the milliseconds. If you specify the msValue parameter, you must also specify the minutesValue and secondsValue.

If you do not specify the minutesValue, secondsValue, and msValue parameters, the values returned from the getUTCMinutes, getUTCSecs, and getUTCMillis methods are used.

If a parameter you specify is outside the expected range, setUTCHours attempts to update the date information in the Date object accordingly. For example, if you use 100 for secondsValue, the minutes will be incremented by 1 (min + 1), and 40 will be used for seconds.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript setUTCHours Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setUTCHours( 15 );
</script>
</body>
</html>
```
Javascript

```javascript
    document.write( dt );
</script>
</body>
</html>
```

Output

Thu Aug 28 2008 20:30:00 GMT+0530 (India Standard Time)

**setUTCMilliseconds()**

Javascript date **setUTCMilliseconds()** method sets the milliseconds for a specified date according to universal time.

**Syntax**

Its syntax is as follows:

```javascript
Date.setUTCMilliseconds(millisecondsValue)
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**

**millisecondsValue**: A number between 0 and 999, representing the milliseconds.

If a parameter you specify is outside the expected range, setUTCMilliseconds attempts to update the date information in the Date object accordingly. For example, if you use 1100 for millisecondsValue, the seconds stored in the Date object will be incremented by 1, and 100 will be used for milliseconds.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript setUTCMilliseconds Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setUTCMilliseconds( 1100 );
</script>
</body>
</html>
```
Javascript

```javascript
    document.write( dt );
    </script>
    </body>
    </html>
```

Output

Thu Aug 28 2008 23:30:01 GMT+0530 (India Standard Time)

**setUTCMinutes()**

Javascript date **setUTCMinutes()** method sets the minutes for a specified date according to universal time.

**Syntax**

Its syntax is as follows:

```
Date.setUTCMinutes(minutesValue[, secondsValue[, msValue]])
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**

- **minutesValue**: An integer between 0 and 59, representing the minutes.
- **secondsValue**: An integer between 0 and 59, representing the seconds. If you specify the secondsValue parameter, you must also specify the minutesValue.
- **msValue**: A number between 0 and 999, representing the milliseconds. If you specify the msValue parameter, you must also specify the minutesValue and secondsValue.

If you do not specify the secondsValue and msValue parameters, the values returned from getUTCSecs and getUTCMillis methods are used.

If a parameter you specify is outside of the expected range, setUTCMinutes attempts to update the date information in the Date object accordingly. For example, if you use 100 for secondsValue, the minutes (minutesValue) will be incremented by 1 (minutesValue + 1), and 40 will be used for seconds.

**Example**
Try the following example.

```html
<html>
<head>
<title>JavaScript setUTCMinutes Method</title>
</head>
<body>
<script type="text/javascript">
  var dt = new Date( "Aug 28, 2008 13:30:00" );
  dt.setUTCMinutes( 65 );
  document.write( dt );
</script>
</body>
</html>
```

**Output**

Thu Aug 28 2008 14:35:00 GMT+0530 (India Standard Time)

---

**setUTC Month ()**

Javascript date **setUTCMonth ( )** method sets the month for a specified date according to universal time.

**Syntax**

The following syntax for setUTCMonth ( ) Method.

```
Date.setUTCMonth ( monthvalue )
```

**Note:** Parameters in the bracket are always optional.

**Parameter Detail**

- **monthValue** : An integer between 0 and 11, representing the month.

---

**Example**

Try the following example.
Output

Fri Mar 28 2008 13:30:00 GMT+0530 (India Standard Time)

setUTCSeconds()

Javascript date `setUTCSeconds()` method sets the seconds for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setUTCSeconds(secondsValue[, msValue])
```

**Note:** Parameters in the bracket are always optional.

Parameter Detail

- **secondsValue:** An integer between 0 and 59, representing the seconds.
- **msValue:** A number between 0 and 999, representing the milliseconds.

If you do not specify the `msValue` parameter, the value returned from the `getUTCMilliseconds` methods is used.

If a parameter you specify is outside the expected range, `setUTCSeconds` attempts to update the date information in the Date object accordingly. For example, if you use 100 for `secondsValue`, the minutes stored in the Date object will be incremented by 1, and 40 will be used for seconds.
Example
Try the following example.

```html
<html>
<head>
<title>JavaScript setUTCSeconds Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 13:30:00" );
    dt.setUTCSeconds( 65 );
    document.write( dt );
</script>
</body>
</html>
```

Output
Thu Aug 28 2008 13:31:05 GMT+0530 (India Standard Time)

setYear()
Javascript date setYear() method sets the year for a specified date according to universal time.

Syntax
Its syntax is as follows:

```
Date.setYear(yearValue)
```

Note: Parameters in the bracket are always optional.

Parameter Detail
yearValue: An integer value.
Javascript setYear Method

```javascript
var dt = new Date( "Aug 28, 2008 13:30:00" );
dt.setYear( 2000 );
document.write( dt );
```

Output

Mon Aug 28 2000 13:30:00 GMT+0530 (India Standard Time)

toDateString ()

Javascript date `toDateString()` method returns the date portion of a Date object in human readable form.

**Syntax**

Its syntax is as follows:

```javascript
Date.toDateString()
```

**Return Value**

Returns the date portion of a Date object in human readable form.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript toDateString Method</title>
</head>
<body>
<script type="text/javascript">
</script>
</body>
</html>
```
```javascript
var dt = new Date(1993, 6, 28, 14, 39, 7);
document.write( "Formated Date : " + dt.toDateString() );
</script>
</body>
</html>
```

**Output**

| Formated Date : Wed Jul 28 1993 |

---

**toGMTString()**

Javascript date **toGMTString()** method converts a date to a string, using Internet GMT conventions.

This method is no longer used and has been replaced by the toUTCString method.

**Syntax**

Its syntax is as follows:

```
Date.toGMTString()
```

**Return Value**

Returns a date to a string, using Internet GMT conventions.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript toGMTString Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
document.write( "Formated Date : " + dt.toGMTString() );
</script>
</body>
```

262
toLocaleDateString ()

Javascript date toLocaleDateString() method converts a date to a string, returning the "date" portion using the operating system's locale's conventions.

Syntax

Its syntax is as follows:

Date.toLocaleDateString()

Return Value

Returns a date to a string, using Internet GMT conventions.

Example

Try the following example.

```html
<html>
<head>
<title>JavaScript toGMTString Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write("Formated Date : " + dt.toGMTString());
</script>
</body>
</html>
```

Output

Formated Date : Wed, 28 Jul 1993 09:09:07 GMT
**toLocaleDateString()**

Javascript date **toLocaleDateString()** method converts a date to a string, returning the "date" portion using the operating system's locale's conventions.

**Syntax**

Its syntax is as follows:

```
Date.toLocaleString()
```

**Return Value**

Returns the "date" portion using the operating system's locale's conventions.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript toLocaleDateString Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formated Date : " + dt.toLocaleDateString() );
</script>
</body>
</html>
```

**Output**

```
Formated Date : 7/28/1993
```

---

**toLocaleFormat()**

Javascript date **toLocaleFormat()** method converts a date to a string using the specified formatting.

**Note:** This method may not compatible with all the browsers.
**Javascript**

**Syntax**

Its syntax is as follows:

Date.toLocaleFormat()

**Parameter Details**

**formatString**: A format string in the same format expected by the `strftime()` function in C.

**Return Value**

Returns the formatted date.

**Example**

Try the following example.

```
<html>
<head>
<title>JavaScript toLocaleFormat Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formated Date : " + dt.toLocaleFormat("%A, %B %e, %Y");
</script>
</body>
</html>
```

**Output**

Formated Date : Wed Jul 28 1993 14:39:07 GMT+0530 (India Standard Time)

**toLocaleString()**

Javascript date `toLocaleString()` method converts a date to a string, using the operating system's local conventions.

The toLocaleString method relies on the underlying operating system in formatting dates. It converts the date to a string using the formatting convention of the operating system where the script is running. For example, in the United States, the month appears before the date (04/15/98), whereas in Germany the date appears before the month (15.04.98).
Syntax

Its syntax is as follows:

```javascript
Date.toLocaleString()
```

Return Value

Returns the formatted date in a string format.

Example

Try the following example.

```html
<html>
<head>
<title>JavaScript toLocaleString Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formated Date : " + dt.toLocaleString() );
</script>
</body>
</html>
```

Output

```
Formated Date : 7/28/1993, 2:39:07 PM
```

**toLocaleTimeString()**

Javascript date **toLocaleTimeString()** method converts a date to a string, returning the "date" portion using the current locale's conventions.

The `toLocaleTimeString()` method relies on the underlying operating system in formatting dates. It converts the date to a string using the formatting convention of the operating system where the script is running. For example, in the United States, the month appears before the date (04/15/98), whereas in Germany, the date appears before the month (15.04.98).

Syntax

Its syntax is as follows:
Date.toLocaleTimeString ()

**Return Value**

Returns the formatted date in a string format.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript toLocaleTimeString Method</title>
</head>
<body>
<script type="text/javascript">
  var dt = new Date(1993, 6, 28, 14, 39, 7);
  document.write( "Formated Date : " + dt.toLocaleTimeString() );
</script>
</body>
</html>
```

**Output**

Formated Date : 2:39:07 PM

toSource ()

This method returns a string representing the source code of the object.

**Note:** This method may not be compatible with all the browsers.

**Syntax**

The following syntax for toSource () Method.

```javascript
Date.toSource ()
```

**Return Value**

- For the built-in Date object, toSource returns a string (new Date(...)) indicating that the source code is not available
For instances of Date, toSource returns a string representing the source code.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript toSource Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formated Date : " + dt.toSource() );
</script>
</body>
</html>
```

**Output**

Formated Date : (new Date(743850547000))

### toString ()

This method returns a string representing the specified Date object.

**Syntax**

The following syntax for toString () Method.

```
Date.toString()
```

**Return Value**

Returns a string representing the specified Date object.

**Example**

Try the following example.

```html
<html>
```

268
<head>
<title>JavaScript toString Method</title>
</head>
<body>
<script type="text/javascript">
    var dateobject = new Date(1993, 6, 28, 14, 39, 7);
    stringobj = dateobject.toString();
    document.write( "String Object : " + stringobj );
</script>
</body>
</html>

**Output**


### toTimeString()

This method returns the time portion of a Date object in human readable form.

**Syntax**

Its syntax is as follows:

```javascript
Date.toTimeString()
```

**Return Value**

Returns the time portion of a Date object in human readable form.

**Example**

Try the following example.

```html
<html>
<head>
<title>JavaScript toTimeString Method</title>
</head>
```
Output

14:39:07 GMT+0530 (India Standard Time)

toUTCString ()

This method converts a date to a string, using the universal time convention.

Syntax

Its syntax is as follows:

```
Date.toTimeString()
```

Return Value

Returns converted date to a string, using the universal time convention.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toUTCString Method</title>
</head>
<body>
<script type="text/javascript">
    var dateobject = new Date(1993, 6, 28, 14, 39, 7);
    document.write( dateobject.toUTCString() );
</script>
</body>
```
This method returns the primitive value of a Date object as a number data type, the number of milliseconds since midnight 01 January, 1970 UTC.

Syntax
Its syntax is as follows:

```javascript
Date.valueOf()
```

Return Value
Returns the primitive value of a Date object.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript valueOf Method</title>
</head>
<body>
<script type="text/javascript">
    var dateobject = new Date(1993, 6, 28, 14, 39, 7);
    document.write( dateobject.valueOf() );
</script>
</body>
</html>
```
Date Static Methods

In addition to the many instance methods listed previously, the Date object also defines two static methods. These methods are invoked through the Date() constructor itself.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date.parse()</td>
<td>Parses a string representation of a date and time and returns the internal millisecond representation of that date.</td>
</tr>
<tr>
<td>Date.UTC()</td>
<td>Returns the millisecond representation of the specified UTC date and time.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usages of Date Static methods.

Date.parse()

Javascript date parse() method takes a date string and returns the number of milliseconds since midnight of January 1, 1970.

Syntax

Its syntax is as follows:

```
Date.parse(datestring)
```

Note: Parameters in the bracket are always optional.

Parameter Details
datestring: A string representing a date.

Return Value

Number of milliseconds since midnight of January 1, 1970.

Example
Try the following example.

```html
<html>
<head>
<title>JavaScript parse Method</title>
</head>
<body>
<script type="text/javascript">
    var msecs = Date.parse( "Aug 28, 2008 23:30:00" );
    document.write( "Number of milliseconds from 1970: " + msecs );
</script>
</body>
</html>
```

Output

```
Number of milliseconds from 1970: 1219946400000
```

**Date.UTC()**

This method takes a date and returns the number of milliseconds since midnight of January 1, 1970 according to universal time.

**Syntax**

Its syntax is as follows:

```
Date.year,month,day,[hours,[minutes,[seconds,[ms]]]]
```

**Note:** Parameters in the bracket are always optional.

**Parameter Details**

- **year**: A four digit number representing the year.
- **month**: An integer between 0 and 11 representing the month.
- **day**: An integer between 1 and 31 representing the date.
- **hours**: An integer between 0 and 23 representing the hour.
- **minutes**: An integer between 0 and 59 representing the minutes.
- **seconds**: An integer between 0 and 59 representing the seconds.
- **ms**: An integer between 0 and 999 representing the milliseconds.

**Return Value**
Number of milliseconds since midnight of January 1, 1970.

**Example**
Try the following example.

```html
<html>
<head>
<title>JavaScript UTC Method</title>
</head>
<body>
<script type="text/javascript">
    var msecs = Date.UTC(2008,9,6);
    document.write( "Number of milliseconds from 1970: " + msecs );
</script>
</body>
</html>
```

**Output**
```
Number of milliseconds from 1970: 1223251200000
```
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.

Thus, you refer to the constant **pi** as `Math.PI` and you call the `sine` function as `Math.sin(x)`, where `x` is the method's argument.

**Syntax**
The syntax to call the properties and methods of Math are as follows:

```javascript
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

**Math Properties**
Here is a list of all the properties of Math and their description.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Euler's constant and the base of natural logarithms, approximately 2.718.</td>
</tr>
<tr>
<td>LN2</td>
<td>Natural logarithm of 2, approximately 0.693.</td>
</tr>
<tr>
<td>LN10</td>
<td>Natural logarithm of 10, approximately 2.302.</td>
</tr>
<tr>
<td>LOG2E</td>
<td>Base 2 logarithm of E, approximately 1.442.</td>
</tr>
<tr>
<td>LOG10E</td>
<td>Base 10 logarithm of E, approximately 0.434.</td>
</tr>
<tr>
<td>PI</td>
<td>Ratio of the circumference of a circle to its diameter, approximately 3.14159.</td>
</tr>
<tr>
<td>SQRT1_2</td>
<td>Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.</td>
</tr>
<tr>
<td>SQRT2</td>
<td>Square root of 2, approximately 1.414.</td>
</tr>
</tbody>
</table>
In the following sections, we will have a few examples to demonstrate the usage of Math properties.

**Math-E**

This is an Euler's constant and the base of natural logarithms, approximately 2.718.

**Syntax**

Its syntax is as follows:

```javascript
Math.E
```

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math E Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.E
    document.write("Property Value is ": property_value);
</script>
</body>
</html>
```

**Output**

```
Property Value is 2.718281828459045
```
**Math-LN2**

It returns the natural logarithm of 2 which is approximately 0.693.

**Syntax**

Its syntax is as follows:

```javascript
Math.LN2
```

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math LN2 Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.LN2
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

**Output**

```
Property Value is : 0.6931471805599453
```

**Math-LN10**

It returns the natural logarithm of 10 which is approximately 2.302.

**Syntax**

Its syntax is as follows:

```javascript
Math.LN10
```
Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math LN10 Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.LN10
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

Output

```
Property Value is : 2.302585092994046
```

**Math-LOG2E**

It returns the base 2 logarithm of E which is approximately 1.442.

**Syntax**

Its syntax is as follows:

```
Math.LOG2E
```

Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math LOG2E Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.LOG2E
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```
var property_value = Math.LOG2E
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>

Output

Property Value is : 1.4426950408889634

Math-LOG10E

It returns the base 10 logarithm of E which is approximately 0.434.

Syntax

Its syntax is as follows:

Math.LOG10E

Example

Try the following example program.

<html>
<head>
<title>JavaScript Math LOG10E Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.LOG10E
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>

Output
Property Value is : 0.4342944819032518

Math-PI
It returns the ratio of the circumference of a circle to its diameter which is approximately 3.14159.

Syntax
Its syntax is as follows:

Math.PI

Example
Try the following example program.

```html
<html>
<head>
<title>JavaScript Math PI Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.PI
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

Output

Property Value is : 3.141592653589793

Math-SQRT1_2
It returns the square root of 1/2; equivalently, 1 over the square root of 2 which is approximately 0.707.
**Syntax**
Its syntax is as follows:

```
Math.SQRT1_2
```

**Example**
Try the following example program.

```
<html>
<head>
<title>JavaScript Math SQRT1_2 Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.SQRT1_2
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

**Output**
```
Property Value is : 0.7071067811865476
```

**Math-SQRT2**
It returns the square root of 2 which is approximately 1.414.

**Syntax**
Its syntax is as follows:

```
Math.SQRT2
```

**Example**
Try the following example program.

```
<html>
```
<head>
<title>JavaScript Math SQRT2 Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.SQRT2
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>

Output

Property Value is : 1.4142135623730951

**Math Methods**

Here is a list of the methods associated with Math object and their description.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs()</td>
<td>Returns the absolute value of a number.</td>
</tr>
<tr>
<td>acos()</td>
<td>Returns the arccosine (in radians) of a number.</td>
</tr>
<tr>
<td>asin()</td>
<td>Returns the arcsine (in radians) of a number.</td>
</tr>
<tr>
<td>atan()</td>
<td>Returns the arctangent (in radians) of a number.</td>
</tr>
<tr>
<td>atan2()</td>
<td>Returns the arctangent of the quotient of its arguments.</td>
</tr>
<tr>
<td>ceil()</td>
<td>Returns the smallest integer greater than or equal to a number.</td>
</tr>
<tr>
<td>cos()</td>
<td>Returns the cosine of a number.</td>
</tr>
<tr>
<td>exp()</td>
<td>Returns $E^n$, where N is the argument, and E is Euler's constant, the base of the natural logarithm.</td>
</tr>
<tr>
<td>floor()</td>
<td>Returns the largest integer less than or equal to a number.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>log()</td>
<td>Returns the natural logarithm (base E) of a number.</td>
</tr>
<tr>
<td>max()</td>
<td>Returns the largest of zero or more numbers.</td>
</tr>
<tr>
<td>min()</td>
<td>Returns the smallest of zero or more numbers.</td>
</tr>
<tr>
<td>pow()</td>
<td>Returns base to the exponent power, that is, base exponent.</td>
</tr>
<tr>
<td>random()</td>
<td>Returns a pseudo-random number between 0 and 1.</td>
</tr>
<tr>
<td>round()</td>
<td>Returns the value of a number rounded to the nearest integer.</td>
</tr>
<tr>
<td>sin()</td>
<td>Returns the sine of a number.</td>
</tr>
<tr>
<td>sqrt()</td>
<td>Returns the square root of a number.</td>
</tr>
<tr>
<td>tan()</td>
<td>Returns the tangent of a number.</td>
</tr>
<tr>
<td>toSource()</td>
<td>Returns the string &quot;Math&quot;.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of the methods associated with Math.

**abs()**

This method returns the absolute value of a number.

**Syntax**

Its syntax is as follows:

```javascript
Math.abs( x ) ;
```

**Parameter Details**

- **x**: A number.

**Return Value**

Returns the absolute value of a number.
Try the following example program.

```
<html>
<head>
<title>JavaScript Math abs() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.abs(-1);
    document.write("First Test Value : " + value);

    var value = Math.abs(null);
    document.write("<br />Second Test Value : " + value);

    var value = Math.abs(20);
    document.write("<br />Third Test Value : " + value);

    var value = Math.abs("string");
    document.write("<br />Fourth Test Value : " + value);

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

Output

| First Test Value : 1 |
| Second Test Value : 0 |
| Third Test Value : 20 |
| Fourth Test Value : NaN |

**acos()**

This method returns the arccosine in radians of a number. The acos method returns a numeric value between 0 and pi radians for x between -1 and 1. If the value of number is outside this range, it returns NaN.

**Syntax**
Its syntax is as follows:

```
Math.cos( x );
```

**Parameter Details**

- **x**: A number.

**Return Value**

Returns the arccosine in radians of a number.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math acos() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.acos(-1);
    document.write("First Test Value : " + value);

    var value = Math.acos(null);
    document.write("\nSecond Test Value : " + value);

    var value = Math.acos(30);
    document.write("\nThird Test Value : " + value);

    var value = Math.acos("string");
    document.write("\nFourth Test Value : " + value);

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

**Output**

285
First Test Value : 3.141592653589793
Second Test Value : 1.5707963267948966
Third Test Value : NaN
Fourth Test Value : NaN

**asin()**

This method returns the arcsine in radians of a number. The asin method returns a numeric value between -pi/2 and pi/2 radians for x between -1 and 1. If the value of number is outside this range, it returns NaN.

**Syntax**

Its syntax is as follows:

```javascript
Math.asin( x ) ;
```

**Parameter Details**

**x:** A number.

**Return Value**

Returns the arcsine in radians of a number.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math asin() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.asin(-1);
    document.write("First Test Value : " + value );
    
    var value = Math.asin(null);
    document.write("<br />Second Test Value : " + value );
</script>
</body>
</html>
```
```
var value = Math.asin(30);
document.write("<br />Third Test Value : " + value );

var value = Math.asin("string");
document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

<table>
<thead>
<tr>
<th>Test Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Test Value</td>
</tr>
<tr>
<td>-1.5707963267948966</td>
</tr>
<tr>
<td>Second Test Value</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Third Test Value</td>
</tr>
<tr>
<td>NaN</td>
</tr>
<tr>
<td>Fourth Test Value</td>
</tr>
<tr>
<td>NaN</td>
</tr>
</tbody>
</table>

**atan()**

This method returns the arctangent in radians of a number. The atan method returns a numeric value between -π/2 and π/2 radians.

**Syntax**

Its syntax is as follows:

```
Math.atan( x ) ;
```

**Parameter Details**

**x:** A number.

**Return Value**

Returns the arctangent in radians of a number.

**Example**

Try the following example program.

```
<html>
<head>
```
<title>JavaScript Math atan() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.atan(-1);
    document.write("First Test Value : " + value);

    var value = Math.atan(.5);
    document.write("<br />Second Test Value : " + value);

    var value = Math.atan(30);
    document.write("<br />Third Test Value : " + value);

    var value = Math.atan("string");
    document.write("<br />Fourth Test Value : " + value);

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

Output
First Test Value : -0.7853981633974483
Second Test Value : 0.4636476090008061
Third Test Value : 1.5374753309166493
Fourth Test Value : NaN

atan2() 
This method returns the arctangent of the quotient of its arguments. The atan2 method returns a numeric value between -pi and pi representing the angle theta of an (x, y) point.

Syntax
Its syntax is as follows:

Math.atan2( x, y );
Parameter Details

**X and y:** numbers.

Return Value

Returns the arctangent in radians of a number.

- `Math.atan2(±0, -0)` returns ±PI.
- `Math.atan2(±0, +0)` returns ±0.
- `Math.atan2(±0, -x)` returns ±PI for x < 0.
- `Math.atan2(±0, x)` returns ±0 for x > 0.
- `Math.atan2(y, ±0)` returns -PI/2 for y > 0.
- `Math.atan2(±y, -Infinity)` returns ±PI for finite y > 0.
- `Math.atan2(±y, +Infinity)` returns ±0 for finite y > 0.
- `Math.atan2(±Infinity, +x)` returns ±PI/2 for finite x.
- `Math.atan2(±Infinity, -Infinity)` returns ±3*PI/4.

Example

Try the following example program.

```html
<html>
<head>
	<title>JavaScript Math atan2() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.atan2(90,15);
    document.write("First Test Value : " + value);

    var value = Math.atan2(15,90);
    document.write("<br />
Second Test Value : " + value);

    var value = Math.atan2(0, -0);
    document.write("<br />
Third Test Value : " + value);
</script>
</body>
</html>
```
var value = Math.atan2(+Infinity, -Infinity);
document.write("<br />Fourth Test Value : " + value);
</script>
</body>
</html>

Output

<table>
<thead>
<tr>
<th>Test Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Test Value : 1.4056476493802699</td>
</tr>
<tr>
<td>Second Test Value : 0.16514867741462683</td>
</tr>
<tr>
<td>Third Test Value : 3.141592653589793</td>
</tr>
<tr>
<td>Fourth Test Value : 2.356194490192345</td>
</tr>
</tbody>
</table>

**ceil ()**

This method returns the smallest integer greater than or equal to a number.

**Syntax**

Its syntax is as follows:

```javascript
Math.ceil ( x ) ;
```

**Parameter Details**

x: a number.

**Return Value**

Returns the smallest integer greater than or equal to a number.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math ceil() Method</title>
</head>
<body>
<script type="text/javascript">
```
```javascript
var value = Math.ceil(45.95);
document.write("First Test Value : " + value);

var value = Math.ceil(45.20);
document.write("<br />Second Test Value : " + value);

var value = Math.ceil(-45.95);
document.write("<br />Third Test Value : " + value);

var value = Math.ceil(-45.20);
document.write("<br />Fourth Test Value : " + value);
</script>
</body>
</html>

Output

First Test Value : 46  
Second Test Value : 46  
Third Test Value : -45  
Fourth Test Value : -45

**cos()**

This method returns the cosine of a number. The cos method returns a numeric value between -1 and 1, which represents the cosine of the angle.

**Syntax**

Its syntax is as follows:

```javascript
Math.cos ( x );
```

**Parameter Details**

x: a number.

**Return Value**

Returns the cosine of a number.
Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math cos() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.cos(90);
    document.write("First Test Value : " + value);

    var value = Math.cos(30);
    document.write("Second Test Value : " + value);

    var value = Math.cos(-1);
    document.write("Third Test Value : " + value);

    var value = Math.cos(2*Math.PI);
    document.write("Fourth Test Value : " + value);
</script>
</body>
</html>
```

Output

```
First Test Value : -0.4480736161291702
Second Test Value : 0.15425144988758405
Third Test Value : 0.5403023058681398
Fourth Test Value : 1
```

**exp()**

This method returns $E^x$, where $x$ is the argument, and $E$ is the Euler's constant, the base of the natural logarithms.
Syntax
Its syntax is as follows:

```
Math.exp ( x ) ;
```

Parameter Details

**x**: a number.

Return Value

Returns the exponential value of the variable `x`.

Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math exp() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.exp(1);
    document.write("First Test Value : " + value);

    var value = Math.exp(30);
    document.write("<br />Second Test Value : " + value);

    var value = Math.exp(-1);
    document.write("<br />Third Test Value : " + value);

    var value = Math.exp(.5);
    document.write("<br />Fourth Test Value : " + value);
</script>
</body>
</html>
```
### Javascript

**Output**

<table>
<thead>
<tr>
<th>First Test Value</th>
<th>2.718281828459045</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second Test Value</td>
<td>10686474581524.482</td>
</tr>
<tr>
<td>Third Test Value</td>
<td>0.3678794411714424</td>
</tr>
<tr>
<td>Fourth Test Value</td>
<td>1.6487212707001282</td>
</tr>
</tbody>
</table>

**floor()**

This method returns the largest integer less than or equal to a number.

**Syntax**

Its syntax is as follows:

```javascript
Math.floor(x);
```

**Parameter Details**

- **x**: a number.

**Return Value**

Returns the largest integer less than or equal to a number `x`.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math floor() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.floor(10.3);
    document.write("First Test Value : " + value);

    var value = Math.floor(30.9);
    document.write("<br/>Second Test Value : " + value);

    var value = Math.floor(-2.9);
</script>
</body>
</html>
```
log()

This method returns the natural logarithm (base E) of a number. If the value of number is negative, the return value is always NaN.

Syntax

Its syntax is as follows:

Math.log(x);

Parameter Details

x: a number.

Return Value

Returns the natural logarithm (base E) of a number.

Example

Try the following example program.

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

```
<script type="text/javascript">
    var value = Math.log(10);
    document.write("First Test Value : " + value);

    var value = Math.log(0);
    document.write("Second Test Value : " + value);

    var value = Math.log(-1);
    document.write("Third Test Value : " + value);

    var value = Math.log(100);
    document.write("Fourth Test Value : " + value);
</script>
</body>
</html>

Output

First Test Value : 2.302585092994046
Second Test Value : -Infinity
Third Test Value : NaN
Fourth Test Value : 4.605170185988092

max() 

This method returns the largest of zero or more numbers. If no arguments are given, the results is -Infinity.

Syntax

Its syntax is as follows:

Math.max(value1, value2, ... valueN);

Parameter Details

value1, value2, ... valueN : Numbers.

Return Value
Returns the largest of zero or more numbers.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math max() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.max(10, 20, -1, 100);
    document.write("First Test Value : " + value);

    var value = Math.max(-1, -3, -40);
    document.write("<br />Second Test Value : " + value);

    var value = Math.max(0, -1);
    document.write("<br />Third Test Value : " + value);

    var value = Math.max(100);
    document.write("<br />Fourth Test Value : " + value);
</script>
</body>
</html>
```

**Output**

First Test Value : 100  
Second Test Value : -1  
Third Test Value : 0  
Fourth Test Value : 100
**min()**

This method returns the smallest of zero or more numbers. If no arguments are given, the result is **+Infinity**.

**Syntax**

Its syntax is as follows:

```
Math.min(value1, value2, ... valueN);
```

**Parameter Details**

value1, value2, ... valueN : Numbers.

**Return Value**

Returns the smallest of zero or more numbers.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math min() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.min(10, 20, -1, 100);
    document.write("First Test Value : " + value);

    var value = Math.min(-1, -3, -40);
    document.write("<br />Second Test Value : " + value);

    var value = Math.min(0, -1);
    document.write("<br />Third Test Value : " + value);

    var value = Math.min(100);
    document.write("<br />Fourth Test Value : " + value);
</script>
</body>
</html>
```
Output

First Test Value : -1
Second Test Value : -40
Third Test Value : -1
Fourth Test Value : 100

pow()

This method returns the base to the exponent power, that is, base^exponent.

Syntax

Its syntax is as follows:

Math.pow(base, exponent );

Parameter Details

- **base**: The base number.
- **exponents**: The exponent to which to raise the base.

Return Value

Returns the base to the exponent power, that is, base^exponent.

Example

Try the following example program.
var value = Math.pow(7, 2);
document.write("First Test Value : " + value);

var value = Math.pow(8, 8);
document.write("Second Test Value : " + value);

var value = Math.pow(-1, 2);
document.write("Third Test Value : " + value);

var value = Math.pow(0, 10);
document.write("Fourth Test Value : " + value);
</script>
</body>
</html>

Output
First Test Value : 49
Second Test Value : 16777216
Third Test Value : 1
Fourth Test Value : 0

random ()
This method returns a random number between 0 (inclusive) and 1 (exclusive).

Syntax
Its syntax is as follows:

Math.random();

Return Value
Returns a random number between 0 (inclusive) and 1 (exclusive).

Example
Try the following example program.
<html>
<head>
<title>JavaScript Math random() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.random( );
    document.write("First Test Value : " + value );

    var value = Math.random( );
    document.write("<br />Second Test Value : " + value );

    var value = Math.random( );
    document.write("<br />Third Test Value : " + value );

    var value = Math.random( );
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>

Output

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Test Value: 0.4093269258737564</td>
</tr>
<tr>
<td>Second Test Value: 0.023646741174161434</td>
</tr>
<tr>
<td>Third Test Value: 0.2672571325674653</td>
</tr>
<tr>
<td>Fourth Test Value: 0.38755513448268175</td>
</tr>
</tbody>
</table>

**Math.random()**

This method returns the value of a number rounded to the nearest integer.

**Syntax**

Its syntax is as follows:

```javascript
Math.round ( );
```
Return Value
Returns the value of a number rounded to the nearest integer.

Example
Try the following example program.

```html
<html>
<head>
<title>JavaScript Math round() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.round( 0.5 );
    document.write("First Test Value : " + value );

    var value = Math.round( 20.7 );
    document.write("<br />Second Test Value : " + value );

    var value = Math.round( 20.3 );
    document.write("<br />Third Test Value : " + value );

    var value = Math.round( -20.3 );
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output
First Test Value : 1
Second Test Value : 21
Third Test Value : 20
Fourth Test Value : -20
**sin()**

This method returns the sine of a number. The `sin` method returns a numeric value between -1 and 1, which represents the sine of the argument.

**Syntax**

Its syntax is as follows:

```javascript
Math.sin(x);
```

**Parameter Details**

- **x**: A number.

**Return Value**

Returns the sine of a number.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math sin() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.sin(0.5);
    document.write("First Test Value : " + value);

    var value = Math.sin(90);
    document.write("<br />Second Test Value : " + value);

    var value = Math.sin(1);
    document.write("<br />Third Test Value : " + value);

    var value = Math.sin(Math.PI/2);
    document.write("<br />Fourth Test Value : " + value);
</script>
</body>
</html>
```
Output

<table>
<thead>
<tr>
<th>Test Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Test Value : 0.479425538604203</td>
</tr>
<tr>
<td>Second Test Value : 0.8939966636005579</td>
</tr>
<tr>
<td>Third Test Value : 0.8414709848078965</td>
</tr>
<tr>
<td>Fourth Test Value : 1</td>
</tr>
</tbody>
</table>

**sqrt()**

This method returns the square root of a number. If the value of a number is negative, sqrt returns NaN.

**Syntax**

Its syntax is as follows:

```javascript
Math.sqrt(x);
```

**Parameter Details**

**x**: A number.

**Return Value**

Returns the square root of a given number.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math sqrt() Method</title>
</head>
<body>
<script type="text/javascript">
</script>
</body>
</html>
```
```javascript
var value = Math.sqrt( 0.5 );
document.write("First Test Value : " + value);

var value = Math.sqrt( 81 );
document.write("<br />Second Test Value : " + value);

var value = Math.sqrt( 13 );
document.write("<br />Third Test Value : " + value);

var value = Math.sqrt(-4);
document.write("<br />Fourth Test Value : " + value);
</script>
</body>
</html>

Output

First Test Value : 0.7071067811865476
Second Test Value : 9
Third Test Value : 3.605551275463989
Fourth Test Value : NaN

tan()

This method returns the tangent of a number. The tan method returns a numeric value that represents the tangent of the angle.

Syntax
Its syntax is as follows:

```javascript
Math.tan( x );
```

Parameter Details

x: A number representing an angle in radians.

Return Value

Returns the tangent of a number.
Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math tan() Method</title>
</head>
<body>
<script type="text/javascript">
var value = Math.tan( -30 );
document.write("First Test Value : " + value );

var value = Math.tan( 90 );
document.write("<br />Second Test Value : " + value );

var value = Math.tan( 45 );
document.write("<br />Third Test Value : " + value );

var value = Math.tan( Math.PI/180 );
document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

First Test Value : 1
Second Test Value : 21
Third Test Value : 20
Fourth Test Value : -20

toSource ()

This method returns the string "Math". But this method does not work with IE.
Syntax

Its syntax is as follows:

Math.toSource ( );

Return Value

Returns the string “Math”.

Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript Math toSource() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.toSource( );
    document.write("Value : " + value );

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

Output

Value : Math
A regular expression is an object that describes a pattern of characters. The JavaScript RegExp class represents regular expressions, and both String and RegExp define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

**Syntax**

A regular expression could be defined with the RegExp() constructor, as follows:

```javascript
var pattern = new RegExp(pattern, attributes);
```

or simply

```javascript
var pattern = /pattern/attributes;
```

Here is the description of the parameters:

- **pattern**: A string that specifies the pattern of the regular expression or another regular expression.
- **attributes**: An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively.

**Brackets**

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[...]</td>
<td>Any one character between the brackets.</td>
</tr>
<tr>
<td>[^...]</td>
<td>Any one character not between the brackets.</td>
</tr>
<tr>
<td>[0-9]</td>
<td>It matches any decimal digit from 0 through 9.</td>
</tr>
<tr>
<td>Character Range</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[a-z]</td>
<td>It matches any character from lowercase a through lowercase z.</td>
</tr>
<tr>
<td>[A-Z]</td>
<td>It matches any character from uppercase A through uppercase Z.</td>
</tr>
<tr>
<td>[a-zA-Z]</td>
<td>It matches any character from lowercase a through uppercase Z.</td>
</tr>
</tbody>
</table>

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

**Quantifiers**

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character has a specific connotation. The +, *, ?, and $ flags all follow a character sequence.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p+</td>
<td>It matches any string containing at least one p.</td>
</tr>
<tr>
<td>p*</td>
<td>It matches any string containing zero or more p's.</td>
</tr>
<tr>
<td>p?</td>
<td>It matches any string containing one or more p's.</td>
</tr>
<tr>
<td>p{N}</td>
<td>It matches any string containing a sequence of N p's.</td>
</tr>
<tr>
<td>p{2,3}</td>
<td>It matches any string containing a sequence of two or three p's.</td>
</tr>
<tr>
<td>p{2,}</td>
<td>It matches any string containing a sequence of at least two p's.</td>
</tr>
<tr>
<td>p$</td>
<td>It matches any string with p at the end of it.</td>
</tr>
<tr>
<td>^p</td>
<td>It matches any string with p at the beginning of it.</td>
</tr>
</tbody>
</table>
Examples
Following examples explain more about matching characters.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[^a-zA-Z]</td>
<td>It matches any string not containing any of the characters ranging from a through z and A through Z.</td>
</tr>
<tr>
<td>p.p</td>
<td>It matches any string containing p, followed by any character, in turn followed by another p.</td>
</tr>
<tr>
<td>^.{2}$</td>
<td>It matches any string containing exactly two characters.</td>
</tr>
<tr>
<td>&lt;b&gt;(.*)&lt;b&gt;</td>
<td>It matches any string enclosed within &lt;b&gt; and &lt;/b&gt;.</td>
</tr>
<tr>
<td>p(hp)*</td>
<td>It matches any string containing a p followed by zero or more instances of the sequence hp.</td>
</tr>
</tbody>
</table>

Literal Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphanumeric</td>
<td>Itself</td>
</tr>
<tr>
<td>\0</td>
<td>The NUL character (\u0000)</td>
</tr>
<tr>
<td>\t</td>
<td>Tab (\u0009)</td>
</tr>
<tr>
<td>\n</td>
<td>Newline (\u000A)</td>
</tr>
<tr>
<td>\v</td>
<td>Vertical tab (\u000B)</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed (\u000C)</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return (\u000D)</td>
</tr>
<tr>
<td>\xnn</td>
<td>The Latin character specified by the hexadecimal number nn; for example, \x0A is the same as \n</td>
</tr>
<tr>
<td>\uxxxx</td>
<td>The Unicode character specified by the hexadecimal number xxxx; for example, \u0009 is the same as \t</td>
</tr>
</tbody>
</table>
Metacharacters

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for a large sum of money using the \d metacharacter: /([\d]+)000/. Here \d will search for any string of numerical character.

The following table lists a set of metacharacters which can be used in PERL Style Regular Expressions.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>a single character</td>
</tr>
<tr>
<td>\s</td>
<td>a whitespace character (space, tab, newline)</td>
</tr>
<tr>
<td>\S</td>
<td>non-whitespace character</td>
</tr>
<tr>
<td>\d</td>
<td>a digit (0-9)</td>
</tr>
<tr>
<td>\D</td>
<td>a non-digit</td>
</tr>
<tr>
<td>\w</td>
<td>a word character (a-z, A-Z, 0-9, _)</td>
</tr>
<tr>
<td>\W</td>
<td>a non-word character</td>
</tr>
<tr>
<td>[b]</td>
<td>a literal backspace (special case).</td>
</tr>
<tr>
<td>[aeiou]</td>
<td>matches a single character in the given set</td>
</tr>
<tr>
<td>[^aeiou]</td>
<td>matches a single character outside the given set</td>
</tr>
<tr>
<td>(foo</td>
<td>bar</td>
</tr>
</tbody>
</table>

Modifiers

Several modifiers are available that can simplify the way you work with regexps, like case sensitivity, searching in multiple lines, etc.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Performs case-insensitive matching.</td>
</tr>
</tbody>
</table>
### Javascript

<table>
<thead>
<tr>
<th>m</th>
<th>Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Performs a global match that is, find all matches rather than stopping after the first match.</td>
</tr>
</tbody>
</table>

### RegExp Properties

Here is a list of the properties associated with RegExp and their description.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>Specifies the function that creates an object's prototype.</td>
</tr>
<tr>
<td>global</td>
<td>Specifies if the &quot;g&quot; modifier is set.</td>
</tr>
<tr>
<td>ignoreCase</td>
<td>Specifies if the &quot;i&quot; modifier is set.</td>
</tr>
<tr>
<td>lastIndex</td>
<td>The index at which to start the next match.</td>
</tr>
<tr>
<td>multiline</td>
<td>Specifies if the &quot;m&quot; modifier is set.</td>
</tr>
<tr>
<td>source</td>
<td>The text of the pattern.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of RegExp properties.

**constructor**

It returns a reference to the array function that created the instance's prototype.

**Syntax**

Its syntax is as follows:

```javascript
RegExp.constructor
```

**Return Value**

Returns the function that created this object's instance.
Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript RegExp constructor Property</title>
</head>
<body>
<script type="text/javascript">
    var re = new RegExp( "string" );
    document.write("re.constructor is:" + re.constructor);
</script>
</body>
</html>
```

Output

```
re.constructor is:function RegExp() { [native code]
```

**global**

`global` is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs global matching, i.e., whether it was created with the "g" attribute.

**Syntax**

Its syntax is as follows:

```
RegExpObject.global
```

**Return Value**

Returns "TRUE" if the "g" modifier is set, "FALSE" otherwise.

**Example**
Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp global Property</title>
</head>
<body>
<script type="text/javascript">
  var re = new RegExp( "string" );

  if ( re.global ){
    document.write("Test1 - Global property is set");
  }else{
    document.write("Test1 - Global property is not set");
  }
  re = new RegExp( "string", "g" );

  if ( re.global ){
    document.write("<br />Test2 - Global property is set");
  }else{
    document.write("<br />Test2 - Global property is not set");
  }
</script>
</body>
</html>
```

**Output**

Test1 - Global property is not set  
Test2 - Global property is set

**ignoreCase**

*ignoreCase* is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs case-insensitive matching, i.e., whether it was created with the "i" attribute.
Syntax
Its syntax is as follows:

```
RegExpObject.ignoreCase
```

Return Value
Returns "TRUE" if the "i" modifier is set, "FALSE" otherwise.

Example
Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp ignoreCase Property</title>
</head>
<body>
<script type="text/javascript">
    var re = new RegExp( "string" );

    if ( re.ignoreCase ){
        document.write("Test1 - ignoreCase property is set");
    }else{
        document.write("Test1 - ignoreCase property is not set");
    }
    re = new RegExp( "string", "i" );

    if ( re.ignoreCase ){
        document.write("<br />Test2 - ignoreCase property is set");
    }else{
        document.write("<br />Test2 - ignoreCase property is not set");
    }
</script>
</body>
</html>
```
lastIndex

lastIndex is a read/write property of RegExp objects. For regular expressions with the "g" attribute set, it contains an integer that specifies the character position immediately following the last match found by the RegExp.exec() and RegExp.test() methods. These methods use this property as the starting point for the next search they conduct.

This property allows you to call those methods repeatedly, to loop through all matches in a string and works only if the "g" modifier is set.

This property is read/write, so you can set it at any time to specify where in the target string, the next search should begin. exec() and test() automatically reset the lastIndex to 0 when they fail to find a match (or another match).

Syntax

Its syntax is as follows:

RegExpObject.lastIndex

Return Value

Returns an integer that specifies the character position immediately following the last match.

Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript RegExp lastIndex Property</title>
</head>
<body>
<script type="text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp("script", "g");

    re.test(str);
</script>
</body>
</html>
```
multiline

**multiline** is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs multiline matching, i.e., whether it was created with the "m" attribute.

**Syntax**

Its syntax is as follows:

```
RegExpObject.multiline
```

**Return Value**

Returns "TRUE" if the "m" modifier is set, "FALSE" otherwise.

**Example**

Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp multiline Property</title>
</head>
<body>
<script type="text/javascript">
    var re = new RegExp( "string" );
</script>
</body>
</html>
```
if ( re.multiline ){
    document.write("Test1-multiline property is set");
}else{
    document.write("Test1-multiline property is not set");
}
re = new RegExp( "string", "m" );
if ( re.multiline ){
    document.write("<br/>Test2-multiline property is set");
}else{
    document.write("<br/>Test2-multiline property is not set");
}
</script>
</body>
</html>

Output

Test1-multiline property is not set
Test2-multiline property is set

source

source is a read-only string property of RegExp objects. It contains the text of the RegExp pattern. This text does not include the delimiting slashes used in regular-expression literals, and it does not include the "g", "i", and "m" attributes.

Syntax

Its syntax is as follows:

RegExpObject.source

Return Value

Returns the text used for pattern matching.
Try the following example program.

```html
<html>
<head>
<title>JavaScript RegExp source Property</title>
</head>
<body>
<script type="text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp( "script", "g" );

    re.test(str);
    document.write("The regular expression is : " + re.source);
</script>
</body>
</html>
```

Output

```
The regular expression is : script
```

**RegExp Methods**

Here is a list of the methods associated with RegExp along with their description.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exec()</td>
<td>Executes a search for a match in its string parameter.</td>
</tr>
<tr>
<td>test()</td>
<td>Tests for a match in its string parameter.</td>
</tr>
<tr>
<td>toSource()</td>
<td>Returns an object literal representing the specified object; you can use this value to create a new object.</td>
</tr>
<tr>
<td>toString()</td>
<td>Returns a string representing the specified object.</td>
</tr>
</tbody>
</table>

In the following sections, we will have a few examples to demonstrate the usage of RegExp methods.
**exec()**

The `exec` method searches string for text that matches regexp. If it finds a match, it returns an array of results; otherwise, it returns null.

**Syntax**

Its syntax is as follows:

```
RegExpObject.exec( string );
```

**Parameter Details**

**string**: The string to be searched.

**Return Value**

Returns the matched text if a match is found, and null if not.

**Example**

Try the following example program.

```html
<html>
<head>
<title>JavaScript RegExp exec Method</title>
</head>
<body>
<script type="text/javascript">
var str = "Javascript is an interesting scripting language";
var re = new RegExp( "script", "g" );

var result = re.exec(str);
document.write("Test 1 - returned value : " + result);

re = new RegExp( "pushing", "g" );
var result = re.exec(str);
document.write("<br />Test 2 - returned value : " + result);
</script>
</body>
</html>
```
Output

<table>
<thead>
<tr>
<th>Test 1 - returned value : script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 2 - returned value : null</td>
</tr>
</tbody>
</table>

test ()

The **test** method searches string for text that matches regexp. If it finds a match, it returns true; otherwise, it returns false.

Syntax

Its syntax is as follows:

```
RegExpObject.test( string );
```

Parameter Details

**string**: The string to be searched.

Return Value

Returns the matched text if a match is found, and null if not.

Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript RegExp test Method</title>
</head>
<body>
<script type="text/javascript">
 var str = "Javascript is an interesting scripting language";
 var re = new RegExp( "script", "g" );

 var result = re.test(str);
 document.write("Test 1 - returned value : " + result);

 re = new RegExp( "pushing", "g" );
 var result = re.test(str);
</script>
</body>
</html>
```
document.write("<br />Test 2 - returned value : " + result);
</script>
</body>
</html>

Output

Test 1 - returned value : true
Test 2 - returned value : false

toSource()

The `toSource` method string represents the source code of the object. This method does not work with all the browsers.

Syntax

Its syntax is as follows:

```
RegExpObject.toSource(string);
```

Return Value

Returns the string representing the source code of the object.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp toSource Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp( "script", "g" );

    var result = re.toSource(str);
    document.write("Test 1 - returned value : " + result);
</script>
</body>
</html>
```
Output

Test 1 - returned value : /script/g
Test 2 - returned value : /\//g

toString ()

The **toString** method returns a string representation of a regular expression in the form of a regular-expression literal.

Syntax

Its syntax is as follows:

```javascript
RegExpObject.toString ( );
```

Return Value

Returns the string representing of a regular expression.

Example

Try the following example program.

```html
<html>
<head>
<title>JavaScript RegExp toString Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp( "script", "g" );

    var result = re.toString(str);

    document.write("<br />Test 2 - returned value : " + result);
</script>
</body>
</html>
```
document.write("Test 1 - returned value : " + result);

re = new RegExp("/", "g");
var result = re.toString(str);
document.write(<br />"Test 2 - returned value : " + result);
</script>
</body>
</html>

Output

Test 1 - returned value : /script/g
Test 2 - returned value : /\//g
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or DOM. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object**: Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object**: Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object**: Everything enclosed in the `<form>...</form>` tags sets the form object.
- **Form control elements**: The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects:
There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

- **The Legacy DOM**: This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.

- **The W3C DOM**: This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.

- **The IE4 DOM**: This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

### The Legacy DOM

This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.

This model provides several read-only properties, such as title, URL, and lastModified provide information about the document as a whole. Apart from that, there are various methods provided by this model which can be used to set and get document property values.

### Document Properties in Legacy DOM

Here is a list of the document properties which can be accessed using Legacy DOM.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>alinkColor</code></td>
</tr>
<tr>
<td></td>
<td>Deprecated - A string that specifies the color of activated links.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.alinkColor</td>
</tr>
<tr>
<td>2</td>
<td><code>anchors[ ]</code></td>
</tr>
<tr>
<td></td>
<td>An array of Anchor objects, one for each anchor that appears in the document</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.anchors[0], document.anchors[1] and so on</td>
</tr>
<tr>
<td>3</td>
<td><strong>applets[ ]</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.applets[0], document.applets[1] and so on</td>
</tr>
<tr>
<td>4</td>
<td><strong>bgColor</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.bgColor</td>
</tr>
<tr>
<td>5</td>
<td><strong>Cookie</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.cookie</td>
</tr>
<tr>
<td>6</td>
<td><strong>Domain</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.domain</td>
</tr>
<tr>
<td>7</td>
<td><strong>embeds[ ]</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.embeds[0], document.embeds[1] and so on</td>
</tr>
<tr>
<td>8</td>
<td><strong>fgColor</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.fgColor</td>
</tr>
<tr>
<td>#</td>
<td>Property</td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
</tr>
<tr>
<td>9</td>
<td>forms[]</td>
</tr>
<tr>
<td>10</td>
<td>images[]</td>
</tr>
<tr>
<td>11</td>
<td>lastModified</td>
</tr>
<tr>
<td>12</td>
<td>linkColor</td>
</tr>
<tr>
<td>13</td>
<td>links[]</td>
</tr>
<tr>
<td>14</td>
<td>Location</td>
</tr>
<tr>
<td>15</td>
<td>plugins[]</td>
</tr>
</tbody>
</table>
### Javascript

**Ex:** document.plugins[0], document.plugins[1] and so on

<table>
<thead>
<tr>
<th>16</th>
<th><strong>Referrer</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A read-only string that contains the URL of the document, if any, from which the current document was linked.</td>
<td></td>
</tr>
<tr>
<td><strong>Ex:</strong> document.referrer</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17</th>
<th><strong>Title</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The text contents of the <code>&lt;title&gt;</code> tag.</td>
<td></td>
</tr>
<tr>
<td><strong>Ex:</strong> document.title</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>18</th>
<th><strong>URL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A read-only string that specifies the URL of the document.</td>
<td></td>
</tr>
<tr>
<td><strong>Ex:</strong> document.URL</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19</th>
<th><strong>vlinkColor</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Deprecated - A string that specifies the color of visited links.</td>
<td></td>
</tr>
<tr>
<td><strong>Ex:</strong> document.vlinkColor</td>
<td></td>
</tr>
</tbody>
</table>

### Document Methods in Legacy DOM

Here is a list of methods supported by Legacy DOM.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>clear( )</code></td>
</tr>
<tr>
<td></td>
<td>Deprecated - Erases the contents of the document and returns nothing.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.clear( )</td>
</tr>
<tr>
<td>2</td>
<td><code>close( )</code></td>
</tr>
<tr>
<td></td>
<td>Closes a document stream opened with the <code>open( )</code> method and returns nothing.</td>
</tr>
</tbody>
</table>
Example

We can locate any HTML element within any HTML document using HTML DOM. For instance, if a web document contains a form element, then using JavaScript, we can refer to it as document.forms[0]. If your Web document includes two form elements, the first form is referred to as document.forms[0] and the second as document.forms[1].

Using the hierarchy and properties given above, we can access the first form element using document.forms[0].elements[0] and so on.

Here is an example to access document properties using Legacy DOM method.

```html
<html>
<head>
<title>Document Title</title>
<script type="text/javascript">
!--
function myFunc()
```
```javascript
{
    var ret = document.title;
    alert("Document Title : " + ret );

    var ret = document.URL;
    alert("Document URL : " + ret );

    var ret = document.forms[0];
    alert("Document First Form : " + ret );

    var ret = document.forms[0].elements[1];
    alert("Second element : " + ret );
}
//-->
</script>
</head>
<body>
<h1 id="title">This is main title</h1>
<p>Click the following to see the result:</p>
<form name="FirstForm">
<input type="button" value="Click Me" onclick="myFunc();" />
<input type="button" value="Cancel">
</form>

<form name="SecondForm">
<input type="button" value="Don't ClickMe"/>
</form>

</body>
</html>

Output

331
NOTE: This example returns objects for forms and elements and we would have to access their values by using those object properties which are not discussed in this tutorial.

The W3C DOM

This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.

The W3C DOM standardizes most of the features of the legacy DOM and adds new ones as well. In addition to supporting forms[], images[], and other array properties of the Document object, it defines methods that allow scripts to access and manipulate any document element and not just special-purpose elements like forms and images.

Document Properties in W3C DOM

This model supports all the properties available in Legacy DOM. Additionally, here is a list of document properties which can be accessed using W3C DOM.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Body</strong>&lt;br&gt;A reference to the Element object that represents the &lt;body&gt; tag of this document.&lt;br&gt;&lt;br&gt;<strong>Ex:</strong> document.body</td>
</tr>
<tr>
<td>2</td>
<td><strong>defaultView</strong>&lt;br&gt;It is a read-only property and represents the window in which the document is displayed.</td>
</tr>
</tbody>
</table>
**Document Methods in W3C DOM**

This model supports all the methods available in Legacy DOM. Additionally, here is a list of methods supported by W3C DOM.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>createAttribute(name)</code></td>
</tr>
<tr>
<td></td>
<td>Returns a newly-created Attr node with the specified name.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> <code>document.createAttribute(name)</code></td>
</tr>
<tr>
<td>2</td>
<td><code>createComment(text)</code></td>
</tr>
<tr>
<td></td>
<td>Creates and returns a new Comment node containing the specified text.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> <code>document.createComment(text)</code></td>
</tr>
<tr>
<td>3</td>
<td><code>createDocumentFragment()</code></td>
</tr>
<tr>
<td></td>
<td>Creates and returns an empty DocumentFragment node.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> <code>document.createDocumentFragment()</code></td>
</tr>
</tbody>
</table>

**Ex:** `document.defaultView`

**documentElement**

A read-only reference to the `<html>` tag of the document.

**Ex:** `document.documentElement` 8/31/2008

**Implementation**

It is a read-only property and represents the DOMImplementation object that represents the implementation that created this document.

**Ex:** `document.implementation`
<table>
<thead>
<tr>
<th></th>
<th>Javascript</th>
</tr>
</thead>
</table>
| 4 | `createElement(tagName)`  
Creates and returns a new Element node with the specified tag name.  
**Ex:** `document.createElement(tagName)` |
| 5 | `createTextNode(text)`  
Creates and returns a new Text node that contains the specified text.  
**Ex:** `document.createTextNode(text)` |
| 6 | `getElementById(id)`  
Returns the Element of this document that has the specified value for its id attribute, or null if no such Element exists in the document.  
**Ex:** `document.getElementById(id)` |
| 7 | `getElementsByTagName(tagName)`  
Returns an array of all Element nodes in this document that have the specified tag name. The Element nodes appear in the returned array in the same order they appear in the document source.  
**Ex:** `document.getElementsByTagName(tagName)` |
| 8 | `getElementsByName(name)`  
Returns an array of nodes of all elements in the document that have a specified value for their name attribute. If no such elements are found, returns a zero-length array.  
**Ex:** `document.getElementsByName(name)` |
| 9 | `importNode(importedNode, deep)`  
Creates and returns a copy of a node from some other document that is suitable for insertion into this document. If the deep argument is true, it recursively copies the children of the node too. Supported in DOM Version 2 |
Example
This is very easy to manipulate (Accessing and Setting) document element using W3C DOM. You can use any of the methods like `getElementById`, `getElementsByClassName`, or `getElementsByTagName`.

Here is an example to access document properties using W3C DOM method.

```html
<html>
<head>
<title>Document Title</title>
<script type="text/javascript">
<!--
function myFunc()
{
    var ret = document.getElementsByTagName("title");
    alert("Document Title : " + ret[0].text);

    var ret = document.getElementById("heading");
    alert("Document URL : " + ret.innerHTML);
}
//-->
</script>
</head>
<body>
<h1 id="heading">This is main title</h1>
<p>Click the following to see the result:</p>
<form id="form1" name="FirstForm">
<input type="button" value="Click Me" onclick="myFunc();" />
<input type="button" value="Cancel" />
</form>
</form>
</body>
</html>
```
<input type="button" value="Don't ClickMe"/>

</form>

</body>
</html>

NOTE: This example returns objects for forms and elements and we would have to access their values by using those object properties which are not discussed in this tutorial.

Output

This is main title

Click the following to see the result:

[Click Me]  [Cancel]

[Don't Click Me]

The IE 4 DOM

This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

Document Properties in IE 4 DOM

The following non-standard (and non-portable) properties are defined by Internet Explorer 4 and later versions.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>activeElement</td>
</tr>
<tr>
<td></td>
<td>A read-only property that refers to the input element that is currently active (i.e., has the input focus).</td>
</tr>
<tr>
<td></td>
<td>Ex: document.activeElement</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| 2 | **all[ ]** | An array of all Element objects within the document. This array may be indexed numerically to access elements in source order, or it may be indexed by element id or name.  
**Ex:** document.all[ ] |
| 3 | **Charset** | The character set of the document.  
**Ex:** document.charset |
| 4 | **children[ ]** | An array that contains the HTML elements that are the direct children of the document. Note that this is different from the all [ ] array that contains all the elements in the document, regardless of their position in the containment hierarchy.  
**Ex:** document.children[ ] |
| 5 | **defaultCharset** | The default character set of the document.  
**Ex:** document.defaultCharset |
| 6 | **expand** | This property, if set to false, prevents client-side objects from being expanded.  
**Ex:** document.expando |
| 7 | **parentWindow** | The window that contains the document.  
**Ex:** document.parentWindow |
| 8 | **readyState** |   |
Specifies the loading status of a document. It has one of the following four string values:

**Ex:** document.readyState

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td><strong>Uninitialized</strong></td>
</tr>
<tr>
<td></td>
<td>The document has not started loading.</td>
</tr>
<tr>
<td></td>
<td>Example: document.uninitialized</td>
</tr>
<tr>
<td>10</td>
<td><strong>Loading</strong></td>
</tr>
<tr>
<td></td>
<td>The document is loading.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.loading</td>
</tr>
<tr>
<td>11</td>
<td><strong>interactive</strong></td>
</tr>
<tr>
<td></td>
<td>The document has loaded sufficiently for the user to interact with it.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.interactive</td>
</tr>
<tr>
<td>12</td>
<td><strong>complete</strong></td>
</tr>
<tr>
<td></td>
<td>The document is completely loaded.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.complete</td>
</tr>
</tbody>
</table>

**Document Methods in IE4 DOM**

This model supports all the methods available in Legacy DOM. Additionally, here is a list of methods supported by IE4 DOM.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>elementFromPoint(x,y)</strong></td>
</tr>
<tr>
<td></td>
<td>Returns the Element located at a specified point.</td>
</tr>
<tr>
<td></td>
<td><strong>Ex:</strong> document.elementFromPoint(x,y)</td>
</tr>
</tbody>
</table>
Example

The IE 4 DOM does not support the `getElementById()` method. Instead, it allows you to look up arbitrary document elements by `id` attribute within the `all []` array of the document object.

Here's how to find all `<li>` tags within the first `<ul>` tag. Note that you must specify the desired HTML tag name in uppercase with the `all.tags()` method.

```javascript
var lists = document.all.tags("UL");
var items = lists[0].all.tags("LI");
```

Here is another example to access document properties using IE4 DOM method.

```html
<html>
<head>
<title> Document Title </title>
<script type="text/javascript">
<!--
function myFunc()
{
    var ret = document.all["heading"];  
    alert("Document Heading : " + ret.innerHTML  );

    var ret = document.all.tags("P");
    alert("First Paragraph : " + ret[0].innerHTML);  
}
//-->
</script>
</head>
<body>
<h1 id="heading">This is main title</h1>
<p>Click the following to see the result:</p>
<form id="form1" name="FirstForm">
    <input type="button" value="Click Me" onclick="myFunc()"/>
    <input type="button" value="Cancel"/>
</form>
</body>
</html>
```
NOTE: This example returns objects for forms and elements and we would have to access their values by using those object properties which are not discussed in this tutorial.

Output

This is main title

Click the following to see the result:

- [Click Me](#)
- [Cancel](#)
- [Don't Click Me](#)

DOM Compatibility

If you want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then you can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example:

```javascript
if (document.getElementById) {
    // If the W3C method exists, use it
}
else if (document.all) {
    // If the all[] array exists, use it
}
else {
```
// Otherwise use the legacy DOM
}

Part 3: JavaScript Advanced
There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

**Syntax Errors**

Syntax errors, also called **parsing errors**, occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

```javascript
<script type="text/javascript">
<!--
    window.print();
//-->
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

**Runtime Errors**

Runtime errors, also called **exceptions**, occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```javascript
<script type="text/javascript">
<!--
    window.printme();
//-->
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.
Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.

You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

Here is the try...catch...finally block syntax:

```javascript
<script type="text/javascript">
<!--
try {
   // Code to run
   [break;]
} catch ( e ) {
   // Code to run if an exception occurs
   [break;]
}[ finally {
   // Code that is always executed regardless of
   // an exception occurring
}
//-->
</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the try block, the exception is placed in e and the catch block is executed. The optional finally block executes unconditionally after try/catch.
Example
Here is an example where we are trying to call a non-existing function which in turn is raising an exception. Let us see how it behaves without try...catch.

```html
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;

    document.write("Value of variable a is : " + a);
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
    <input type="button" value="Click Me" onclick="myFunc();" />
</form>
<p>Error will happen and depending on your browser it will give different result.</p>
</body>
</html>
```

Output

Click the following to see the result:

Click Me

Error will happen and depending on your browser it will give different result.
Now let us try to catch this exception using **try...catch** and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

```html
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;

    try {
        document.write ("Value of variable a is : " + a);
    } catch ( e ) {
        document.write ("Error: " + e.description);
    }
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
</body>
</html>
```

**Output**

Click the following to see the result:

Click Me
You can use a **finally** block which will always execute unconditionally after the try/catch. Here is an example.

**Example**

```html
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;

    try {
        document.write("Value of variable a is : " + a);
    }catch ( e ){
        document.write("Error: " + e.description);
    }finally {
        document.write("Finally block will always execute!");
    }
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
<p>Try running after fixing the problem with method name.</p>
</body>
</html>
```
The throw Statement

You can use a `throw` statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

Example

The following example demonstrates how to use a `throw` statement.

```html
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;
    var b = 0;

    try{
        if ( b == 0 ){
            throw( "Divide by zero error." );
        }else{
            var c = a / b;
        }
    }catch ( e ) {
        document.write ("Error: " + e);
    }
}
//-->
</script>
</head>
</html>
```

Click the following to see the result:

Click Me

Try running after fixing the problem with method name.
You can raise an exception in one function using a string, integer, Boolean, or an object and then you can capture that exception either in the same function as we did above, or in another function using a `try...catch` block.

## The `onerror()` Method

The `onerror` event handler was the first feature to facilitate error handling in JavaScript. The `error` event is fired on the window object whenever an exception occurs on the page.

### Example

```html
<html>
<head>
<script type="text/javascript">
<!--
window.onerror = function () {
  document.write ("An error occurred.");
}
//-->
</script>
</head>
<body>
</body>
</html>
```
The `onerror` event handler provides three pieces of information to identify the exact nature of the error:

- **Error message**: The same message that the browser would display for the given error
- **URL**: The file in which the error occurred
- **Line number**: The line number in the given URL that caused the error

Here is the example to show how to extract this information.

Example

```html
<html>
<head>
<script type="text/javascript">
<!--
window.onerror = function (msg, url, line) {
    document.write ("Message : " + msg );
    document.write ("url : " + url );
    document.write ("Line number : " + line );
}
//-->
</script>
</head>
<body>
Click Me
</body>
</html>
```
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
</body>
</html>

Output

Click the following to see the result:

Click Me

You can display extracted information in whatever way you think it is better.

You can use an `<onerror>` method, as shown below, to display an error message in case there is any problem in loading an image.

```html
<img src="myimage.gif"
     onerror = "alert('An error occurred loading the image.')" />
```

You can use `<onerror>` with many HTML tags to display appropriate messages in case of errors.
Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** - First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.

- **Data Format Validation** - Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

**Example**

We will take an example to understand the process of validation. Here is a simple form in html format.

```html
<html>
<head>
<title>Form Validation</title>
<script type="text/javascript">
<!--
// Form validation code will come here.
//-->
</script>
</head>
<body>
<form action="/cgi-bin/test.cgi" name="myForm"
     onsubmit="return(validate());">
<table cellspacing="2" cellpadding="2" border="1">
  <tr>
    <td align="right">Name</td>
    <td><input type="text" name="name"></td>
  </tr>
</table>
</form>
</body>
</html>
```
<td><input type="text" name="Name" /></td>
</tr>
<tr>
<td align="right">EMail</td>
<td><input type="text" name="EMail" /></td>
</tr>
<tr>
<td align="right">Zip Code</td>
<td><input type="text" name="Zip" /></td>
</tr>
<tr>
<td align="right">Country</td>
<td><select name="Country">
<option value="-1" selected>[choose yours]</option>
<option value="1">USA</option>
<option value="2">UK</option>
<option value="3">INDIA</option>
</select></td>
</tr>
<tr>
<td align="right"></td>
<td><input type="submit" value="Submit" /></td>
</tr>
</table>
</form>
Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling `validate()` to validate data when `onsubmit` event is occurring. The following code shows the implementation of this `validate()` function.

```html
<script type="text/javascript">
<!--
// Form validation code will come here.
function validate()
{

    if( document.myForm.Name.value == "" )
    {
        alert( "Please provide your name!" );
        document.myForm.Name.focus();
        return false;
    }
    if( document.myForm.EMail.value == "" )
    {
        alert( "Please provide your Email!" );
        document.myForm.EMail.focus();
        return false;
    }
// Form validation code will come here.
    return true;
}
</script>
```
Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example

Try the following code for email validation.

```javascript
<script type="text/javascript">
<!--
function validateEmail()
{

```
var emailID = document.myForm.EMail.value;
atpos = emailID.indexOf("@");
dotpos = emailID.lastIndexOf(".");
if (atpos < 1 || ( dotpos - atpos < 2 ))
{
    alert("Please enter correct email ID")
    document.myForm.EMail.focus();
    return false;
}
return( true );

//-->
</script>
You can use JavaScript to create a complex animation having, but not limited to, the following elements:

- Fireworks
- Fade Effect
- Roll-in or Roll-out
- Page-in or Page-out
- Object movements

You might be interested in existing JavaScript based animation library: Script.Aculo.us.

This tutorial provides a basic understanding of how to use JavaScript to create an animation.

JavaScript can be used to move a number of DOM elements (<img />, <div>, or any other HTML element) around the page according to some sort of pattern determined by a logical equation or function.

JavaScript provides the following two functions to be frequently used in animation programs.

- `setTimeout(function, duration)` - This function calls `function` after `duration` milliseconds from now.
- `setInterval(function, duration)` - This function calls `function` after every `duration` milliseconds.
- `clearTimeout(setTimeout_variable)` - This function clears any timer set by the setTimeout() function.

JavaScript can also set a number of attributes of a DOM object including its position on the screen. You can set `top` and `left` attribute of an object to position it anywhere on the screen. Here is its syntax.

```javascript
// Set distance from left edge of the screen.
object.style.left = distance in pixels or points;

or

// Set distance from top edge of the screen.
object.style.top = distance in pixels or points;
```
Manual Animation

So let's implement one simple animation using DOM object properties and JavaScript functions as follows. The following list contains different DOM methods.

- We are using the JavaScript function `getElementById()` to get a DOM object and then assigning it to a global variable `imgObj`.
- We have defined an initialization function `init()` to initialize `imgObj` where we have set its `position` and `left` attributes.
- We are calling initialization function at the time of window load.
- Finally, we are calling `moveRight()` function to increase the left distance by 10 pixels. You could also set it to a negative value to move it to the left side.

Example

Try the following example.

```html
<html>
<head>
<title>JavaScript Animation</title>
<script type="text/javascript">
<!--
var imgObj = null;
function init(){
    imgObj = document.getElementById('myImage');
    imgObj.style.position= 'relative';
    imgObj.style.left = '0px';
}
function moveRight(){
    imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
}
window.onload =init;
//-->
</script>
</head>
<body>
</body>
```
Automated Animation

In the above example, we saw how an image moves to right with every click. We can automate this process by using the JavaScript function `setTimeout()` as follows.

Here we have added more methods. So let’s see what is new here:

- The `moveRight()` function is calling `setTimeout()` function to set the position of `imgObj`.
- We have added a new function `stop()` to clear the timer set by `setTimeout()` function and to set the object at its initial position.

Example

Try the following example code.

```html
<html>
<head>
<title>JavaScript Animation</title>
<script type="text/javascript">
<!--
var imgObj = null;
var animate ;
function init(){
    imgObj = document.getElementById('myImage');
    imgObj.style.position= 'relative';
    imgObj.style.left = '0px';

    function moveRight(){
        imgObj.style.left = '200px';
        setTimeout("moveRight()", 1000);
    }

    function stop(){
        imgObj.style.left = '0px';
        imgObj.style.position= 'static';
    }

    document.getElementById('myButton').onclick =  moveRight;
    document.getElementById('myStop').onclick = stop;

    init();
}
</script>
</head>
<body>
<form>
<img id="myImage" src="/images/html.gif" />
<p>Click button below to move the image to right</p>
<input type="button" value="Click Me" onclick="moveRight();" />
</form>
</body>
</html>
```
function moveRight()
{
    imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
    animate = setTimeout(moveRight,20); // call moveRight in 20msec
}
function stop()
{
    clearTimeout(animate);
    imgObj.style.left = '0px';
}
window.onload =init;
//-->
</script>
</head>
<body>
<form>
<img id="myImage" src="/images/html.gif" />
<p>Click the buttons below to handle animation</p>
<input type="button" value="Start" onclick="moveRight();" />
<input type="button" value="Stop" onclick="stop();" />
</form>
</body>
</html>

It is not possible to show animation in this tutorial. But you can [Try it here.]

**Rollover with a Mouse Event**

Here is a simple example showing image rollover with a mouse event.

Let's see what we are using in the following example:

- At the time of loading this page, the 'if' statement checks for the existence of the image object. If the image object is unavailable, this block will not be executed.
- The `Image()` constructor creates and preloads a new image object called `image1`.
• The src property is assigned the name of the external image file called /images/html.gif.

• Similarly, we have created image2 object and assigned /images/http.gif in this object.

• The # (hash mark) disables the link so that the browser does not try to go to a URL when clicked. This link is an image.

• The onMouseOver event handler is triggered when the user's mouse moves onto the link, and the onMouseOut event handler is triggered when the user's mouse moves away from the link (image).

• When the mouse moves over the image, the HTTP image changes from the first image to the second one. When the mouse is moved away from the image, the original image is displayed.

• When the mouse is moved away from the link, the initial image html.gif will reappear on the screen.

```html
<html>
<head>
<title>Rollover with a Mouse Events</title>
<script type="text/javascript">
-->
if(document.images){
   var image1 = new Image();     // Preload an image
   image1.src = "/images/html.gif";
   var image2 = new Image();     // Preload second image
   image2.src = "//images/http.gif";
}
//-->
</script>
</head>
<body>
<p>Move your mouse over the image to see the result</p>
<a href="#" onMouseOver="document.myImage.src=image2.src;"
   onMouseOut="document.myImage.src=image1.src;">
<img name="myImage" src="/images/html.gif" />
</a>
</body>
```
It is not possible to show animation in this tutorial. But you can [Try it here.](#)
The JavaScript `navigator` object includes a child object called `plugins`. This object is an array, with one entry for each plug-in installed on the browser. The `navigator.plugins` object is supported only by Netscape, Firefox, and Mozilla only.

**Example**

Here is an example that shows how to list down all the plug-ins installed in your browser:

```html
<html>
<head>
<title>List of Plug-Ins</title>
</head>
<body>
<table border="1">
<tr><th>Plug-in Name</th><th>Filename</th><th>Description</th></tr>
<script LANGUAGE="JavaScript" type="text/javascript">
for (i=0; i<navigator.plugins.length; i++) {
    document.write("<tr><td>");
    document.write(navigator.plugins[i].name);
    document.write("</td><td>");
    document.write(navigator.plugins[i].filename);
    document.write("</td><td>");
    document.write(navigator.plugins[i].description);
    document.write("</td></tr>");
}
</script>
</table>
</body>
</html>
```
Checking for Plug-Ins

Each plug-in has an entry in the array. Each entry has the following properties:

- **name** - is the name of the plug-in.
- **filename** - is the executable file that was loaded to install the plug-in.
- **description** - is a description of the plug-in, supplied by the developer.
- **mimeTypes** - is an array with one entry for each MIME type supported by the plug-in.

You can use these properties in a script to find out the installed plug-ins, and then using JavaScript, you can play appropriate multimedia file. Take a look at the following example.

```html
<html>
<head>
<title>Using Plug-Ins</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
media = navigator.mimeTypes["video/quicktime"]; if (media){
   document.write("<embed src='quick.mov' height=100 width=100>");
}
else{
```

Output

![List of Plug-Ins](image)
NOTE: Here we are using HTML `<embed>` tag to embed a multimedia file.

**Controlling Multimedia**

Let us take a real example which works in almost all the browsers.

```html
<html>
<head>
<title>Using Embedded Object</title>
<script type="text/javascript">
<!--
function play()
{
  if (!document.demo(IsPlaying())){
    document.demo.Play();
  }
}

function stop()
{
  if (document.demo.IsPlaying()){
    document.demo.StopPlay();
  }
}

function rewind()
  if (document.demo.IsPlaying()){
    document.demo.StopPlay();
  }
  document.demo.Rewind();

```
If you are using Mozilla, Firefox or Netscape, then Try it yourself.
Every now and then, developers commit mistakes while coding. A mistake in a program or a script is referred to as a bug.

The process of finding and fixing bugs is called debugging and is a normal part of the development process. This section covers tools and techniques that can help you with debugging tasks.

**Error Messages in IE**

The most basic way to track down errors is by turning on error information in your browser. By default, Internet Explorer shows an error icon in the status bar when an error occurs on the page.

Double-clicking this icon takes you to a dialog box showing information about the specific error that occurred.

Since this icon is easy to overlook, Internet Explorer gives you the option to automatically show the Error dialog box whenever an error occurs.

To enable this option, select **Tools --> Internet Options --> Advanced tab** and then finally check the “Display a Notification about Every Script Error” box option as shown below.
Error Messages in Firefox or Mozilla

Other browsers like Firefox, Netscape, and Mozilla send error messages to a special window called the JavaScript Console or Error Console. To view the console, select Tools --&gt; Error Console or Web Development.

Unfortunately, since these browsers give no visual indication when an error occurs, you must keep the Console open and watch for errors as your script executes.
**Error Notifications**

Error notifications that show up on Console or through Internet Explorer dialog boxes are the result of both syntax and runtime errors. These error notification include the line number at which the error occurred.

If you are using Firefox, then you can click on the error available in the error console to go to the exact line in the script having error.

**How to Debug a Script**

There are various ways to debug your JavaScript:

**Use a JavaScript Validator**

One way to check your JavaScript code for strange bugs is to run it through a program that checks it to make sure it is valid and that it follows the official syntax rules of the language. These programs are called **validating parsers** or just **validators** for short, and often come with commercial HTML and JavaScript editors.

The most convenient validator for JavaScript is Douglas Crockford's JavaScript Lint, which is available for free at [Douglas Crockford's JavaScript Lint](http://www.jslint.com/).

Simply visit that web page, paste your JavaScript (Only JavaScript) code into the text area provided, and click the jslint button. This program will parse through your JavaScript code, ensuring that all the variable and function definitions follow the correct syntax. It will also check JavaScript statements, such as **if** and **while**, to ensure they too follow the correct format.

**Add Debugging Code to Your Programs**

You can use the **alert()** or **document.write()** methods in your program to debug your code. For example, you might write something as follows:

```javascript
    var debugging = true;
    var whichImage = "widget";
    if( debugging )
        alert( "Calls swapImage() with argument: " + whichImage );
    var swapStatus = swapImage( whichImage );
    if( debugging )
        alert( "Exits swapImage() with swapStatus=" + swapStatus );
```

By examining the content and order of the **alert()** as they appear, you can examine the health of your program very easily.
Use a JavaScript Debugger

A debugger is an application that places all aspects of script execution under the control of the programmer. Debuggers provide fine-grained control over the state of the script through an interface that allows you to examine and set values as well as control the flow of execution.

Once a script has been loaded into a debugger, it can be run one line at a time or instructed to halt at certain breakpoints. Once execution is halted, the programmer can examine the state of the script and its variables in order to determine if something is amiss. You can also watch variables for changes in their values.

The latest version of the Mozilla JavaScript Debugger (code-named Venkman) for both Mozilla and Netscape browsers can be downloaded at http://www.hacksrus.com/~ginda/venkman.

Useful Tips for Developers

You can keep the following tips in mind to reduce the number of errors in your scripts and simplify the debugging process:

- Use plenty of comments. Comments enable you to explain why you wrote the script the way you did and to explain particularly difficult sections of code.

- Always use indentation to make your code easy to read. Indenting statements also makes it easier for you to match up beginning and ending tags, curly braces, and other HTML and script elements.

- Write modular code. Whenever possible, group your statements into functions. Functions let you group related statements, and test and reuse portions of code with minimal effort.

- Be consistent in the way you name your variables and functions. Try using names that are long enough to be meaningful and that describe the contents of the variable or the purpose of the function.

- Use consistent syntax when naming variables and functions. In other words, keep them all lowercase or all uppercase; if you prefer Camel-Back notation, use it consistently.

- Test long scripts in a modular fashion. In other words, do not try to write the entire script before testing any portion of it. Write a piece and get it to work before adding the next portion of code.

- Use descriptive variable and function names and avoid using single-character names.
• **Watch your quotation marks.** Remember that quotation marks are used in pairs around strings and that both quotation marks must be of the same style (either single or double).

• **Watch your equal signs.** You should not use a single `=` for comparison purpose.

• Declare **variables explicitly** using the `var` keyword.
You can use JavaScript to create client-side image maps. Client-side image maps are enabled by the `usemap` attribute for the `<img />` tag and defined by special `<map>` and `<area>` extension tags.

The image that is going to form the map is inserted into the page using the `<img />` element as normal, except that it carries an extra attribute called `usemap`. The value of the `usemap` attribute is the value of the `name` attribute on the `<map>` element, which you are about to meet, preceded by a pound or hash sign.

The `<map>` element actually creates the map for the image and usually follows directly after the `<img />` element. It acts as a container for the `<area />` elements that actually define the clickable hotspots. The `<map>` element carries only one attribute, the `name` attribute, which is the name that identifies the map. This is how the `<img />` element knows which `<map>` element to use.

The `<area>` element specifies the shape and the coordinates that define the boundaries of each clickable hotspot.

The following code combines imagemaps and JavaScript to produce a message in a text box when the mouse is moved over different parts of an image.

```html
<html>
<head>
<title>Using JavaScript Image Map</title>
<script type="text/javascript">

--

function showTutorial(name){
    document.myform.stage.value = name
}

//-->

</script>
</head>
<body>
<form name="myform">
    <input type="text" name="stage" size="20" />
</form>
</body>
</html>
```
<!-- Create Mappings -->
<img src="/images/usemap.gif" alt="HTML Map"
    border="0" usemap="#tutorials"/>

<map name="tutorials">
    <area shape="poly"
        coords="74,0,113,29,98,72,52,72,38,27"
        href="/perl/index.htm" alt="Perl Tutorial"
        target="_self"
        onMouseOver="showTutorial('perl')"
        onMouseOut="showTutorial('')"/>

    <area shape="rect"
        coords="22,83,126,125"
        href="/html/index.htm" alt="HTML Tutorial"
        target="_self"
        onMouseOver="showTutorial('html')"
        onMouseOut="showTutorial('')"/>

    <area shape="circle"
        coords="73,168,32"
        href="/php/index.htm" alt="PHP Tutorial"
        target="_self"
        onMouseOver="showTutorial('php')"
        onMouseOut="showTutorial('')"/>
</map>
</body>
</html>
Output
You can feel the map concept by placing the mouse cursor on the image object.
It is important to understand the differences between different browsers in order to handle each in the way it is expected. So it is important to know which browser your web page is running in.

To get information about the browser your webpage is currently running in, use the built-in `navigator` object.

**Navigator Properties**

There are several Navigator related properties that you can use in your Web page. The following is a list of the names and descriptions of each.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>appCodeName</code></td>
</tr>
<tr>
<td></td>
<td>This property is a string that contains the code name of the browser, Netscape for Netscape and Microsoft Internet Explorer for Internet Explorer.</td>
</tr>
<tr>
<td>2</td>
<td><code>appVersion</code></td>
</tr>
<tr>
<td></td>
<td>This property is a string that contains the version of the browser as well as other useful information such as its language and compatibility.</td>
</tr>
<tr>
<td>3</td>
<td><code>language</code></td>
</tr>
<tr>
<td></td>
<td>This property contains the two-letter abbreviation for the language that is used by the browser. Netscape only.</td>
</tr>
<tr>
<td>4</td>
<td><code>mimTypes[]</code></td>
</tr>
<tr>
<td></td>
<td>This property is an array that contains all MIME types supported by the client. Netscape only.</td>
</tr>
<tr>
<td>5</td>
<td><code>platform[]</code></td>
</tr>
<tr>
<td></td>
<td>This property is a string that contains the platform for which the browser was compiled.&quot;Win32&quot; for 32-bit Windows operating systems</td>
</tr>
</tbody>
</table>
plugins[]
This property is an array containing all the plug-ins that have been installed on the client. Netscape only.

userAgent[]
This property is a string that contains the code name and version of the browser. This value is sent to the originating server to identify the client.

**Navigator Methods**

There are several Navigator-specific methods. Here is a list of their names and descriptions.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>javaEnabled()</td>
</tr>
<tr>
<td></td>
<td>This method determines if JavaScript is enabled in the client. If JavaScript is enabled, this method returns true; otherwise, it returns false.</td>
</tr>
<tr>
<td>2</td>
<td>plugings.refresh</td>
</tr>
<tr>
<td></td>
<td>This method makes newly installed plug-ins available and populates the plugins array with all new plug-in names. Netscape only.</td>
</tr>
<tr>
<td>3</td>
<td>preference(name,value)</td>
</tr>
<tr>
<td></td>
<td>This method allows a signed script to get and set some Netscape preferences. If the second parameter is omitted, this method will return the value of the specified preference; otherwise, it sets the value. Netscape only.</td>
</tr>
<tr>
<td>4</td>
<td>taintEnabled()</td>
</tr>
<tr>
<td></td>
<td>This method returns true if data tainting is enabled; false otherwise.</td>
</tr>
</tbody>
</table>
Browser Detection

There is a simple JavaScript which can be used to find out the name of a browser and then accordingly an HTML page can be served to the user.

```html
<html>
<head>
<title>Browser Detection Example</title>
</head>
<body>
<script type="text/javascript">
<!--
var userAgent   = navigator.userAgent;
var opera       = (userAgent.indexOf('Opera') != -1);
var ie          = (userAgent.indexOf('MSIE') != -1);
var gecko       = (userAgent.indexOf('Gecko') != -1);
var netscape    = (userAgent.indexOf('Mozilla') != -1);
var version     = navigator.appVersion;

if (opera){
    document.write("Opera based browser");
    // Keep your opera specific URL here.
}else if (gecko){
    document.write("Mozilla based browser");
    // Keep your gecko specific URL here.
}else if (ie){
    document.write("IE based browser");
    // Keep your IE specific URL here.
}else if (netscape){
    document.write("Netscape based browser");
    // Keep your Netscape specific URL here.
}else{
    document.write("Unknown browser");
}
// You can include version to along with any above condition.
</script>
</body>
</html>
```
document.write("<br /> Browser version info : " + version );
//-->
</script>
</body>
</html>

Output
Mozilla based browser
Browser version info : 5.0
(Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2272.101 Safari/537.36