HTML

What is HTML?

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

Output:

Explanation

- The <!DOCTYPE html> declaration defines this document to be HTML5
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the document
- The <title> element specifies a title for the document
- The <body> element contains the visible page content
- The <h1> element defines a large heading
- The element defines a paragraph

HTML Tags

HTML tags are element names surrounded by angle brackets:

<tagname>content goes here...</tagname>

- HTML tags normally come **in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

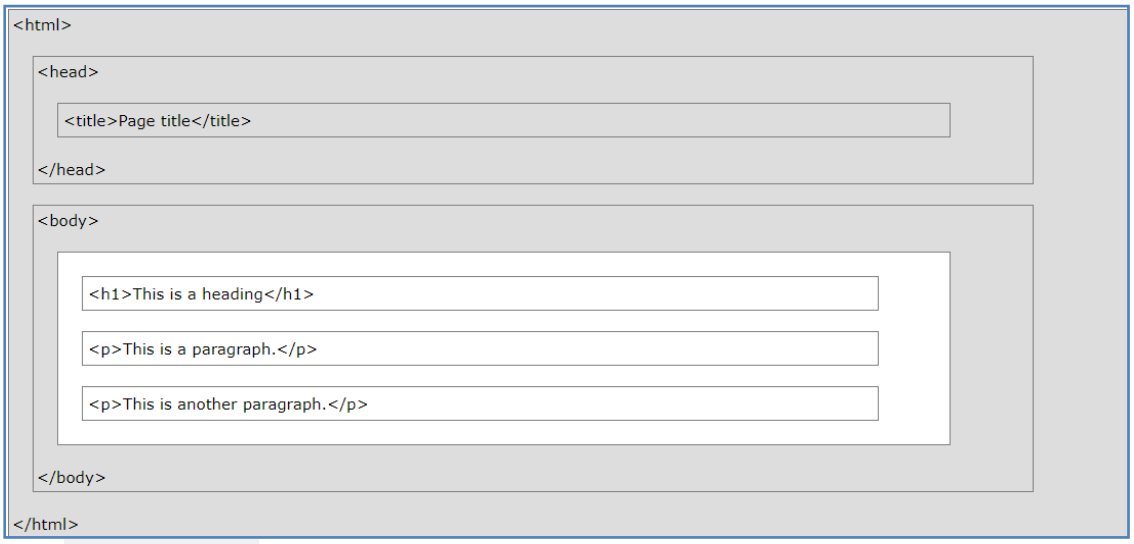
Web Browsers

The purpose of a web browser (Chrome, IE, Firefox, Safari) is to read HTML documents and display them.

The browser does not display the HTML tags, but uses them to determine how to display the document

HTML Page Structure

Below is a visualization of an HTML page structure:



The <!DOCTYPE> declaration represents the document type, and helps browsers to display web pages correctly.
It must only appear once, at the top of the page (before any HTML tags).
The <!DOCTYPE> declaration is not case sensitive.

HTML Versions

Since the early days of the web, there have been many versions of HTML:

Version	Year
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

HTML History

Since the early days of the World Wide Web, there have been many versions of HTML:

Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5

2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition
2017	W3C Recommendation: HTML5.2

Write HTML Using Notepad or TextEdit

Web pages can be created and modified by using professional HTML editors.
However, for learning HTML we recommend a simple text editor like Notepad (PC) or TextEdit (Mac).

Steps to create a web page

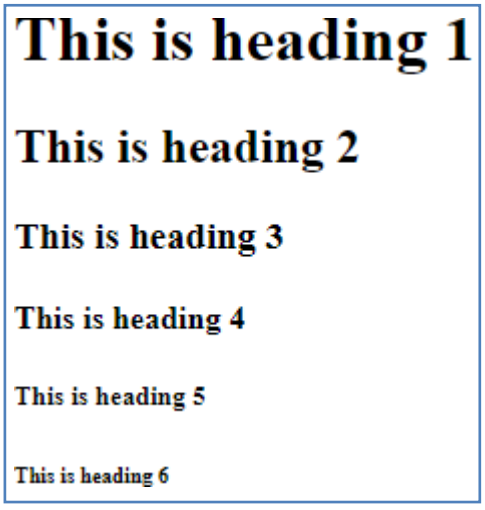
1. Open Notepad
2. Write or copy some HTML into Notepad.
3. Save the file on your computer. Select **File > Save as** in the Notepad menu.
4. Name the file **"index.htm"** and set the encoding to **UTF-8** (which is the preferred encoding for HTML files).
5. Open the saved HTML file in your favorite browser (double click on the file, or right-click - and choose "Open with")

HTML Headings

HTML headings are defined with the <h1> to <h6> tags.
<h1> defines the most important heading. <h6> defines the least important heading:

```
<!DOCTYPE html>
<html>
<body>
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
</body>
</html>
```

Output



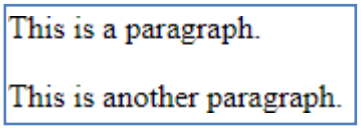
HTML Paragraphs

HTML paragraphs are defined with the <p> tag:

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

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
```

Output



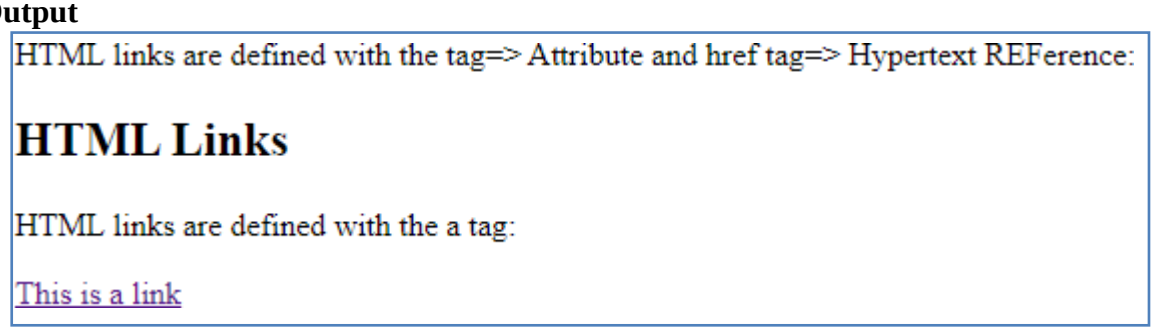
This is a paragraph.
This is another paragraph.

HTML Links

HTML links are defined with the <a> tag=> Attribute and href tag=> Hypertext REFerence:

```
<!DOCTYPE html>
<html>
<body>
<h2>HTML Links</h2>
<p>HTML links are defined with the a tag:</p>
<a href="https://www.w3schools.com">This is a link</a>
</body>
</html>
```

Output



Explanation

The link's destination is specified in the href attribute.
Attributes are used to provide additional information about HTML elements.
You will learn more about attributes in a later chapter.

HTML Images

HTML images are defined with the <img> tag.
The source file (src), alternative text (alt), width, and height are provided as attributes:

```
<!DOCTYPE html>
<html>
<body>
<h2>HTML Images</h2>
<p>HTML images are defined with the imgtag:</p>
<img src="w3schoolslogo.png" alt="W3Schools.com" width="104" height="142">
</body>
</html>
```

Output

HTML Attributes

Attributes provide additional information about HTML elements.

- All HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes usually come in name/value pairs like: **name="value"**

The src Attribute

HTML images are defined with the <img> tag.

The filename of the image source is specified in the src attribute:

The width and height Attributes

Images in HTML have a set of **size** attributes, which specifies the width and height of the image

```
<!DOCTYPE html>
<html>
<body>
<h2>HTML Images</h2>
<p>HTML images are defined with the imgtag:</p>
<img src="" alt="W3Schools.com" width="104" height="142">
</body>
</html>
```

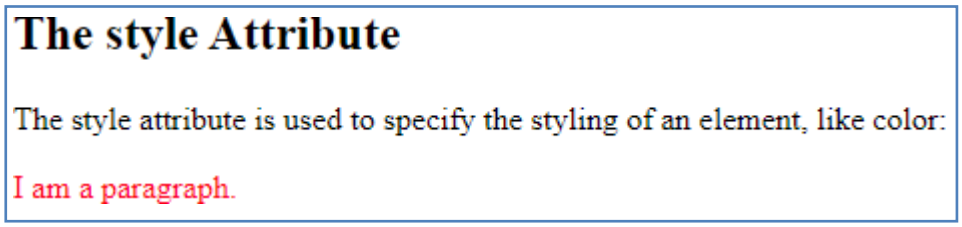
Output

The style Attribute

The style attribute is used to specify the styling of an element, like color, font, size etc.

```
<!DOCTYPE html>
<html>
<body>
<h2>The style Attribute</h2>
<p>The style attribute is used to specify the styling of an element, like color:</p>
<p style="color:red">I am a paragraph.</p>
</body>
</html>
```

Output

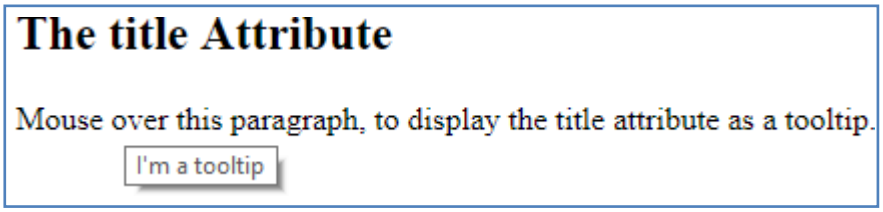


The title Attribute

Here, a title attribute is added to the <p> element. The value of the title attribute will be displayed as a tooltip when you mouse over the paragraph:

```
<!DOCTYPE html>
<html>
<body>
<h2 title="I'm a header in size h2">The title Attribute</h2>
<p title="I'm a tooltip">
Mouse over this paragraph, to display the title attribute as a tooltip.
</p>
</body>
</html>
```

Output



HTML Horizontal Rules

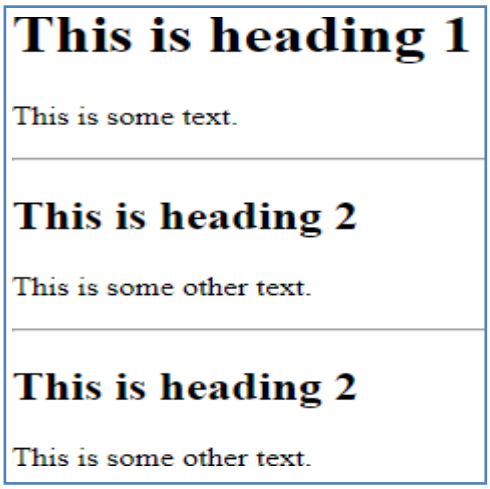
The <hr> tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.

The <hr> element is used to separate content (or define a change) in an HTML page:

```
<!DOCTYPE html>
<html>
<body>
<h1>This is heading 1</h1>
<p>This is some text.</p>
<hr>
<h2>This is heading 2</h2>
<p>This is some other text.</p>
<hr>
<h2>This is heading 2</h2>
<p>This is some other text.</p>
</body>
```

</html>

Output



The HTML <head> Element

The HTML <head> element has nothing to do with HTML headings.

The <head> element is a container for metadata. HTML metadata is data about the HTML document. Metadata is not displayed.

The <head> element is placed between the <html> tag and the <body> tag:

```
<!DOCTYPE html>
<html>
<head>
<title>My First HTML</title>
<meta charset="UTF-8">
</head>
<body>
<p>The HTML head element contains meta data.</p>
<p>Meta data is data about the HTML document.</p>
</body>
</html>
```

Output

How to View HTML Source?

Have you ever seen a Web page and wondered "Hey! How did they do that?"

View HTML Source Code:

Right-click in an HTML page and select "View Page Source" (in Chrome) or "View Source" (in IE), or similar in other browsers. This will open a window containing the HTML source code of the page.

Inspect an HTML Element:

Right-click on an element (or a blank area), and choose "Inspect" or "Inspect Element" to see what elements are made up of (you will see both the HTML and the CSS). You can also edit the HTML or CSS on-the-fly in the Elements or Styles panel that opens.

HTML Background Color

The background-color property defines the background color for an HTML element.

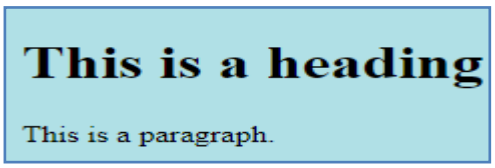
This example sets the background color for a page to powderblue:

(<body bgcolor="red;">)

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

```
<body style="background-color:powderblue;">
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Output

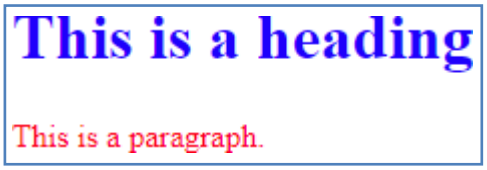


HTML Text Color

The color property defines the text color for an HTML element:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="color:blue;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>
</body>
</html>
```

Output

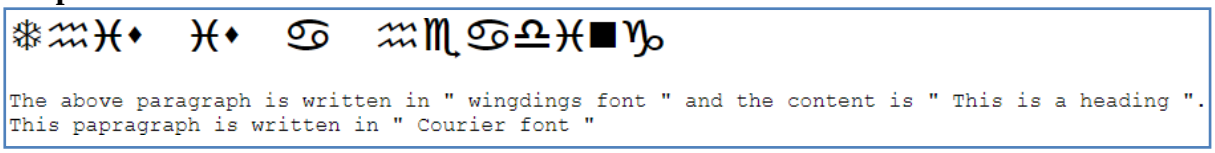


HTML Fonts

The font-family property defines the font to be used for an HTML element:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="font-family:wingdings;">This is a heading</h1>
<p style="font-family:courier;">The above paragraph is written in &amp;quot; wingdings font
&amp;quot; and the content is &amp;quot; This is a heading &amp;quot;.<br>
This papragraph is written in &amp;quot; Courier font &amp;quot;</p>
</body>
</html>
```

Output



HTML Text Alignment

The text-align property defines the horizontal text alignment for an HTML element:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="text-align:Left;">left Heading</h1>
<h1 style="text-align:center;">Center Heading</h1>
<h1 style="text-align:Right;">Right Heading</h1>
</body>
```

```
</html>
```

Output

HTML Formatting Elements

In the previous chapter, you learned about the HTML **style attribute**.
HTML also defines special **elements** for defining text with a special **meaning**.
HTML uses elements like <b> and <i> for formatting output, like **bold** or *italic* text.
Formatting elements were designed to display special types of text:

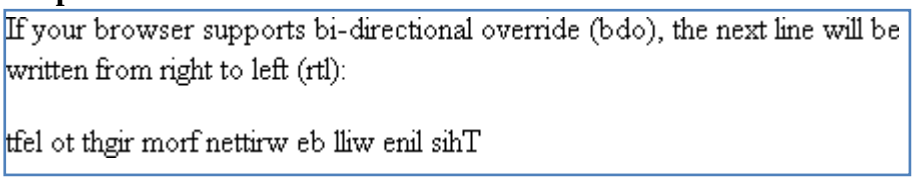
- <b> - Bold text
- - Important text
- <i> - Italic text
- - Emphasized text
- <mark> - Marked text
- <small> - Small text
- <del> - Deleted text
- <ins> - Inserted text
- <sub> - Subscript text
- <sup> - Superscript text

HTML <bdo> for Bi-Directional Override

The HTML <bdo> element defines bi-directional override.
The <bdo> element is used to override the current text direction:

```
<!DOCTYPE html>
<html>
<body>
<p>If your browser supports bi-directional override (bdo), the next line will be written from right to left (rtl):</p>
<bdo dir="rtl">This line will be written from right to left</bdo>
</body>
</html>
```

Output



HTML Comment Tags

You can add comments to your HTML source by using the following syntax:

```
<!-- Write your comments here -->
```

Notice that there is an exclamation point (!) in the opening tag, but not in the closing tag.
Note: Comments are not displayed by the browser, but they can help document your HTML source code.
With comments you can place notifications and reminders in your HTML:

```
<!DOCTYPE html>
<html>
<body>
<!-- This is a comment -->
<p>This is a testing for comment.</p>
<!-- Comments are not displayed in the browser -->
```

</body>
</html>

Output

This is a testing for comment.

HTML Colors

HTML colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

Named Colors Sorted by HEX Value

Colorname	HEX	RGB
Black	0	0,0,0
Blue	0000FF	0,0,255
Brown	A52A2A	165,42,42
DeepPink	FF1493	2,55,20,147
Gold	FFD700	255,215,0
Green	8000	0,128,0
Magenta	FF00FF	255,0,255
Maroon	800000	128,0,0
Orange	FFA500	255,165,0
Red	FF0000	255,0,0
White	FFFFFF	25,52,55,255

Background Color or Background

You can set the background color for HTML elements:

(<body bgcolor="red;">)

```
<!DOCTYPE html>
<html>
<body>
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">
• HTML stands for Hyper Text Markup Language<br>
• HTML describes the structure of Web pages using markup<br>
• HTML elements are the building blocks of HTML pages<br>
• HTML elements are represented by tags<br>
• HTML tags label pieces of content such as "heading", "paragraph", "table", and so on<br>
• Browsers do not display the HTML tags, but use them to render the content of the page
</p>
</body>
</html>
```

Output

Hello World

• HTML stands for Hyper Text Markup Language
• HTML describes the structure of Web pages using markup
• HTML elements are the building blocks of HTML pages
• HTML elements are represented by tags
• HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
• Browsers do not display the HTML tags, but use them to render the content of the page

Text Color

You can set the color of text:

```
<!DOCTYPE html>
```

```
<html>
<body>
<h3 style="color:Tomato;">Hello World</h3>
<p style="color:DodgerBlue;">Hello World</p>
<p style="color:MediumSeaGreen;">Hello World</p>
</body>
</html>
```

Output

Border Color

You can set the color of borders:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="border: 2px solid Tomato;">Hello World</h1>
<h1 style="border: 2px solid DodgerBlue;">Hello World</h1>
<h1 style="border: 2px solid Violet;">Hello World</h1>
</body>
</html>
```

Output

Color Values

In HTML, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

Same as color name "Tomato":

RGB Value

In HTML, a color can be specified as an RGB value, using this formula:

rgb(*red, green, blue*)

Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255.

For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255) and the others are set to 0.

To display the color black, all color parameters must be set to 0, like this: rgb(0, 0, 0).

To display the color white, all color parameters must be set to 255, like this: rgb(255, 255, 255).

HEX Value

In HTML, a color can be specified using a hexadecimal value in the form:

#rrggbb

Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).

For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).

HSL Value

In HTML, a color can be specified using hue, saturation, and lightness (HSL) in the form:

hsl(*hue, saturation, lightness*)
Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
Saturation is a percentage value, 0% means a shade of gray, and 100% is the full color.
Lightness is also a percentage, 0% is black, 50% is neither light or dark, 100% is white
Saturation
Saturation can be described as the intensity of a color.
100% is pure color, no shades of gray
50% is 50% gray, but you can still see the color.
0% is completely gray, you can no longer see the color.
Lightness
The lightness of a color can be described as how much light you want to give the color, where 0% means no light (black), 50% means 50% light (neither dark nor light) 100% means full lightness (white).

RGBA Value

RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color.
An RGBA color value is specified with:
rgba(*red, green, blue, alpha*)
The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):
HSLA Value
HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color.
An HSLA color value is specified with:
hsla(*hue, saturation, lightness, alpha*)
The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

HTML Lists

Unordered HTML List

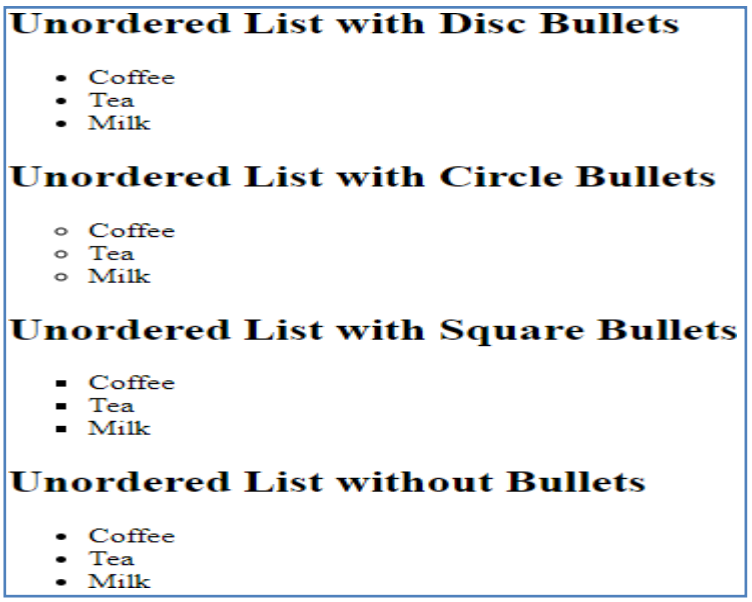
An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.
The list items will be marked with bullets (small black circles) by default:
Unordered HTML List - Choose List Item Marker
The CSS list-style-type property is used to define the style of the list item marker:

Value	Description
disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

```
<!DOCTYPE html>
<html>
<body>
<h2>Unordered List with Disc Bullets</h2>
<ul style="list-style-type:disc">
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ul>

<h2>Unordered List with Circle Bullets</h2>
<ul style="list-style-type:circle">
<li>Coffee</li>
```

```
<li>Tea</li>
<li>Milk</li>
</ul>

<h2>Unordered List with Square Bullets</h2>
<ul style="list-style-type:square">
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ul>

<h2>Unordered List without Bullets</h2>
<ul>
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ul>
</body>
</html>
```

Output

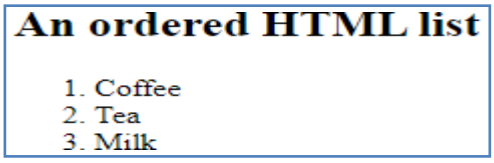


Ordered HTML List

An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.
The list items will be marked with numbers by default:

```
<!DOCTYPE html>
<html>
<body>
<h2>An ordered HTML list</h2>
<ol>
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ol>
</body>
</html>
```

Output

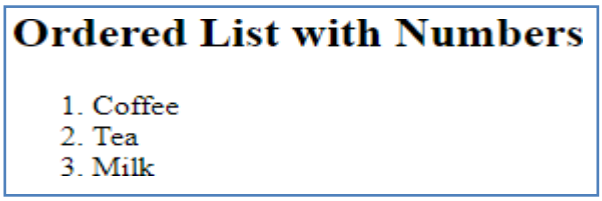


Ordered HTML List - The Type Attribute

The type attribute of the <ol> tag, defines the type of the list item marker:

```
<!DOCTYPE html>
<html>
<body>
<h2>Ordered List with Numbers</h2>
<ol type="1">
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ol>
</body>
</html>
```

Output



Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers
type="3"	The list items will be numbered with number 3
type="A" start="4"	The list items will be numbered with uppercase letters from D
type="a" start="4"	The list items will be numbered with lowercase letters from d
type="I" start="4"	The list items will be numbered with uppercase roman numbers from IV
type="i" start="4"	The list items will be numbered with lowercase roman numbers from iv

Ordered List with Numbers

1. Coffee
2. Tea
3. Milk

Ordered List with Letters

A. Coffee
B. Tea
C. Milk

Ordered List with Lowercase Letters

a. Coffee
b. Tea
c. Milk

Ordered List with Roman Numbers

I. Coffee
II. Tea
III. Milk

Ordered List with Numbers starting from 4

4. Coffee
5. Tea
6. Milk

Ordered List with Uppercase letters starting from 4

D. Coffee
E. Tea
F. Milk

Ordered List with Lowercase letters starting from 4

a. Coffee
b. Tea
c. Milk

Ordered List with Roman letters starting from 4

IV. Coffee
V. Tea
VI. Milk

Ordered List with Roman letters starting from 4

iv. Coffee
v. Tea
vi. Milk

HTML Description Lists

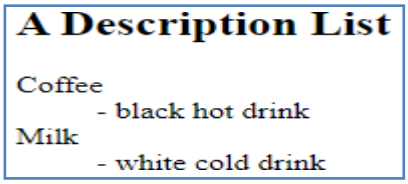
HTML also supports description lists.

A description list is a list of terms, with a description of each term.

The <dl> tag defines the description list, the <dt> tag defines the term (name), and the <dd> tag describes each term:

```
<!DOCTYPE html>
<html>
<body>
<h2>A Description List</h2>
<dl>
<dt>Coffee</dt>
<dd>- black hot drink</dd>
<dt>Milk</dt>
<dd>- white cold drink</dd>
</dl>
</body>
</html>
```

Output

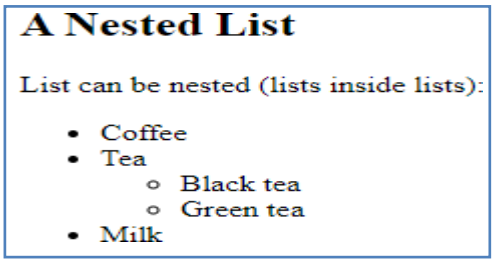


Nested HTML Lists

List can be nested (lists inside lists):

```
<!DOCTYPE html>
<html>
<body>
<h2>A Nested List</h2>
<p>List can be nested (lists inside lists):</p>
<ul>
<li>Coffee</li>
<li>Tea
        <ul>
        <li>Black tea</li>
        <li>Green tea</li>
        </ul>
</li>
<li>Milk</li>
</ul>
</body>
</html>
```

Output



Control List Counting

By default, an ordered list will start counting from 1. If you want to start counting from a specified number, you can use the start attribute:

```
<!DOCTYPE html>
<html>
<body>
<h2>The start attribute</h2>
<p>By default, an ordered list will start counting from 1. Use the start attribute to start counting from a specified number:</p>
<ol start="50">
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ol>
<ol type="I" start="50">
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ol>
</body>
</html>
```

Output

The start attribute

By default, an ordered list will start counting from 1. Use the start attribute to start counting from a specified number:

50. Coffee
51. Tea
52. Milk

L. Coffee
LI. Tea
LII. Milk

Defining an HTML Table

An HTML table is defined with the <table> tag.

Each table row is defined with the <tr> tag. A table header is defined with the <th> tag. By default, table headings are bold and centered. A table data/cell is defined with the <td> tag.

```
<!DOCTYPE html>
<html>
<table border = "1">
<tr>
<th>Name</th>
<th>Oops</th>
<th>dbms</th>
</tr>
<tr>
<td>Jill</td>
<td>80</td>
<td>50</td>
</tr>
<tr>
<td>Eve</td>
<td>50</td>
<td>94</td>
</tr>
<tr>
<td>John</td>
<td>90</td>
<td>80</td>
</tr>
</table>
</html>
```

Output

Name	Oops	dbms
Jill	80	50
Eve	50	94
John	90	80

Name	Oops	dbms
Jill	80	50
Eve	50	94
John	90	80

HTML Table - Cells that Span Many Columns& Rows, Caption

To make a cell span more than one column, use the colspan attribute:

To make a cell span more than one row, use the rowspan attribute:
To add a caption to a table, use the <caption> tag:

```
<!DOCTYPE html>
<html>

<table border = "1">
<caption>Marks Statement</caption>
<tr>
<!--Rowspan merges n number of rows
                colspan merges n number of columns
        -->
<th rowspan=2>Name</th><th colspan=3>Marks</th>
</tr>
        <tr>
        <th>Oops</th><th>Dbms</th><th>Java</th>
        </tr>
<tr>

<td>Jill</td><td>80</td><td>50</td><td>50</td>
</tr>
<tr>
<td>Eve</td><td>50</td><td>94</td><td>50</td>
</tr>
<tr>
<td>John</td><td>90</td><td>80</td><td>50</td>
</tr>
</table>
</html>
```

Output

Marks Statement

Name	Marks		
	Oops	Dbms	Java
Jill	80	50	50
Eve	50	94	50
John	90	80	50

Use the HTML <table> element to define a table
Use the HTML <tr> element to define a table row
Use the HTML <td> element to define a table data
Use the HTML <th> element to define a table heading
Use the HTML <caption> element to define a table caption
Use the CSS border property to define a border
Use the CSS border-collapse property to collapse cell borders
Use the CSS padding property to add padding to cells
Use the CSS text-align property to align cell text
Use the CSS border-spacing property to set the spacing between cells

Use the colspan attribute to make a cell span many columns

Use the rowspan attribute to make a cell span many rows
Use the id attribute to uniquely define one table

Attributes of table tag

Define Table: <TABLE></TABLE>
Columns to Span: <TH COLSPAN=?>
Rows to Span: <TH ROWSPAN=?>
Desired Width: <TH WIDTH=?> – *(in pixels)*
Width Percent: <TH WIDTH="%"> – *(percentage of table)*
Cell Color: <TH BGCOLOR="#$$$$$$">
Table Caption: <CAPTION></CAPTION>
Alignment: <CAPTION ALIGN=TOP|BOTTOM> – *(above/below table)*
Table Border: <TABLE BORDER=?></TABLE>
Cell Spacing: <TABLE CELLSPACING=?>
Cell Padding: <TABLE CELLPADDING=?>
Desired Width: <TABLE WIDTH=?> – *(in pixels)*
Width Percent: <TABLE WIDTH="%"> – *(percentage of page)*
Table Row: <TR></TR>
Alignment: <TR ALIGN=LEFT|RIGHT| CENTER|MIDDLE|BOTTOM
VALIGN=TOP|BOTTOM|MIDDLE>
Table Cell: <TD></TD> – *(must appear within table rows)*
Alignment: <TD ALIGN=LEFT|RIGHT| CENTER|MIDDLE|BOTTOM
VALIGN=TOP|BOTTOM|MIDDLE>
No linebreaks: <TD NOWRAP>
Columns to Span: <TD COLSPAN=?>
Rows to Span: <TD ROWSPAN=?>
Desired Width: <TD WIDTH=?>
Width Percent: <TD WIDTH="%"> – *(percentage of table)*
Cell Color: <TD BGCOLOR="#$$$$$$">
Table Header: <TH></TH> – *(same as data, except bold centered)*
Alignment: <TH ALIGN=LEFT|RIGHT| CENTER|MIDDLE|BOTTOM
VALIGN=TOP|BOTTOM|MIDDLE>
No Linebreaks: <TH NOWRAP>

Attributes of <tr> tag

Attribute name	**Notes**

<tr align="">	Sets the horizontal alignment for the contents of each <td> element in a table row.
<trvalign="">	Sets the vertical alignment of all content in a table row.
<trbgcolor="">	Sets the background color for a single table row in an HTML table.
<tr background="">	Identifies the URL of a file to be used as a background image for a table row.
<trbordercolor="">	Sets the border color for all inside borders of a table row.

Attributes of <td > tag

Attribute name	Notes
<td nowrap>	NOWRAP indicates that text should not wrap in the cell.
<td bgcolor="">	Sets the background color of a single cell in a table.
<td bordercolor="">	Sets the color of the entire border around a cell.
<td background="">	Specifies the URL of an image file to be used as the <td> element background image.
<td colspan="">	Indicates how many columns a cell should take up.
<td align="">	Was used to specify the alignment of the contents of a single table data cell. This attribute has been deprecated. Use CSS to control alignment of the contents of a table data cell.
<td width="">	Was used to set the width of a table data cell to a value that would override the default width. This attribute has been deprecated. Use CSS to control layout of data cells in HTML tables.

HTML - Frames

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Disadvantages of Frames

There are few drawbacks with using frames, so it's never recommended to use frames in your webpages −

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.
- The browser's *back* button might not work as the user hopes.
- There are still few browsers that do not support frame technology.

Creating Frames

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines, how to divide the window into frames. The **rows** attribute of <frameset> tag defines horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Note − The <frame> tag deprecated in HTML5. Do not use this element.

Red.html

```
<!DOCTYPE html>
<html>
<body style="background-color:red;">
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
```

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

Green.html

```
<!DOCTYPE html>
<html>
<body style="background-color:green;">
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

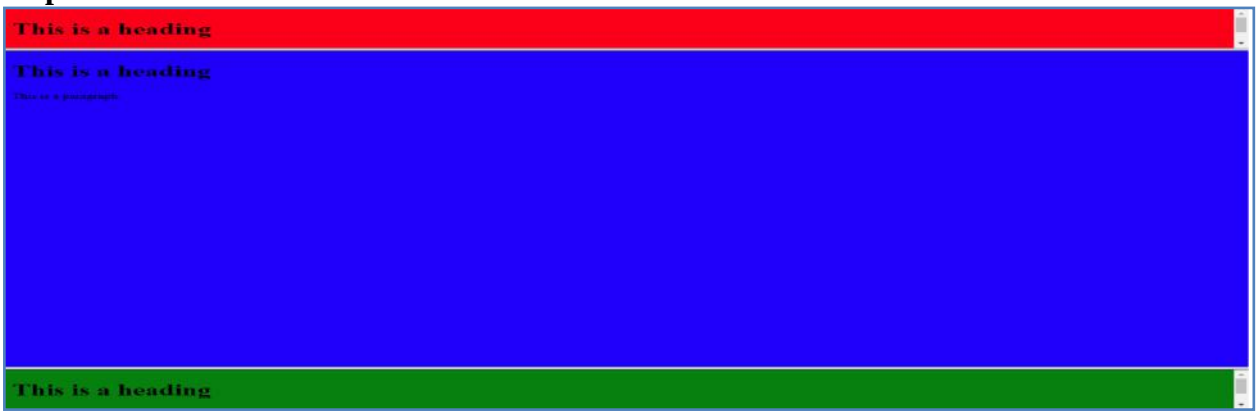
Blue.html

```
<!DOCTYPE html>
<html>
<body style="background-color:blue;">
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Horzframes.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Frames</title>
  </head>
  <frameset rows = "10%,80%,10%">
    <frame name = "top" src = "red.html" />
    <frame name = "main" src = "blue.html" />
    <frame name = "bottom" src = "green.html" />
      <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>
    </frameset>
  </html>
```

output



Vertframes.html

```
<!DOCTYPE html>
<html>
    <head>
    <title>HTML Frames</title>
  </head>
    <frameset cols = "25%,50%,25%">
    <frame name = "left" src = "red.html" />
```

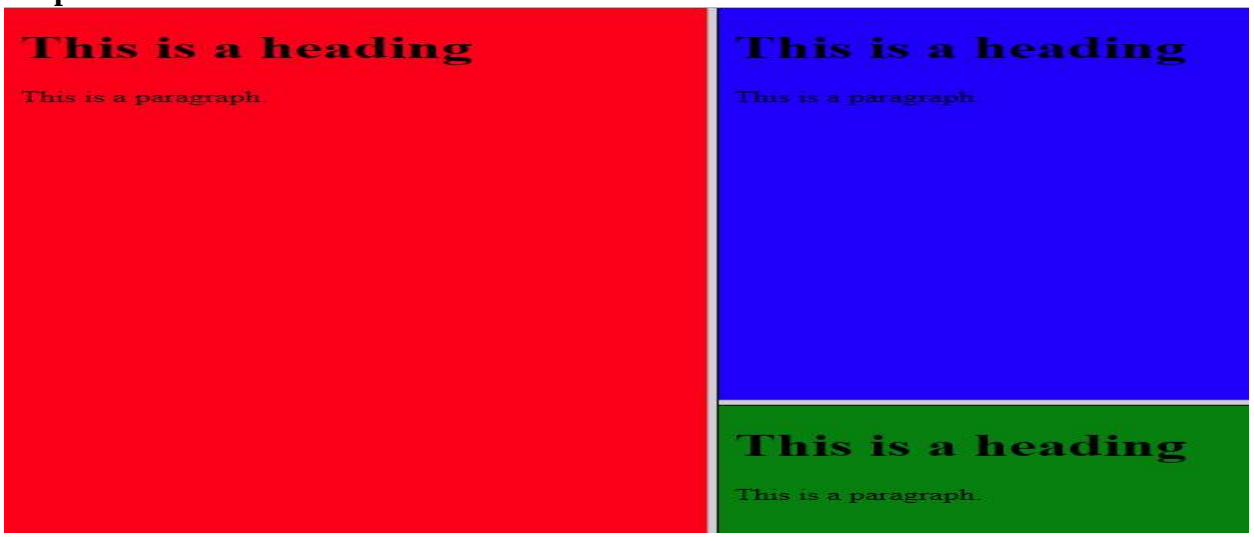
```
    <frame name = "center" src = "blue.html" />
    <frame name = "right" src = "green.html" />
        <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>
  </frameset>
  </html>
```

output

Mixedframes.html

```
<html>
<frameset cols="25%,*" scrolling="no" noresize>
<frame name = "top" src = "red.html" />
<frameset rows="50%,*" scrolling="no" noresize>
    <frame name = "main" src = "blue.html" />
    <frame name = "bottom" src = "green.html" />
</frameset>
</html>
```

Output



The <frameset> Tag Attributes

Following are important attributes of the <frameset> tag −

Sr.No	Attribute & Description
1	**cols** Specifies how many columns are contained in the frameset and the size of each column. You can specify the width of each column in one of the four ways − Absolute values in pixels. For example, to create three vertical frames, use *cols = "100, 500, 100"*. A percentage of the browser window. For example, to create three vertical frames,

	use *cols = "10%, 80%, 10%"*. Using a wildcard symbol. For example, to create three vertical frames, use *cols = "10%, *, 10%"*. In this case wildcard takes remainder of the window. As relative widths of the browser window. For example, to create three vertical frames, use *cols = "3*, 2*, 1*"*. This is an alternative to percentages. You can use relative widths of the browser window. Here the window is divided into sixths: the first column takes up half of the window, the second takes one third, and the third takes one sixth.
2	**rows** This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example, to create two horizontal frames, use *rows = "10%, 90%"*. You can specify the height of each row in the same way as explained above for columns.
3	**border** This attribute specifies the width of the border of each frame in pixels. For example, border = "5". A value of zero means no border.
4	**frameborder** This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example frameborder = "0" specifies no border.
5	**framespacing** This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example framespacing = "10" means there should be 10 pixels spacing between each frames.

The <frame> Tag Attributes

Following are the important attributes of <frame> tag −

Sr.No	Attribute & Description
1	**src** This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src = "/html/top_frame.htm" will load an HTML file available in html directory.
2	**name**

	This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.
3	**frameborder** This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
4	**marginwidth** This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth = "10".
5	**marginheight** This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight = "10".
6	**noresize** By default, you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize = "noresize".
7	**scrolling** This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling = "no" means it should not have scroll bars.
8	**longdesc** This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example longdesc = "framedescription.htm"

Javascript

https://www.tutorialspoint.com/javascript/javascript_overview.htm

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 multi-threading or multiprocessor capabilities.

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

Applications of Javascript Programming

As mentioned before, **Javascript** is one of the most widely used **programming languages** (Front-end as well as Back-end). It has it's presence in almost every area of software development.

- **Client side validation** - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validting those inputs at front-end itself.
- **Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.
- **User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
- **Back-end Data Loading** - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
- **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
- **Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

JavaScript - Syntax

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.

```
<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".

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

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

OUTPUT

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.

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

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

```
<scriptlanguage="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.

```
<scriptlanguage="javascript"type="text/javascript">
<!--
// This is a comment. It is similar to comments in C++
/*
    * This is a multi-line comment in JavaScript
    * It is very similar to comments in C Programming
    */
//-->
</script>
```

Enabling JavaScript in Browsers

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 simple steps to turn on or turn off JavaScript in your 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 **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 **toogle**. 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 - Placement in HTML File

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.

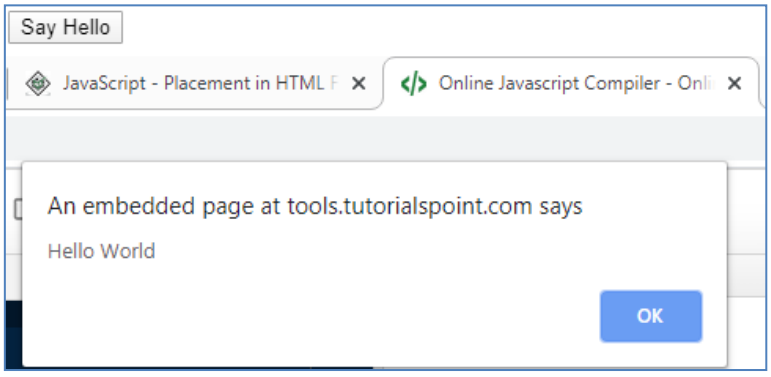
JavaScript in <head>...</head> section

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>
<head>
<scripttype="text/javascript">
<!--
function sayHello(){
        alert("Hello World")
```

```
}
//-->
</script>
</head>
<body>
<inputtype="button"onclick="sayHello()"value="Say Hello"/>
</body>
</html>
```

OUTPUT



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

OUTPUT

Hello World

This is web page body

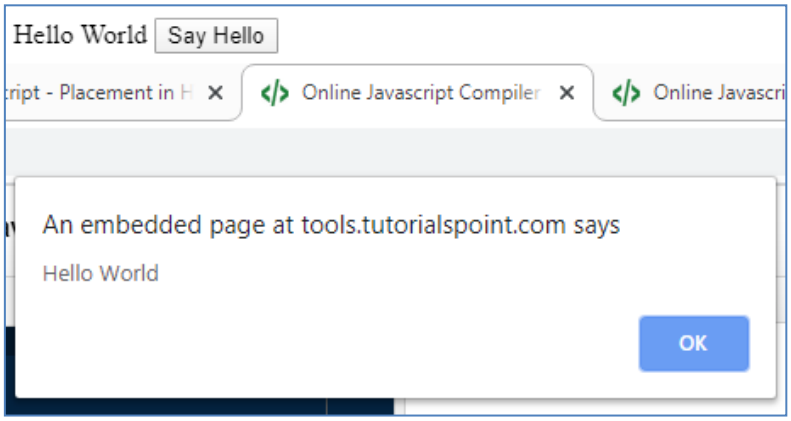
JavaScript in <body> and <head> Sections

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

```
<html>
<head>
<scripttype="text/javascript">
<!--
function sayHello(){
           alert("Hello World")
}
```

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

<body>
<scripttype="text/javascript">
<!--
        document.write("Hello World")
//-->
</script>
<inputtype="button"onclick="sayHello()"value="Say Hello"/>
</body>
</html>
```

OUTPUT



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.

```
<html>
<head>
<scripttype="text/javascript"src="filename.js"></script>
</head>
<body>
     .....
</body>
</html>
```

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.

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

JavaScript - Variables

JavaScript Datatypes

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

JavaScript allows you to work with three primitive data types −

- **Numbers,** eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined,** each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

Note − JavaScript 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.

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

You can also declare multiple variables with the same **var** keyword as follows −

```
<scripttype="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.

```
<scripttype="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.

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

OUTPUT

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.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws

catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

Operators

JavaScript supports the following types of operators.

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

Arithmetic Operators

```
<html>
<body>

<scripttype="text/javascript">
<!--
var a =33;
var b =10;
var c ="Test";
var linebreak ="<br />";

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

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

        document.write("a / b = ");
```

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

        document.write("a % b = ");
        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>

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

OUTPUT

a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...

Comparison Operators

```
<html>
<body>
<scripttype="text/javascript">
<!--
var a =10;
var b =20;
var linebreak ="<br />";
```

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

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

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

        document.write("(a != b) =>");
        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>
    Set the variables to different values and different operators and then try...
</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

```
<html>
<body>
<scripttype="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

```
<html>
<body>
<scripttype="text/javascript">
<!--
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);

        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

```
<html>
<body>
<scripttype="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);
        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 Operator

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.

? : (Conditional)

If Condition is true? Then value X : Otherwise value Y

```
<html>
<body>
<scripttype="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) =>");
        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) => 200
((a < b) ? 100 : 200) => 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.

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

```
<html>
<body>
<scripttype="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...

JavaScript - if...else Statement

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 −

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

```
<html>
<body>
<scripttype="text/javascript">
<!--
var age =20;

if( age >18){
            document.write("<b>Qualifies for driving</b>");
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
```

```
</html>
```

OUTPUT

Qualifies for driving

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

if...else statement

The '**if...else**' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

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

```
<html>
<body>
<scripttype="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>
<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 −

```
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
}
```

```
<html>
```

```
<body>
<scripttype="text/javascript">
<!--
var book ="maths";
if( book =="history"){
        document.write("<b>History Book</b>");
}elseif( book =="maths"){
        document.write("<b>Maths Book</b>");
}elseif( book =="economics"){
        document.write("<b>Economics Book</b>");
}else{
        document.write("<b>Unknown Book</b>");
}
//-->
</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...

JavaScript - Switch Case

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.

```
switch (expression) {
  case condition 1: statement(s)
  break;

  case condition 2: statement(s)
  break;
  ...

  case condition n: statement(s)
  break;

  default: statement(s)
}
```

```
<html>
<body>
<scripttype="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>
<scripttype="text/javascript">
<!--
var grade ='A';
        document.write("Entering switch block<br />");
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");
//-->
</script>
<p>Set the variable to different value and then try...</p>
```

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

OUTPUT

Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
Set the variable to different value and then try...

JavaScript - While Loops

Syntax

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

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

```
<html>
<body>

<scripttype="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
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 do...while Loop

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.

```
<html>
<body>
<scripttype="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...

JavaScript - For Loop

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
}
```

```
<html>
<body>
<scripttype="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...

JavaScript *for...in* loop

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 −

```
for (variablename 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.

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

OUTPUT

Navigator Object Properties
serviceWorker
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack
onLine
languages
language
userAgent
product
platform
appVersion
appName
appCodeName
hardwareConcurrency
maxTouchPoints
vendorSub
vendor
productSub
cookieEnabled
mimeTypes
plugins
javaEnabled
getStorageUpdates
getGamepads
webkitGetUserMedia
vibrate
getBattery
sendBeacon
registerProtocolHandler
unregisterProtocolHandler
Exiting from the loop!
Set the variable to different object and then try...

JavaScript - Loop Control

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.

Break Statement

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>
<scripttype="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...

Continue Statement

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>
<body>
<scripttype="text/javascript">
<!--
var x =1;
           document.write("Entering the loop<br />");

while(x <10){
             x = x +1;
```

```
if(x ==5){
continue;// skip 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

Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
Set the variable to different value and then try...

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.

```
<html>
<body>
<scripttype="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++){
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

```
<html>
<body>

<scripttype="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!
```

JavaScript - Functions

Function Definition

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

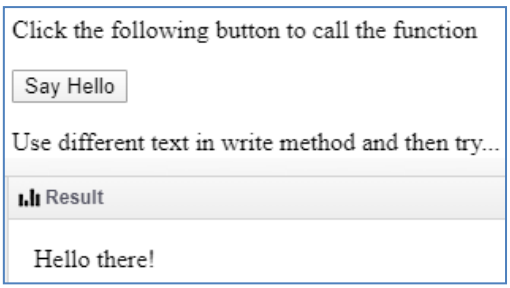
```
<scripttype="text/javascript">
<!--
function sayHello(){
       alert("Hello there");
}
//-->
</script>
```

Calling a Function

```
<html>
<head>
<scripttype="text/javascript">
function sayHello(){
         document.write ("Hello there!");
}
</script>

</head>

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

```
<form>
<inputtype="button"onclick="sayHello()"value="Say Hello">
</form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```

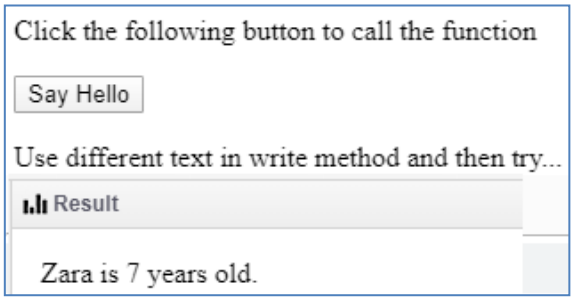
OUTPUT



Function Parameters

```
<html>
<head>
<scripttype="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>
<inputtype="button"onclick="sayHello('Zara',7)"value="Say Hello">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

OUTPUT

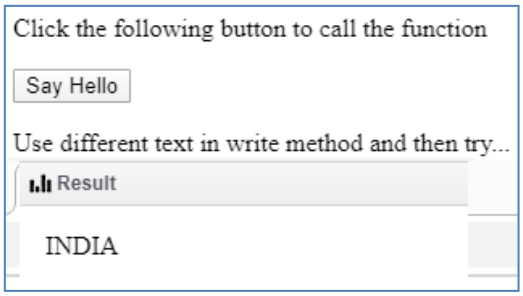


The return Statement

```
<html>
<head>
<scripttype="text/javascript">
```

```
function concatenate(first, last){
var full;
        full = first + last;
return full;
}
function secondFunction(){
var result;
        result = concatenate('IND','IA');
        document.write (result );
}
</script>
</head>

<body>
<p>Click the following button to call the function</p>
<form>
<inputtype="button"onclick="secondFunction()"value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

OUTPUT



JavaScript - Events

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.

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.

```
<body>
<p>Click the following button and see result</p>
<form>
<inputtype="button"onclick="sayHello()"value="Say Hello"/>
</form>
</body>
```

onsubmit Event Type

```
<body>
<formmethod="POST"action="t.cgi"onsubmit="return validate()">
          .......
<inputtype="submit"value="Submit"/>
</form>
</body>
```

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.

```
<html>
<head>
<scripttype="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>
<divonmouseover="over()"onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```

HTML 5 Standard Events

Attribute	Description
Offline	Triggers when the document goes offline
Onabort	Triggers on an abort event
onafterprint	Triggers after the document is printed
onbeforeonload	Triggers before the document loads
onbeforeprint	Triggers before the document is printed

onblur	Triggers when the window loses focus
oncanplay	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	Triggers when media can be played to the end, without stopping for buffering
onchange	Triggers when an element changes
onclick	Triggers on a mouse click
oncontextmenu	Triggers when a context menu is triggered
ondblclick	Triggers on a mouse double-click
ondrag	Triggers when an element is dragged
ondragend	Triggers at the end of a drag operation
ondragenter	Triggers when an element has been dragged to a valid drop target
ondragleave	Triggers when an element is being dragged over a valid drop target
ondragover	Triggers at the start of a drag operation
ondragstart	Triggers at the start of a drag operation
ondrop	Triggers when dragged element is being dropped
ondurationchange	Triggers when the length of the media is changed
onemptied	Triggers when a media resource element suddenly becomes empty.
onended	Triggers when media has reach the end
onerror	Triggers when an error occur
onfocus	Triggers when the window gets focus
onformchange	Triggers when a form changes
onforminput	Triggers when a form gets user input
onhaschange	Triggers when the document has change
oninput	Triggers when an element gets user input
oninvalid	Triggers when an element is invalid
onkeydown	Triggers when a key is pressed
onkeypress	Triggers when a key is pressed and released

onkeyup	Triggers when a key is released
onload	Triggers when the document loads
onloadeddata	Triggers when media data is loaded
onloadedmetadata	Triggers when the duration and other media data of a media element is loaded
onloadstart	Triggers when the browser starts to load the media data
onmessage	Triggers when the message is triggered
onmousedown	Triggers when a mouse button is pressed
onmousemove	Triggers when the mouse pointer moves
onmouseout	Triggers when the mouse pointer moves out of an element
onmouseover	Triggers when the mouse pointer moves over an element
onmouseup	Triggers when a mouse button is released
onmousewheel	Triggers when the mouse wheel is being rotated
onoffline	Triggers when the document goes offline
onoine	Triggers when the document comes online
ononline	Triggers when the document comes online
onpagehide	Triggers when the window is hidden
onpageshow	Triggers when the window becomes visible
onpause	Triggers when media data is paused
onplay	Triggers when media data is going to start playing
onplaying	Triggers when media data has start playing
onpopstate	Triggers when the window's history changes
onprogress	Triggers when the browser is fetching the media data
onratechange	Triggers when the media data's playing rate has changed
onreadystatechange	Triggers when the ready-state changes
onredo	Triggers when the document performs a redo
onresize	Triggers when the window is resized

onscroll	Triggers when an element's scrollbar is being scrolled
onseeked	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
onseeking	Triggers when a media element's seeking attribute is true, and the seeking has begun
onselect	Triggers when an element is selected
onstalled	Triggers when there is an error in fetching media data
onstorage	Triggers when a document loads
onsubmit	Triggers when a form is submitted
onsuspend	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	Triggers when media changes its playing position
onundo	Triggers when a document performs an undo
onunload	Triggers when the user leaves the document
onvolumechange	Triggers when media changes the volume, also when volume is set to "mute"
onwaiting	Triggers when media has stopped playing, but is expected to resume

JavaScript - Objects Overview

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.

```
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.

```
var employee =newObject();
var books =newArray("C++","Perl","Java");
var day =newDate("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>
<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
        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.

```
<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>
```

OUTPUT

```
Book title is : Perl
Book author is : Mohtashim
```

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>

<head>
<title>User-defined objects</title>
<script type = "text/javascript">
        // Define a function which will work as a method
        function addPrice(amount) {
          this.price = amount;
```

```
        }

        function book(title, author) {
          this.title = title;
          this.author  = author;
          this.addPrice = addPrice;  // Assign that method as property.
        }
</script>

</head>
<body>

<script type = "text/javascript">
        var myBook = new book("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 '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 −

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

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

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

JavaScript Native Objects

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects −

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

JavaScript - The Number Object

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 −

```
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.

Sr.No.	Property & Description
1	**MAX_VALUE** The largest possible value a number in JavaScript can have 1.7976931348623157E+308
2	**MIN_VALUE** The smallest possible value a number in JavaScript can have 5E-324
3	**NaN** Equal to a value that is not a number.
4	**NEGATIVE_INFINITY** A value that is less than MIN_VALUE.
5	**POSITIVE_INFINITY** A value that is greater than MAX_VALUE
6	**prototype** 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
7	**constructor** Returns the function that created this object's instance. By default this is the Number object.

In the following sections, we will take a few examples to demonstrate the properties of Number.

Number Methods

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

Sr.No.	Method & Description
1	**toExponential()** Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
2	**toFixed()** Formats a number with a specific number of digits to the right of the decimal.
3	**toLocaleString()** Returns a string value version of the current number in a format that may vary according to a browser's local settings.
4	**toPrecision()** Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
5	**toString()** Returns the string representation of the number's value.

6	**valueOf()** Returns the number's value.

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

JavaScript - The Boolean Object

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

Syntax

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

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

Boolean Properties

Here is a list of the properties of Boolean object −

Sr.No.	Property & Description
1	**constructor** Returns a reference to the Boolean function that created the object.
2	**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 properties of Boolean object.

Boolean Methods

Here is a list of the methods of Boolean object and their description.

Sr.No.	Method & Description
1	**toSource()** Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	**toString()** Returns a string of either "true" or "false" depending upon the value of the object.
3	**valueOf()** Returns the primitive value of the Boolean object.

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

JavaScript - The Strings Object

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 −

```
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.

Sr.No.	Property & Description
1	**constructor** Returns a reference to the String function that created the object.
2	**length** Returns the length of the string.
3	**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 demonstrate the usage of String properties.

String Methods

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

Sr.No.	Method & Description
1	**charAt()** Returns the character at the specified index.
2	**charCodeAt()** Returns a number indicating the Unicode value of the character at the given index.
3	**concat()** Combines the text of two strings and returns a new string.
4	**indexOf()** Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	**lastIndexOf()** Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	**localeCompare()** Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	**match()** Used to match a regular expression against a string.
8	**replace()** Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
9	**search()** Executes the search for a match between a regular expression and a specified string.
10	**slice()**

	Extracts a section of a string and returns a new string.
11	**split()** Splits a String object into an array of strings by separating the string into substrings.
12	**substr()** Returns the characters in a string beginning at the specified location through the specified number of characters.
13	**substring()** Returns the characters in a string between two indexes into the string.
14	**toLocaleLowerCase()** The characters within a string are converted to lower case while respecting the current locale.
15	**toLocaleUpperCase()** The characters within a string are converted to upper case while respecting the current locale.
16	**toLowerCase()** Returns the calling string value converted to lower case.
17	**toString()** Returns a string representing the specified object.
18	**toUpperCase()** Returns the calling string value converted to uppercase.
19	**valueOf()** Returns the primitive value of the specified object.

String HTML Wrappers

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

Sr.No.	Method & Description
1	**anchor()** Creates an HTML anchor that is used as a hypertext target.
2	**big()** Creates a string to be displayed in a big font as if it were in a <big> tag.
3	**blink()** Creates a string to blink as if it were in a <blink> tag.
4	**bold()** Creates a string to be displayed as bold as if it were in a <b> tag.

5	**fixed()** Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag
6	**fontcolor()** Causes a string to be displayed in the specified color as if it were in a <font color="color"> tag.
7	**fontsize()** Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag.
8	**italics()** Causes a string to be italic, as if it were in an <i> tag.
9	**link()** Creates an HTML hypertext link that requests another URL.
10	**small()** Causes a string to be displayed in a small font, as if it were in a <small> tag.
11	**strike()** Causes a string to be displayed as struck-out text, as if it were in a <strike> tag.
12	**sub()** Causes a string to be displayed as a subscript, as if it were in a <sub> tag
13	**sup()** Causes a string to be displayed as a superscript, as if it were in a <sup> tag

JavaScript - The Arrays Object

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 −

```
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 array by simply assigning values as follows −

```
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.

Sr.No.	Property & Description
1	**constructor** Returns a reference to the array function that created the object.
2	**index** The property represents the zero-based index of the match in the string
3	**input** This property is only present in arrays created by regular expression matches.
4	**length** Reflects the number of elements in an array.
5	**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.

Array Methods

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

Sr.No.	Method & Description
1	**concat()** Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	**every()** Returns true if every element in this array satisfies the provided testing function.
3	**filter()** Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	**forEach()** Calls a function for each element in the array.
5	**indexOf()** Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	**join()** Joins all elements of an array into a string.

7	**lastIndexOf()** Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	**map()** Creates a new array with the results of calling a provided function on every element in this array.
9	**pop()** Removes the last element from an array and returns that element.
10	**push()** Adds one or more elements to the end of an array and returns the new length of the array.
11	**reduce()** Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	**reduceRight()** Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13	**reverse()** Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	**shift()** Removes the first element from an array and returns that element.
15	**slice()** Extracts a section of an array and returns a new array.
16	**some()** Returns true if at least one element in this array satisfies the provided testing function.
17	**toSource()** Represents the source code of an object
18	**sort()** Sorts the elements of an array
19	**splice()** Adds and/or removes elements from an array.
20	**toString()** Returns a string representing the array and its elements.
21	**unshift()**

	Adds one or more elements to the front of an array and returns the new length of the array.

JavaScript - The Math Object

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

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

```
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.

Sr.No.	Property & Description
1	**E ** Euler's constant and the base of natural logarithms, approximately 2.718.
2	**LN2** Natural logarithm of 2, approximately 0.693.
3	**LN10** Natural logarithm of 10, approximately 2.302.
4	**LOG2E** Base 2 logarithm of E, approximately 1.442.
5	**LOG10E** Base 10 logarithm of E, approximately 0.434.
6	**PI** Ratio of the circumference of a circle to its diameter, approximately 3.14159.
7	**SQRT1_2** Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
8	**SQRT2** Square root of 2, approximately 1.414.

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

Math Methods

Here is a list of the methods associated with Math object and their description

Sr.No.	Method & Description
1	**abs()**

	Returns the absolute value of a number.
2	**acos()** Returns the arccosine (in radians) of a number.
3	**asin()** Returns the arcsine (in radians) of a number.
4	**atan()** Returns the arctangent (in radians) of a number.
5	**atan2()** Returns the arctangent of the quotient of its arguments.
6	**ceil()** Returns the smallest integer greater than or equal to a number.
7	**cos()** Returns the cosine of a number.
8	**exp()** Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	**floor()** Returns the largest integer less than or equal to a number.
10	**log()** Returns the natural logarithm (base E) of a number.
11	**max()** Returns the largest of zero or more numbers.
12	**min()** Returns the smallest of zero or more numbers.
13	**pow()** Returns base to the exponent power, that is, base exponent.
14	**random()** Returns a pseudo-random number between 0 and 1.
15	**round()** Returns the value of a number rounded to the nearest integer.
16	**sin()** Returns the sine of a number.
17	**sqrt()**

	Returns the square root of a number.
18	**tan()** Returns the tangent of a number.
19	**toSource()** Returns the string "Math".

JavaScript - Document Object Model or DOM

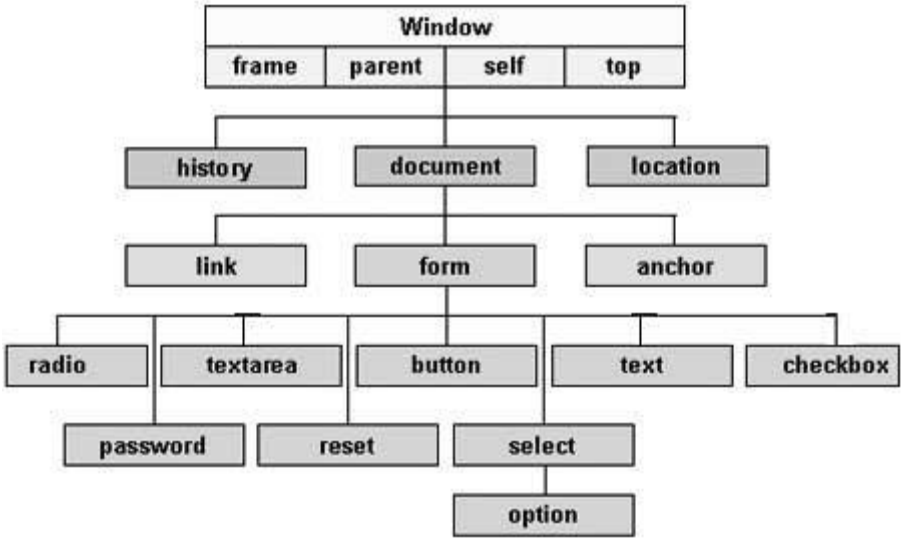
Every web page resides inside a browser window which can be considered as an object.

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.

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 −

```
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
}
```

JavaScript - The Date Object

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.

```
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 agruments** − 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.

Sr.No.	Property & Description
1	**constructor** Specifies the function that creates an object's prototype.
2	**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 demonstrate the usage of different Date properties.

Date Methods

Here is a list of the methods used with **Date** and their description.

Sr.No.	Method & Description
1	**Date()** Returns today's date and time
2	**getDate()** Returns the day of the month for the specified date according to local time.
3	**getDay()** Returns the day of the week for the specified date according to local time.
4	**getFullYear()** Returns the year of the specified date according to local time.
5	**getHours()** Returns the hour in the specified date according to local time.
6	**getMilliseconds()** Returns the milliseconds in the specified date according to local time.
7	**getMinutes()** Returns the minutes in the specified date according to local time.
8	**getMonth()** Returns the month in the specified date according to local time.
9	**getSeconds()** Returns the seconds in the specified date according to local time.
10	**getTime()** Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
11	**getTimezoneOffset()** Returns the time-zone offset in minutes for the current locale.

12	**getUTCDate()** Returns the day (date) of the month in the specified date according to universal time.
13	**getUTCDay()** Returns the day of the week in the specified date according to universal time.
14	**getUTCFullYear()** Returns the year in the specified date according to universal time.
15	**getUTCHours()** Returns the hours in the specified date according to universal time.
16	**getUTCMilliseconds()** Returns the milliseconds in the specified date according to universal time.
17	**getUTCMinutes()** Returns the minutes in the specified date according to universal time.
18	**getUTCMonth()** Returns the month in the specified date according to universal time.
19	**getUTCSeconds()** Returns the seconds in the specified date according to universal time.
20	**getYear()** **Deprecated** - Returns the year in the specified date according to local time. Use getFullYear instead.
21	**setDate()** Sets the day of the month for a specified date according to local time.
22	**setFullYear()** Sets the full year for a specified date according to local time.
23	**setHours()** Sets the hours for a specified date according to local time.
24	**setMilliseconds()** Sets the milliseconds for a specified date according to local time.
25	**setMinutes()** Sets the minutes for a specified date according to local time.
26	**setMonth()** Sets the month for a specified date according to local time.
27	**setSeconds()** Sets the seconds for a specified date according to local time.

28	**setTime()** Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
29	**setUTCDate()** Sets the day of the month for a specified date according to universal time.
30	**setUTCFullYear()** Sets the full year for a specified date according to universal time.
31	**setUTCHours()** Sets the hour for a specified date according to universal time.
32	**setUTCMilliseconds()** Sets the milliseconds for a specified date according to universal time.
33	**setUTCMinutes()** Sets the minutes for a specified date according to universal time.
34	**setUTCMonth()** Sets the month for a specified date according to universal time.
35	**setUTCSeconds()** Sets the seconds for a specified date according to universal time.
36	**setYear()** **Deprecated -** Sets the year for a specified date according to local time. Use setFullYear instead.
37	**toDateString()** Returns the "date" portion of the Date as a human-readable string.
38	**toGMTString()** **Deprecated -** Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
39	**toLocaleDateString()** Returns the "date" portion of the Date as a string, using the current locale's conventions.
40	**toLocaleFormat()** Converts a date to a string, using a format string.
41	**toLocaleString()** Converts a date to a string, using the current locale's conventions.
42	**toLocaleTimeString()** Returns the "time" portion of the Date as a string, using the current locale's conventions.
43	**toSource()**

	Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
44	**toString()** Returns a string representing the specified Date object.
45	**toTimeString()** Returns the "time" portion of the Date as a human-readable string.
46	**toUTCString()** Converts a date to a string, using the universal time convention.
47	**valueOf()** Returns the primitive value of a Date object.

Converts a date to a string, using the universal time convention.

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.

Sr.No.	Method & Description
1	**Date.parse( )** Parses a string representation of a date and time and returns the internal millisecond representation of that date.
2	**Date.UTC( )** Returns the millisecond representation of the specified UTC date and time.

Regular Expressions and RegExp Object

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 −

```
var pattern = new RegExp(pattern, attributes);
or simply
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 multi-line 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.

Sr.No.	Expression & Description
1	**[...]**

	Any one character between the brackets.
2	**[^...]** Any one character not between the brackets.
3	**[0-9]** It matches any decimal digit from 0 through 9.
4	**[a-z]** It matches any character from lowercase **a** through lowercase **z**.
5	**[A-Z]** It matches any character from uppercase **A** through uppercase **Z**.
6	**[a-Z]** It matches any character from lowercase **a** through uppercase **Z**.

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.

Sr.No.	Expression & Description
1	**p+** It matches any string containing one or more p's.
2	**p*** It matches any string containing zero or more p's.
3	**p?** It matches any string containing at most one p.
4	**p{N}** It matches any string containing a sequence of **N** p's
5	**p{2,3}** It matches any string containing a sequence of two or three p's.
6	**p{2, }** It matches any string containing a sequence of at least two p's.
7	**p$** It matches any string with p at the end of it.
8	**^p** It matches any string with p at the beginning of it.

Examples

Following examples explain more about matching characters.

Sr.No.	Expression & Description
1	**[^a-zA-Z]** It matches any string not containing any of the characters ranging from **a** through **z** and **A** through Z.
2	**p.p** It matches any string containing **p,** followed by any character, in turn followed by another **p**.
3	**^.{2}$** It matches any string containing exactly two characters.
4	**(.*)** It matches any string enclosed within and .
5	**p(hp)*** It matches any string containing a **p** followed by zero or more instances of the sequence **hp**.

Literal characters

Sr.No.	Character & Description
1	**Alphanumeric** Itself
2	**\0** The NUL character (\u0000)
3	**\t** Tab (\u0009
4	**\n** Newline (\u000A)
5	**\v** Vertical tab (\u000B)
6	**\f** Form feed (\u000C)
7	**\r** Carriage return (\u000D)
8	**\xnn** The Latin character specified by the hexadecimal number nn; for example, \x0A is the same as \n
9	**\uxxxx** The Unicode character specified by the hexadecimal number xxxx; for example, \u0009 is the same

	as \t
10	**\cX** The control character ^X; for example, \cJ is equivalent to the newline character \n

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.

Sr.No.	Character & Description
1	**.** a single character
2	**\s** a whitespace character (space, tab, newline)
3	**\S** non-whitespace character
4	**\d** a digit (0-9)
5	**\D** a non-digit
6	**\w** a word character (a-z, A-Z, 0-9, _)
7	**\W** a non-word character
8	**[\b]** a literal backspace (special case).
9	**[aeiou]** matches a single character in the given set
10	**[^aeiou]** matches a single character outside the given set
11	**(foo|bar|baz)** matches any of the alternatives specified

Modifiers

Several modifiers are available that can simplify the way you work with **regexps,** like case sensitivity, searching in multiple lines, etc.

Sr.No.	Modifier & Description
1	**i** Perform case-insensitive matching.
2	**m** 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
3	**g** Performs a global matchthat is, find all matches rather than stopping after the first match.

RegExp Properties

Here is a list of the properties associated with RegExp and their description.

Sr.No.	Property & Description
1	**constructor** Specifies the function that creates an object's prototype.
2	**global** Specifies if the "g" modifier is set.
3	**ignoreCase** Specifies if the "i" modifier is set.
4	**lastIndex** The index at which to start the next match.
5	**multiline** Specifies if the "m" modifier is set.
6	**source** The text of the pattern.

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

RegExp Methods

Here is a list of the methods associated with RegExp along with their description.

Sr.No.	Method & Description
1	**exec()** Executes a search for a match in its string parameter.
2	**test()** Tests for a match in its string parameter.
3	**toSource()** Returns an object literal representing the specified object; you can use this value to create a new object.

4	**toString()** Returns a string representing the specified object.

JavaScript - Errors & Exceptions Handling

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.

```
<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.

```
<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 −

```
<scripttype="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>
```

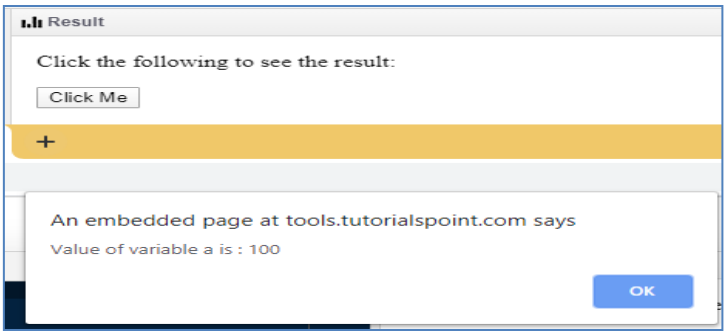
The **try** block must be followed by either exactly one **catch** block or one **finally** block (or one of both). When an exception occurs in the **try** block, the exception is placed in **e** and the **catch** block is executed. The optional **finally** block executes unconditionally after try/catch.

Examples

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>
<head>
<scripttype="text/javascript">
<!--
function myFunc(){
var a =100;
        alert("Value of variable a is : "+ a );
}
//-->
</script>
</head>

<body>
<p>Click the following to see the result:</p>

<form>
<inputtype="button"value="Click Me"onclick="myFunc();"/>
</form>
</body>
</html>
```

Output



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>
<head>

<scripttype="text/javascript">
<!--
function myFunc(){
var a =100;
try{
        alert("Value of variable a is : "+ a );
}
catch( e ){
        alert("Error: "+ e.description );
}
}
//-->
</script>

</head>
<body>
<p>Click the following to see the result:</p>

<form>
<inputtype="button"value="Click Me"onclick="myFunc();"/>
</form>

</body>
</html>
```

You can use **finally** block which will always execute unconditionally after the try/catch. Here is an example.

```
<html>
<head>

<scripttype="text/javascript">
<!--
function myFunc(){
var a =100;

try{
        alert("Value of variable a is : "+ a );
}
catch( e ){
        alert("Error: "+ e.description );
}
      finally {
        alert("Finally block will always execute!");
}
}
```

```
//-->
</script>

</head>
<body>
<p>Click the following to see the result:</p>

<form>
<inputtype="button"value="Click Me"onclick="myFunc();"/>
</form>

</body>
</html>
```

The throw Statement

You can use **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>
<head>

<scripttype="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 ){
                alert("Error: "+ e );
}
}
//-->
</script>

</head>
<body>
<p>Click the following to see the result:</p>

<form>
<inputtype="button"value="Click Me"onclick="myFunc();"/>
</form>
```

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

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.

```
<html>
<head>

<scripttype="text/javascript">
<!--
        window.onerror =function(){
          alert("An error occurred.");
}
//-->
</script>

</head>
<body>
<p>Click the following to see the result:</p>

<form>
<inputtype="button"value="Click Me"onclick="myFunc();"/>
</form>

</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>
<head>

<scripttype="text/javascript">
<!--
        window.onerror =function(msg, url, line){
          alert("Message : "+ msg );
          alert("url : "+ url );
          alert("Line number : "+ line );
}
//-->
```

```
</script>

</head>
<body>
<p>Click the following to see the result:</p>

<form>
<inputtype="button"value="Click Me"onclick="myFunc();"/>
</form>

</body>
</html>
```

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.

```
<imgsrc="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.

JavaScript - Form Validation

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.

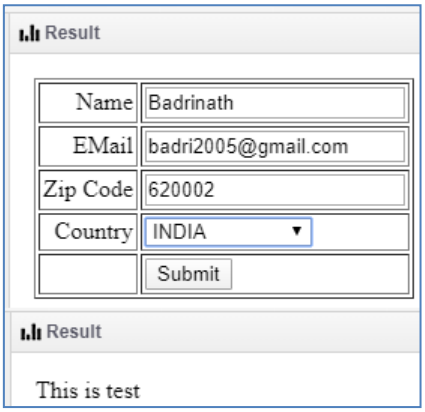
Example

We will take an example to understand the process of validation. Here is a simple form in html format.

```
<html>
<head>
<title>Form Validation</title>
<scripttype="text/javascript">
<!--
// Form validation code will come here.
//-->
</script>
</head>

<body>
<formaction="/cgi-bin/test.cgi"name="myForm"onsubmit="return(validate());">
<tablecellspacing="2"cellpadding="2"border="1">

<tr>
<tdalign="right">Name</td>
<td><inputtype="text"name="Name"/></td>
```

```
</tr>

<tr>
<tdalign="right">EMail</td>
<td><inputtype="text"name="EMail"/></td>
</tr>

<tr>
<tdalign="right">Zip Code</td>
<td><inputtype="text"name="Zip"/></td>
</tr>

<tr>
<tdalign="right">Country</td>
<td>
<selectname="Country">
<optionvalue="-1"selected>[choose yours]</option>
<optionvalue="1">USA</option>
<optionvalue="2">UK</option>
<optionvalue="3">INDIA</option>
</select>
</td>
</tr>

<tr>
<tdalign="right"></td>
<td><inputtype="submit"value="Submit"/></td>
</tr>

</table>
</form>
</body>
</html>
```

OUTPUT



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.

```
<scripttype="text/javascript">
<!--
// Form validation code will come here.
function validate(){

if( document.myForm.Name.value ==""){
        alert("Please provide your name!");
        document.myForm.Name.focus();
returnfalse;
}
if( document.myForm.EMail.value ==""){
        alert("Please provide your Email!");
        document.myForm.EMail.focus();
returnfalse;
}
if( document.myForm.Zip.value ==""|| isNaN( document.myForm.Zip.value )||
        document.myForm.Zip.value.length !=5){

        alert("Please provide a zip in the format #####.");
        document.myForm.Zip.focus();
returnfalse;
}
if( document.myForm.Country.value =="-1"){
        alert("Please provide your country!");
returnfalse;
}
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.

```
<scripttype="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();
returnfalse;
```

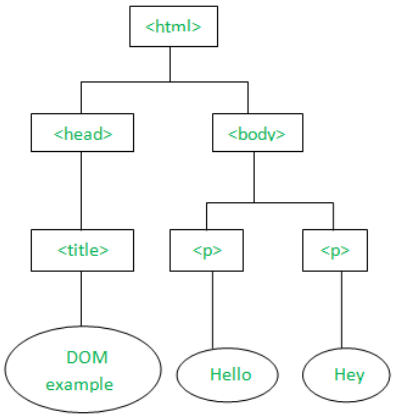
```
}
return(true);
}
//-->
</script>
```

DHTML JavaScript

DHTML stands for Dynamic HTML. Dynamic means that the content of the web page can be customized or changed according to user inputs i.e. a page that is interactive with the user. In earlier times, HTML was used to create a static page. It only defined the structure of the content that was displayed on the page. With the help of CSS, we can beautify the HTML page by changing various properties like text size, background color etc. The HTML and CSS could manage to navigate between static pages but couldn't do anything else. If 1000 users view a page that had their information for eg. Admit card then there was a problem because 1000 static pages for this application build to work. As the number of users increase, the problem also increases and at some point it becomes impossible to handle this problem.

To overcome this problem, DHTML came into existence. DHTML included JavaScript along with HTML and CSS to make the page dynamic. This combo made the web pages dynamic and eliminated this problem of creating static page for each user. To integrate JavaScript into HTML, a Document Object Model(DOM) is made for the HTML document. In DOM, the document is represented as nodes and objects which are accessed by different languages like JavaScript to manipulate the document.

```
<html>
<head>
<title>
 DOM example
</title>
</head>
<body>
<p id = "para1">Hello</p>
<p id =  “para2”>Hey</p>
</body>
</html>
```



HTML document include JavaScript:: The JavaScript document is included in our html page using the html tag. <src> tag is used to specify the source of external JavaScript file. Following are some of the tasks that can be performed with JavaScript:

- Performing html tasks
- Performing CSS tasks
- Handling events
- Validating inputs

Example 1: Example to understand how to use JavaScript in DHTML.

```
<html>
   <head>
         <title>DOM programming</title>
```

```
    </head>

    <body>
        <h1>GeeksforGeeks</h1>
        <p id = "geeks">Hello Geeks!</p>
        <script style = "text/javascript">
        document.getElementById("geeks").innerHTML =
            "A computer science portal for geeks";
        </script>
    </body>
</html>
```

Output

Explanation: In the above example, change the text of paragraph which using id. Document is an object of html that is displayed in the current window or object of DOM. The getElementById(id) gives the element id. The innerHTML defines the content within the id element. The id attribute is used to change HTML document and its property. Paragraph content changed by document id. For example: document.getElementById("geeks").style.color = "blue"; It is used to set the paragraph color using id of elements.

Example 2: Creating an alert on click of a button.

```
<html>
    <head>
        <title>Create an alert</title>
    </head>

    <body>
        <h1 id = "para1">GeeksforGeeks</h1>
        <input type = "Submit" onclick = "Click()"/>
        <script style = "text/javascript">
            function Click() {
                document.getElementById("para1").style.color = "#009900";
                window.alert("Color changed to green");
            }
        </script>
    </body>
</html>
```

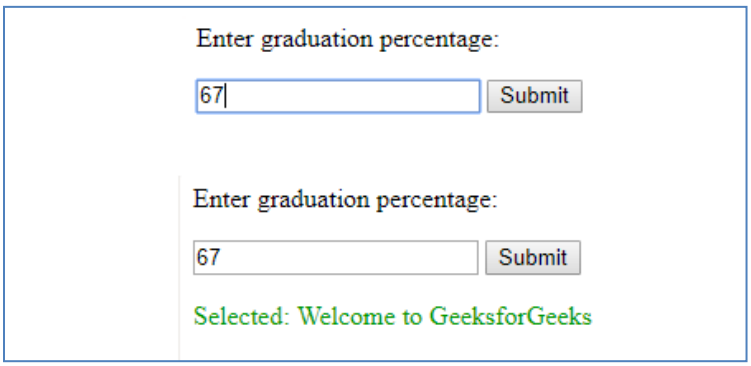
Output

Explanation: In this example, creating a function that will be invoked on click of a button and it changes the color of text and display the alert on the screen. window is an object of current window whose inbuilt method alert() is invoked in Click() function.

Example 3: Validate input data using JavaScript.

```
<html>
   <head>
         <title>Validate input data</title>
   </head>

   <body>
         <p>Enter graduation percentage:</p>
         <input id="perc">
         <button type="button" onclick="Validate()">Submit</button>
         <p id="demo"></p>
         <script>
               function Validate() {
                     var x, text;
                     x = document.getElementById("perc").value;
                     if (isNaN(x) || x < 60) {
                           window.alert("Not selected in GeeksforGeeks");
                     } else {
                     document.getElementById("demo").innerHTML =
                                       "Selected: Welcome to GeeksforGeeks";
                     document.getElementById("demo").style.color = "#009900";
                     }
               }
         </script>
   </body>
</html>
```

Output



Explanation: In this example, make a validate() function which ensures the user is illegible or not. If user enters > 60 then selected otherwise not selected.

JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<elementevent='some JavaScript'>
```

With double quotes:

```
<elementevent="some JavaScript">
```

In the following example, an `onclick` attribute (with code), is added to a `<button>` element:

Example

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using `this.innerHTML`):

Example

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

JavaScript code is often several lines long. It is more common to see event attributes calling functions

Example

```
<button onclick="displayDate()">The time is?</button>
```

Window frames

Example

Change the location of the first frame:

```
window.frames[0].location = "https://www.w3schools.com/jsref/";
```

Definition and Usage

The `frames` property returns an array with all window objects in the window.

The `frames` property is read-only.

The windows can be accessed by index numbers. The first index is 0.

The Window History Object

The **history object** contains the URLs visited by the user (in the browser window).

The **history object** is a property of the **window object**.

The **history object** is accessed with:

`window.history` or just `history`:

Examples

```
let length = window.history.length;

let length = history.length;
```

History Object Properties and Methods

Property/Method	Description
back()	Loads the previous URL (page) in the history list
forward()	Loads the next URL (page) in the history list

go()	Loads a specific URL (page) from the history list
length	Returns the number of URLs (pages) in the history list

Window getComputedStyle()

Example

Get the computed background color of an element:

```
const element = document.getElementById("test");
const cssObj = window.getComputedStyle(element, null);

let bgColor = cssObj.getPropertyValue("background-color");
```

More examples below.

Definition and Usage

The `getComputedStyle()` method gets the computed CSS properties and values of an HTML element.

The `getComputedStyle()` method returns a `CSSStyleDeclaration object`.

Computed Style

The computed style is the style used on the element after all styling sources have been applied.

Style sources: external and internal style sheets, inherited styles, and browser default styles.

See Also:

The CSSStyleDeclaration Object.

Syntax

window.getComputedStyle(*element, pseudoElement*)

Parameters

Parameter	Description
element	Required. The element to get the computed style for.
pseudoElement	Optional. A pseudo-element to get.

Return Value

Type	Description
An object	A CSSStyleDeclaration object with all the CSS properties and values of the element.

More Examples

Get all the computed styles from an element:

```
const element = document.getElementById("test");
const cssObj = window.getComputedStyle(element, null);

let text = "";
for (x in cssObj) {
  cssObjProp = cssObj.item(x)
  text += cssObjProp + " = " + cssObj.getPropertyValue(cssObjProp) + "<br>";
}
```

Get computed font size of the first letter in an element (using pseudo-element):

```
const element = document.getElementById("test"); const cssObj = window.getComputedStyle(element, ":first-letter")

let size = cssObj.getPropertyValue("font-size");
```

Window innerHeight

Example

Get the window height:

```
let height = window.innerHeight;

let height = innerHeight;
```

More examples below.

Definition and Usage

The `innerHeight` property returns the height of a window's content area.

The `innerHeight` property is read only.

See Also:

The innerWidth Property.

The outerWidth Property.

The outerHeight Property.

Syntax

```
window.innerHeight

or just:

innerHeight
```

Return Value

Type	Description
A number	The the inner height of the browser window's content area in pixels.

More Examples

All height and width properties:

```
let text =
"<p>innerWidth: " + window.innerWidth + "</p>" +
"<p>innerHeight: " + window.innerHeight + "</p>" +
"<p>outerWidth: " + window.outerWidth + "</p>" +
"<p>outerHeight: " + window.outerHeight + "</p>";
```

Window innerWidth

Example

Get window width:

```
let width = window.innerWidth;
```

```
let width = innerWidth;
```

More examples below.

Definition and Usage

The `innerWidth` property returns the width of a window's content area.

The `innerWidth` property is read-only.

See Also:

The innerHeight Property.

The outerWith Property.

The outerHeight Property.

Syntax

window.innerWidth

or just:

innerWidth

Return Value

Type	Description
A number	The the inner width of the browser window's content area in pixels.

More Examples

All height and width properties:

```
let text =
"<p>innerWidth: " + window.innerWidth + "</p>" +
"<p>innerHeight: " + window.innerHeight + "</p>" +
"<p>outerWidth: " + window.outerWidth + "</p>" +
"<p>outerHeight: " + window.outerHeight + "</p>";
```

Window length

Example

How many windows are in the window:

```
let length = window.length;
```

More examples below.

Definition and Usage

The `length` property returns the number of (framed) windows in the window.

The `length` property is read-only.

The windows can be accessed by index numbers. The first index is 0.

Note

A frame can be any embedding element:

<frame>, <iframe>, <embed>, <object>, etc.

See Also:

The frames property

The frameElement Property

Syntax

window.length

Return Value

Type	Description
A number	The number of windows in the current window.

Window localStorage

Example

Set and retrieve localStorage name/value pair:

```
localStorage.setItem("lastname", "Smith");
localStorage.getItem("lastname");
```

More examples below.

Definition and Usage

The `localStorage` object allows you to save key/value pairs in the browser.

Note

The localStorage object stores data with no expiration date.

The data is not deleted when the browser is closed, and are available for future sessions.

See Also:

The sessionStorage Object which stores data for one session.

(The data is deleted when the browser window is closed)

Syntax

window.localStorage

or just:

localStorage

Save Data to Local Storage

localStorage.setItem(*key*, *value*);

Read Data from Local Storage

let lastname = localStorage.getItem(*key*);

Remove Data from Local Storage

localStorage.removeItem(*key*);

Remove All (Clear Local Storage)

localStorage.clear();

Parameters

Parameter	Description
key	Required. The name of a key.
value	Required. The value of the key.

Return Value

Type	Description
An object	A localStorage object.

More Examples

Count the number of times a user has clicked a button:

```
if (localStorage.clickcount) {
  localStorage.clickcount = Number(localStorage.clickcount) + 1;
} else {
  localStorage.clickcount = 1;
}
```

Window Location

The Window Location Object

The **location object** contains information about the current URL.

The **location object** is a property of the **window object**.

The **location object** is accessed with:

`window.location` or just `location`

Examples

```
let origin = window.location.origin;

let origin = location.origin;
```

Location Object Properties

Property	Description
hash	Sets or returns the anchor part (#) of a URL
host	Sets or returns the hostname and port number of a URL
hostname	Sets or returns the hostname of a URL
href	Sets or returns the entire URL
origin	Returns the protocol, hostname and port number of a URL
pathname	Sets or returns the path name of a URL
port	Sets or returns the port number of a URL
protocol	Sets or returns the protocol of a URL

search	Sets or returns the querystring part of a URL

Location Object Methods

Method	Description
assign()	Loads a new document
reload()	Reloads the current document
replace()	Replaces the current document with a new one

Window moveBy()

Example

Open a new window and move it 250px relative to its current position:

```
function openWin(){
  myWindow = window.open('', '', 'width=400, height=400');
}

function moveWin(){
  myWindow.moveBy(250, 250);
}
```

More examples below.

Definition and Usage

The `moveBy()` method moves a window a number of pixels relative to its current coordinates.

See Also:

The moveTo() Method

The resizeBy() Method

The resizeTo() Method

Syntax

window.moveBy(*x, y*)

Parameters

Parameter	Description
x	Required. A positive or negative number. The number of pixels to move the window horizontally.
y	Required. A positive or negative number. The number of pixels to move the window vertically.

Return Value

NONE

Window moveTo()

Example

Open a new window, and move it to position 500 x 100:

```
function openWin(){
  myWindow = window.open('', '', 'width=400, height=200');
}

function moveWin(){
  myWindow.moveTo(500, 100);
}
```

More examples below.

Definition and Usage

The `moveTo()` method moves a window to the specified coordinates.

See Also:

The moveBy() Method

The resizeBy() Method

The resizeTo() Method

Syntax

window.moveTo(*x, y*)

Parameters

Parameter	Description
x	Required. A positive or negative number. The horizontal coordinate to move to.
y	Required. A positive or negative number. The vertical coordinate to move to.

Return Value

NONE

More Examples

Using moveTo() together with moveBy():

```
function moveWinTo() {
  myWindow.moveTo(150, 150);
}

function moveWinBy() {
  myWindow.moveBy(75, 50);
}
```

Window name

Examples

Get window name:

```
let name = window.name;
```

Set window name:

```
window.name = "myWindowName";
```

More examples below.

Definition and Usage

The `name` property sets or returns the name of the window.

Note

A windows does not need to have a name.

Syntax

Return the name property:

```
window.name
```

Set the name property:

```
window.name = winName
```

Property Value

Type	Description
winName	The name of the window.

Return Value

Type	Description
A string	The name of the window. Or "view" (If the window has no name).

More Examples

Open a frame with a special name:

```
const otherWindow = window.open();
otherWindow.name = "Butterfly";
```

Window Navigator

The Window Navigator Object

The **navigator object** contains information about the browser.

The **navigator object** is a property of the **window object**.

The **navigator object** is accessed with:

`window.navigator` or just `navigator`:

Examples

```
let url = window.navigator.language;
```

```
let url = navigator.language;
```

Navigator Object Properties

Property	Description
appCodeName	Returns browser code name
appName	Returns browser name
appVersion	Returns browser version

cookieEnabled	Returns true if browser cookies are enabled
geolocation	Returns a geolocation object for the user's location
language	Returns browser language
onLine	Returns true if the browser is online
platform	Returns browser platform
product	Returns browser engine name
userAgent	Returns browser user-agent header

Navigator Object Methods

Method	Description
javaEnabled()	Returns true if the browser has Java enabled
taintEnabled()	Removed in JavaScript version 1.2 (1999).

Window open()

Example 1

Open "www.w3schools.com" in a new browser tab:

```
window.open("https://www.w3schools.com");
```

More examples below.

Definition and Usage

The `open()` method opens a new browser window, or a new tab, depending on your browser settings and the parameter values.

See Also:

The close() method.

Syntax

window.open(*URL, name, specs, replace*)

Parameters

Parameter	Description
URL	Optional. The URL of the page to open. If no URL is specified, a new blank window/tab is opened
name	Optional. The target attribute or the name of the window. The following values are supported:

Value	Description
_blank	URL is loaded into a new window, or tab. This is the default
_parent	URL is loaded into the parent frame
_self	URL replaces the current page
_top	URL replaces any framesets that may be loaded
name	The name of the window (does not specify the title of the window)

specs	Optional. A comma-separated list of items, no whitespaces. The following values are supported:
fullscreen=yes|no|1|0	Whether or not to display the browser in full-screen mode. Default is no. A window in full-screen mode must also be in theater mode. IE only
height=pixels	The height of the window. Min. value is 100
left=pixels	The left position of the window. Negative values not allowed
location=yes|no|1|0	Whether or not to display the address field. Opera only
menubar=yes|no|1|0	Whether or not to display the menu bar

resizable=yes|no|1|0	Whether or not the window is resizable. IE only
scrollbars=yes|no|1|0	Whether or not to display scroll bars. IE, Firefox & Opera only
status=yes|no|1|0	Whether or not to add a status bar
titlebar=yes|no|1|0	Whether or not to display the title bar. Ignored unless the calling application is an HTML Application or a trusted dialog box
toolbar=yes|no|1|0	Whether or not to display the browser toolbar. IE and Firefox only
top=pixels	The top position of the window. Negative values not allowed
width=pixels	The width of the window. Min. value is 100
replace	**Deprecated** Specifies whether the URL creates a new entry or replaces the current entry in the history list. The following values are supported: • true - URL replaces the current document in the history list • false - URL creates a new entry in the history list

Window print() Method

Example

Print the current page:

```
window.print();
```

Definition and Usage

The print() method prints the contents of the current window.

The print() method opens the Print Dialog Box, which lets the user to select preferred printing options.

Syntax

window.print()

Parameters

None

Technical Details

Return Value:	No return value

Window prompt()

Example 1

Prompt for a user name and output a message:

```
let person = prompt("Please enter your name", "Harry Potter");

if (person != null) {
  document.getElementById("demo").innerHTML =
  "Hello " + person + "! How are you today?";
}
```

More examples below.

Definition and Usage

The `prompt()` method displays a dialog box that prompts the user for input.

The `prompt()` method returns the input value if the user clicks "OK", otherwise it returns `null`.

Note

A prompt box is used if you want the user to input a value.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed.

Do not overuse this method. It prevents the user from accessing other parts of the page until the box is closed.

See Also:

The alert() Method

The confirm() Method

Syntax

prompt(*text, defaultText*)

Parameters

Parameter	Description
text	Optional. The text to display in the dialog box.
defaultText	Optional. The default input text.

Return Value

Parameter	Description
A string	If the user clicks "OK", the input value is returned. Otherwise `null` is returned.

More Examples

Prompt for his favourite drink:

```
let text;
let favDrink = prompt("What's your favorite cocktail drink?");
switch(favDrink) {
 case"Coca-Cola":
   text = "Excellent choice! Coca-Cola is good for your soul.";
   break;
 case"Pepsi":
   text = "Pepsi is my favorite too!";
   break;
```

```
 case"Sprite":
   text = "Really? Are you sure the Sprite is your favorite?";
   break;
 default:
   text = "I have never heard of that one!";
}
```

Window removeEventListener()

Example

Remove a "mousemove" event handler:

```
window.removeEventListener("mousemove", myFunction);
```

Definition and Usage

The `removeEventListener()` method removes an event handler from a window.

Document Methods

The addEventListener() Method

The removeEventListener() Method

Element Methods

The addEventListener() Method

The removeEventListener() Method

Tutorials

HTML DOM EventListener

The Complete List of DOM Events

Syntax

window.removeEventListener(*event*, *function*, *capture*)

Parameters

Parameter	Description
event	Required. The name of the event to remove. Do not use the "on" prefix. use "click" instead of "onclick". All HTML DOM events are listed in the: HTML DOM Event Object Reference.
function	Required. The function to remove.
capture	Optional (default = false). `true` - Remove the handler from capturing. `false`- Remove the handler from bubbling. If the event handler was attached two times, one with capturing and one with bubbling, each must be removed separately.

Window resizeBy()

Example 1

Open a new window, and resize (increase) the width and height:

```
function openWin() {
  myWindow = window.open("", "", "width=100, height=100");
}

function resizeWin() {
  myWindow.resizeBy(250, 250);
}
```

Window resizeTo()

Example 1

Open a new window, and resize it to 300 x 300:

```
function openWin() {
  myWindow = window.open("", "", "width=200, height=100");
}

function resizeWin() {
  myWindow.resizeTo(300, 300);
}
```

More examples below.

Definition and Usage

The `resizeTo()` method resizes a window to a new width and height.

See Also:

The resizeBy() Method

The moveTo() Method

The moveTo() Method

Syntax

window.resizeTo(*width, height*)

Parameters

Parameter	Description
width	Required. The new window width, in pixels
height	Required. The new window height, in pixels

Return Value

NONE

More Examples

Using resizeTo() with resizeBy():

```
function resizeWinTo() {
  myWindow.resizeTo(800, 600);
}

function resizeWinBy() {
 myWindow.resizeBy(-100, -50);
}
```

Window Screen

The Window Screen Object

The screen object contains information about the visitor's screen.

Screen Object Properties

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
colorDepth	Returns the bit depth of the color palette for displaying images
height	Returns the total height of the screen
pixelDepth	Returns the color resolution (in bits per pixel) of the screen
width	Returns the total width of the screen

Window screenLeft

Example

Return the x and y coordinates of the window:

```
let x = window.screenLeft;
let y = window.screenTop;
```

Definition and Usage

The `screenLeft` property returns the x (horizontal) position of a window, relative to the screen.

See Also:

The screenTop Property

The screenX Property

The screenY Property

Syntax

```
window.screenLeft
```

Return Value

Type	Description
A number	The x (horizontal) position of the window relative to the screen, in pixels.

Window screenTop

Example

Return the x and y coordinates of the window:

```
let x = window.screenLeft;
let y = window.screenTop;
```

Definition and Usage

The `screenTop` property returns the y (vertical) position of the window relative to the screen.

See Also:

The screenLeft Property

The screenX Property

The screenY Property

Syntax

```
window.screenTop
```

Return Value

Type	Description
A number	The y (vertical) position of the window relative to the screen, in pixels.

Browser Support

`window.screenTop` is supported in all modern browsers:

Window scrollBy()

Example

Scroll the document 100px horizontally:

```
window.scrollBy(100, 0);
```

Scroll the document 100px vertically:

```
window.scrollBy(0, 100);
```

More examples below.

Definition and Usage

The `scrollBy()` method scrolls the document by the specified number of pixels.

Note

For the `scrollBy()` method to work, the document must be larger than the screen, and the scrollbar must be visible.

See Also:

The scrollTo() method.

Syntax

window.scrollBy(*x, y*)

or just:

scrollBy(*x, y*)

Parameters

Parameter	Description
x	Required. Number of pixels to scroll (horizontally). Positive values scroll to the right, negative values to the left.
y	Required. Number ofpixels to scroll (vertically). Positive values scroll down, negative values scroll up.

Return Value

NONE

More Examples

Scroll the document up and down:

```
<button onclick="scrollWin(0, 50)">Scroll down</button>
<button onclick="scrollWin(0, -50)">Scroll up</button>

<script>
function scrollWin(x, y) {
  window.scrollBy(x, y);
}
</script>
```

Scroll the document right and left:

```
<button onclick="scrollWin(100, 0)">Scroll right</button>
<button onclick="scrollWin(-100, 0)">Scroll left</button>

<script>
function scrollWin(x, y) {
  window.scrollBy(x, y);
}
</script>
```

Window scrollTo()

Example

Scroll the document to the horizontal position 500:

```
window.scrollTo(500, 0);
```

Scroll the document to the vertical position 500:

```
window.scrollTo(0, 500);
```

More examples below.

Definition and Usage

The `scrollTo()` method scrolls the document to specified coordinates.

Note

For the `scrollTo()` method to work, the document must be larger than the screen, and the scrollbar must be visible.

See Also:

The scrollBy() method.

Syntax

window.scrollTo(*x, y*)

or just:

scrollTo(*x, y*)

Parameters

Parameter	Description
x	Required. The coordinate to scroll to (horizontally), in pixels.
y	Required. The coordinate to scroll to (vertically), in pixels.

Return Value

NONE

More Examples

Scroll the document to position 300 horizontally and 500 vertically:

```
window.scrollTo(300, 500);
```

Window setTimeout()

Examples

Wait 5 seconds for the greeting:

```
const myTimeout = setTimeout(myGreeting, 5000);
```

Use clearTimeout(myTimeout) to prevent myGreeting from running:

```
const myTimeout = setTimeout(myGreeting, 5000);

function myStopFunction() {
  clearTimeout(myTimeout);
}
```

More examples below.

Definition and Usage

The `setTimeout()` method calls a function after a number of milliseconds.

1 second = 1000 milliseconds.

Notes

The `setTimeout()` is executed only once.

If you need repeated executions, use `setInterval()` instead.

Use the `clearTimeout()` method to prevent the function from starting.

To clear a timeout, use the **id** returned from setTimeout():

```
myTimeout = setTimeout(function, milliseconds);
```

Then you can to stop the execution by calling clearTimeout():

```
clearTimeout(myTimeout);
```

See Also:

The clearTimeout() Method

The setInterval() Method

[The clearInterval() Method]

Syntax

setTimeout(*function, milliseconds, param1, param2, ...*)

Parameters

Parameter	Description
function	Required. The function to execute.
milliseconds	Optional. Number of milliseconds to wait before executing. Default value is 0.
param1, *param2,* *...*	Optional. Parameters to pass to the *function.* Not supported in IE9 and earlier.

Return Value

Type	Description
A number	The id of the timer. Use this id with clearTimeout(id) to cancel the timer.

More Examples

Display an alert box after 3 seconds (3000 milliseconds):

```
let timeout;

function myFunction() {
  timeout = setTimeout(alertFunc, 3000);
}

function alertFunc() {
```

```
 alert("Hello!");
}
```

Window status

Example

Set the text in the status bar:

```
window.status = "Some text in the status bar!!";
```

Definition and Usage

The status property is deprecated.

It should be avoided to prevent RUN-TIME ERRORS in the future.

Syntax

```
window.status
```

Return Value

The text displayed in the status bar

Window stop()

Example

Stop the loading of a window:

```
window.stop();
```

Definition and Usage

The `stop()` method stops window loading.

The `stop()` method is the same as clicking stop in the browser.

Note

The `stop()` method can be used to stop loading an image if it takes too long.

Syntax

window.stop()

Parameters

NONE

Return Value

NONE

Window Console Object

The Console Object

The **console object** provides access to the browser's debugging console.

The **console object** is a property of the **window object**.

The **console object** is accessed with:

`window.console` or just `console`

Examples

```
window.console.error("You made a mistake");

console.error("You made a mistake");
```

Console Object Methods

Method	Description
assert()	Writes an error message to the console if a assertion is false
clear()	Clears the console

count()	Logs the number of times that this particular call to count() has been called
error()	Outputs an error message to the console
group()	Creates a new inline group in the console. This indents following console messages by an additional level, until console.groupEnd() is called
groupCollapsed()	Creates a new inline group in the console. However, the new group is created collapsed. The user will need to use the disclosure button to expand it
groupEnd()	Exits the current inline group in the console
info()	Outputs an informational message to the console
log()	Outputs a message to the console
table()	Displays tabular data as a table
time()	Starts a timer (can track how long an operation takes)
timeEnd()	Stops a timer that was previously started by console.time()
trace()	Outputs a stack trace to the console
warn()	Outputs a warning message to the console

Window History

The Window History Object

The **history object** contains the URLs visited by the user (in the browser window).

The **history object** is a property of the **window object**.

The **history object** is accessed with:

`window.history` or just `history`:

Examples

```
let length = window.history.length;

let length = history.length;
```

History Object Properties and Methods

Property/Method	Description
back()	Loads the previous URL (page) in the history list
forward()	Loads the next URL (page) in the history list
go()	Loads a specific URL (page) from the history list
length	Returns the number of URLs (pages) in the history list

Window Location

The Window Location Object

The **location object** contains information about the current URL.

The **location object** is a property of the **window object**.

The **location object** is accessed with:

`window.location` or just `location`

Examples

```
let origin = window.location.origin;

let origin = location.origin;
```

Location Object Properties

Property	Description
hash	Sets or returns the anchor part (#) of a URL
host	Sets or returns the hostname and port number of a URL
hostname	Sets or returns the hostname of a URL
href	Sets or returns the entire URL

origin	Returns the protocol, hostname and port number of a URL
pathname	Sets or returns the path name of a URL
port	Sets or returns the port number of a URL
protocol	Sets or returns the protocol of a URL
search	Sets or returns the querystring part of a URL

Location Object Methods

Method	Description
assign()	Loads a new document
reload()	Reloads the current document
replace()	Replaces the current document with a new one

Window Navigator

The Window Navigator Object

The **navigator object** contains information about the browser.

The **navigator object** is a property of the **window object**.

The **navigator object** is accessed with:

`window.navigator` or just `navigator`:

Examples

```
let url = window.navigator.language;

let url = navigator.language;
```

Navigator Object Properties

Property	Description

appCodeName	Returns browser code name
appName	Returns browser name
appVersion	Returns browser version
cookieEnabled	Returns true if browser cookies are enabled
geolocation	Returns a geolocation object for the user's location
language	Returns browser language
onLine	Returns true if the browser is online
platform	Returns browser platform
product	Returns browser engine name
userAgent	Returns browser user-agent header

Navigator Object Methods

Method	Description
javaEnabled()	Returns true if the browser has Java enabled
taintEnabled()	Removed in JavaScript version 1.2 (1999).

Window Screen

The Window Screen Object

The screen object contains information about the visitor's screen.

Screen Object Properties

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)

colorDepth	Returns the bit depth of the color palette for displaying images
height	Returns the total height of the screen
pixelDepth	Returns the color resolution (in bits per pixel) of the screen
width	Returns the total width of the screen

How to create printer friendly web page using javascript

When you create a web page you usually keep in mind how it will rendered on the browser and its usability. Sometime, depending upon the content, you also know that your page might be printed by end user. User will click specially open the printable version and take its print or sometime he will directly print your webpage.

Now what are the issues here –

The web page is not printer friendly therefore the content might get distorted while printing it into a A4 size paper.
The print will contain unnecessary links and images while have no use in printable version.
The webpage might have dark background which will lead to wastage of print ink.
Some text might have light colors which may not be visible clear on a black & white printout.
If you create separate printable view then it will involve another user action which sometimes user don't do.
There is a clean solution to this problem using the media attribute in the link tag. By using you can specify on which media a particular style sheet will be applicable. Therefore you can have different style sheet for printer and browser screen and user don't have to worry about this.
Now lets start making it printer friendly. First of all create three different style sheets. One for screen, one for print and one which will contain common classes. Include this css in page and add desired media value i.e. for printer add media = print and for screen css add media = screen. Don't specify the media for the common css.

<link href="style-common.css" rel="stylesheet" type="text/css" /><link href="style-screen.css" media="screen" rel="stylesheet" type="text/css" /><link href="style-print.css" media="print" rel="stylesheet" type="text/css" />

JavaScript Forms

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    returnfalse;
  }
}
```

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page.php"onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

JavaScript Can Validate Numeric Input

JavaScript is often used to validate numeric input:

Please input a number between 1 and 10

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    returnfalse;
  }
}
```

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page.php"onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

JavaScript Can Validate Numeric Input

JavaScript is often used to validate numeric input:

Please input a number between 1 and 10

Automatic HTML Form Validation

HTML form validation can be performed automatically by the browser:

If a form field (fname) is empty, the `required` attribute prevents this form from being submitted:

HTML Form Example

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname"required>
  <input type="submit" value="Submit">
</form>
```

Automatic HTML form validation does not work in Internet Explorer 9 or earlier.

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?

- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server.

Client side validation is performed by a web browser, before input is sent to a web server.

HTML Constraint Validation

HTML5 introduced a new HTML validation concept called **constraint validation**.

HTML constraint validation is based on:

- Constraint validation **HTMLInput Attributes**
- Constraint validation **CSS Pseudo Selectors**
- Constraint validation **DOM Properties and Methods**

Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

For a full list, go to HTML Input Attributes.

Constraint Validation CSS Pseudo Selectors

Selector	Description
:disabled	Selects input elements with the "disabled" attribute specified
:invalid	Selects input elements with invalid values
:optional	Selects input elements with no "required" attribute specified
:required	Selects input elements with the "required" attribute specified
:valid	Selects input elements with valid values

JavaScript HTML DOM - Changing CSS

The HTML DOM allows JavaScript to change the style of HTML elements.

Changing HTML Style

To change the style of an HTML element, use this syntax:

document.getElementById(*id*).style.*property* = *new style*

The following example changes the style of a `<p>` element:

Example

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>

</body>
</html>
```

Using Events

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

You will learn more about events in the next chapter of this tutorial.

This example changes the style of the HTML element with `id="id1"`, when the user clicks a button:

Example

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

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

JavaScript HTML DOM

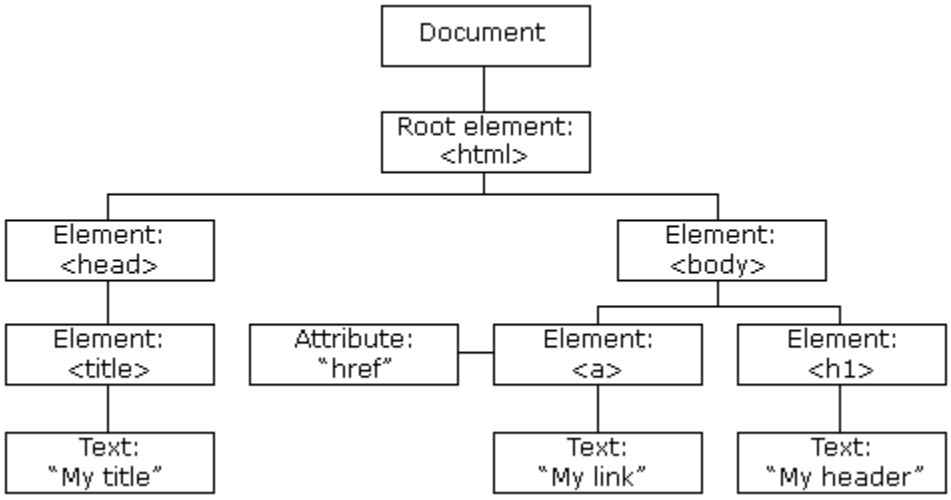
With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

What You Will Learn

In the next chapters of this tutorial you will learn:

- How to change the content of HTML elements
- How to change the style (CSS) of HTML elements
- How to react to HTML DOM events
- How to add and delete HTML elements

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`:

Example

```
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

In the example above, `getElementById` is a **method**, while `innerHTML` is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the `id` of the element.

In the example above the `getElementById` method used `id="demo"` to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the `innerHTML` property.

The `innerHTML` property is useful for getting or replacing the content of HTML elements.

The `innerHTML` property can be used to get or change any HTML element, including `<html>` and `<body>`.

JavaScript HTML DOM Document

The HTML DOM document object is the owner of all other objects in your web page.

The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

---

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

Method	Description
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

---

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

JavaScript HTML DOM Elements

his page teaches you how to find and access HTML elements in an HTML page.

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are several ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with `id="intro"`:

Example

```
const element = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in element).

If the element is not found, element will contain `null`.

Finding HTML Elements by Tag Name

This example finds all `<p>` elements:

Example

```
const element = document.getElementsByTagName("p");
```

This example finds the element with `id="main"`, and then finds all `<p>` elements inside `"main"`:

Example

```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");
```

JavaScript HTML DOM Animation

Learn to create HTML animations using JavaScript.

A Basic Web Page

To demonstrate how to create HTML animations with JavaScript, we will use a simple web page:

Example

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

<h1>My First JavaScript Animation</h1>

<div id="animation">My animation will go here</div>
```

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

Create an Animation Container

All animations should be relative to a container element.

Example

```
<div id ="container">
  <div id ="animate">My animation will go here</div>
</div>
```

Style the Elements

The container element should be created with style = "`position: relative`".

The animation element should be created with style = "`position: absolute`".

Example

```
#container {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background: red;
}
```

JavaScript HTML DOM Events

HTML DOM allows JavaScript to react to HTML events:

Mouse Over Me

Click Me

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

In this example, the content of the `<h1>` element is changed when a user clicks on it:

Example

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

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

</body>
</html>
```

In this example, a function is called from the event handler:

Example

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

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
```

```
}
</script>

</body>
</html>
```

JavaScript HTML DOM EventListener

The addEventListener() method

Example

Add an event listener that fires when a user clicks a button:

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

The `addEventListener()` method attaches an event handler to the specified element.

The `addEventListener()` method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The `addEventListener()` method makes it easier to control how the event reacts to bubbling.

When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the `removeEventListener()` method.

Syntax

element.addEventListener(*event, function, useCapture*);

The first parameter is the type of the event (like "`click`" or "`mousedown`" or any other HTML DOM Event.)

The second parameter is the function we want to call when the event occurs.

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Note that you don't use the "on" prefix for the event; use "`click`" instead of "`onclick`".

Add an Event Handler to an Element

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

You can also refer to an external "named" function:

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
```

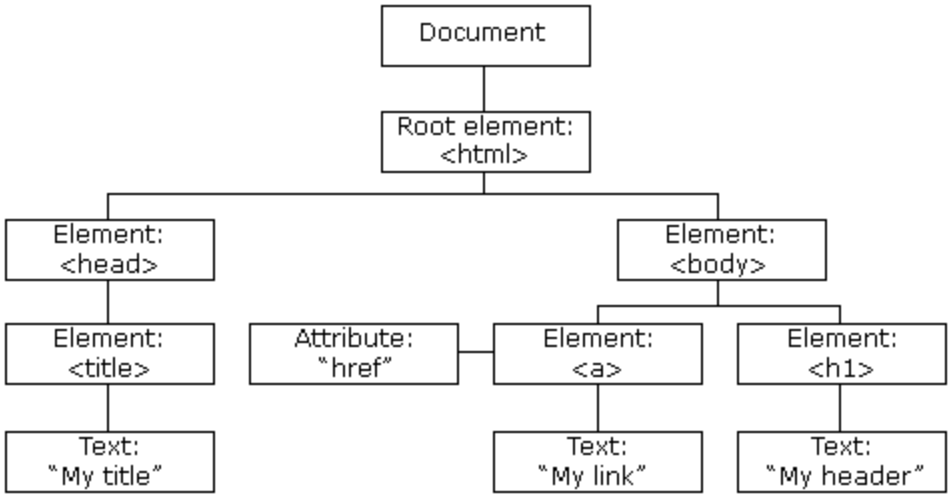
JavaScript HTML DOM Navigation

With the HTML DOM, you can navigate the node tree using node relationships.

DOM Nodes

According to the W3C HTML DOM standard, everything in an HTML document is a node:

- The entire document is a document node
- Every HTML element is an element node
- The text inside HTML elements are text nodes
- Every HTML attribute is an attribute node (deprecated)
- All comments are comment nodes



With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.

New nodes can be created, and all nodes can be modified or deleted.

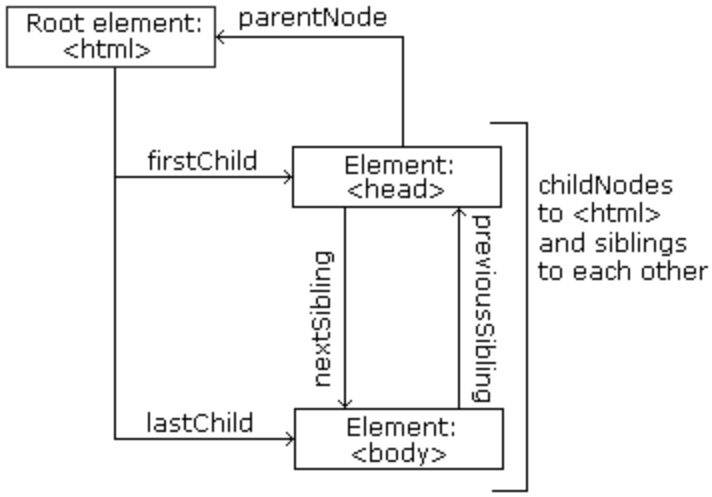
Node Relationships

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

```
<html>

 <head>
  <title>DOM Tutorial</title>
 </head>

 <body>
  <h1>DOM Lesson one</h1>
  <p>Hello world!</p>
 </body>

</html>
```



From the HTML above you can read:

- `<html>` is the root node
- `<html>` has no parents
- `<html>` is the parent of `<head>` and `<body>`
- `<head>` is the first child of `<html>`
- `<body>` is the last child of `<html>`

and:

- `<head>` has one child: `<title>`
- `<title>` has one child (a text node): "DOM Tutorial"
- `<body>` has two children: `<h1>` and `<p>`
- `<h1>` has one child: "DOM Lesson one"
- `<p>` has one child: "Hello world!"
- `<h1>` and `<p>` are siblings

Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

- `parentNode`
- `childNodes[`*`nodenumber`*`]`
- `firstChild`
- `lastChild`
- `nextSibling`
- `previousSibling`

Child Nodes and Node Values

A common error in DOM processing is to expect an element node to contain text.

Example:

```
<title id="demo">DOM Tutorial</title>
```

The element node `<title>` (in the example above) does **not** contain text.

It contains a **text node** with the value "DOM Tutorial".

The value of the text node can be accessed by the node's `innerHTML` property:

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

Accessing the innerHTML property is the same as accessing the `nodeValue` of the first child:

```
myTitle = document.getElementById("demo").firstChild.nodeValue;
```

Accessing the first child can also be done like this:

```
myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

All the (3) following examples retrieves the text of an `<h1>` element and copies it into a `<p>` element:

Example

```
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").innerHTML;
</script>

</body>
</html>
```

Example

```
<html>
<body>

<h1 id="id01">My First Page</h1>
```

```
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").firstChild.nodeValue;
</script>

</body>
</html>
```

Example

```
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02">Hello!</p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").childNodes[0].nodeValue;
</script>

</body>
</html>
```

InnerHTML

In this tutorial we use the innerHTML property to retrieve the content of an HTML element.

However, learning the other methods above is useful for understanding the tree structure and the navigation of the DOM.

DOM Root Nodes

There are two special properties that allow access to the full document:

- `document.body` - The body of the document
- `document.documentElement` - The full document

Example

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

```
<h2>JavaScript HTMLDOM</h2>
<p>Displaying document.body</p>

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

<script>
document.getElementById("demo").innerHTML = document.body.innerHTML;
</script>

</body>
</html>
```

Example

```
<html>
<body>

<h2>JavaScript HTMLDOM</h2>
<p>Displaying document.documentElement</p>

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

<script>
document.getElementById("demo").innerHTML = document.documentElement.innerHTML;
</script>

</body>
</html>
```

The nodeName Property

The `nodeName` property specifies the name of a node.

- nodeName is read-only
- nodeName of an element node is the same as the tag name
- nodeName of an attribute node is the attribute name
- nodeName of a text node is always #text
- nodeName of the document node is always #document

Example

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>
```

```
<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeName;
</script>
```

Note: `nodeName` always contains the uppercase tag name of an HTML element.

The nodeValue Property

The `nodeValue` property specifies the value of a node.

- nodeValue for element nodes is `null`
- nodeValue for text nodes is the text itself
- nodeValue for attribute nodes is the attribute value

The nodeType Property

The `nodeType` property is read only. It returns the type of a node.

Example

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeType;
</script>
```

The most important nodeType properties are:

Node	Type	Example
ELEMENT_NODE	1	<h1 class="heading">W3Schools</h1>
ATTRIBUTE_NODE	2	class = "heading" (deprecated)
TEXT_NODE	3	W3Schools
COMMENT_NODE	8	<!-- This is a comment -->

DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

Type 2 is deprecated in the HTML DOM (but works). It is not deprecated in the XML DOM.

JavaScript HTML DOM Elements (Nodes)

dding and Removing Nodes (HTML Elements)

Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

Example

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const element = document.getElementById("div1");
element.appendChild(para);
</script>
```

Example Explained

This code creates a new <p> element:

```
const para = document.createElement("p");
```

To add text to the <p> element, you must create a text node first. This code creates a text node:

```
const node = document.createTextNode("This is a new paragraph.");
```

Then you must append the text node to the `<p>` element:

```
para.appendChild(node);
```

Finally you must append the new element to an existing element.

This code finds an existing element:

```
const element = document.getElementById("div1");
```

This code appends the new element to the existing element:

```
element.appendChild(para);
```

JavaScript HTML DOM Collections

The HTMLCollection Object

The `getElementsByTagName()` method returns an `HTMLCollection` object.

An `HTMLCollection` object is an array-like list (collection) of HTML elements.

The following code selects all `<p>` elements in a document:

Example

```
const myCollection = document.getElementsByTagName("p");
```

The elements in the collection can be accessed by an index number.

To access the second <p> element you can write:

```
myCollection[1]
```

Note: The index starts at 0.

HTML HTMLCollection Length

The `length` property defines the number of elements in an `HTMLCollection`:

Example

```
myCollection.length
```

The `length` property is useful when you want to loop through the elements in a collection:

Example

Change the text color of all <p> elements:

```
const myCollection = document.getElementsByTagName("p");
for (let i = 0; i < myCollection.length; i++) {
  myCollection[i].style.color = "red";
}
```

An HTMLCollection is NOT an array!

An HTMLCollection may look like an array, but it is not.

You can loop through the list and refer to the elements with a number (just like an array).

However, you cannot use array methods like valueOf(), pop(), push(), or join() on an HTMLCollection.

JavaScript HTML DOM Node Lists

The HTML DOM NodeList Object

A `NodeList` object is a list (collection) of nodes extracted from a document.

A `NodeList` object is almost the same as an `HTMLCollection` object.

Some (older) browsers return a NodeList object instead of an HTMLCollection for methods like `getElementsByClassName()`.

All browsers return a NodeList object for the property `childNodes`.

Most browsers return a NodeList object for the method `querySelectorAll()`.

The following code selects all `<p>` nodes in a document:

Example

```
const myNodeList = document.querySelectorAll("p");
```

The elements in the NodeList can be accessed by an index number.

To access the second <p> node you can write:

```
myNodeList[1]
```

Note: The index starts at 0.

HTML DOM Node List Length

The `length` property defines the number of nodes in a node list:

Example

```
myNodelist.length
```

The `length` property is useful when you want to loop through the nodes in a node list:

Example

Change the color of all <p> elements in a node list:

```
const myNodelist = document.querySelectorAll("p");
for (let i = 0; i < myNodelist.length; i++) {
  myNodelist[i].style.color = "red";
}
```

Website Management and Authoring Tools

Overview

Throughout this course, you have learned to create websites by writing code, including HTML, CSS, and Javascript. Being able to understand websites at a code level is a valuable skill for anyone, but is especially essential if you might be doing web design or development as part of your future career. Virtually every organization uses the web to do its work, so knowledge of HTML, CSS, and Javascript are highly desirable and marketable skills for many jobs. Although it's important to understand the code behind the scenes, let's face it - Developing all your web pages by hand can be pretty time consuming. This is why there are a variety of web authoring tools, applications that enable users to create websites using a familiar, easy-to-use interface, much like a word processing program. These tools also provide a rich variety of other features for managing websites. Also, there are a growing number of tools available on-line that enable novice users to create websites simply by selecting a template and entering their content into web-based form fields. This unit will explore some of these software applications and online tools.

Ajax Tutorial

AJAX tutorial covers concepts and examples of AJAX technology for beginners and professionals.

AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like JavaScript, DOM, XML, HTML/XHTML, CSS, XMLHttpRequest etc.

AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

Where it is used?

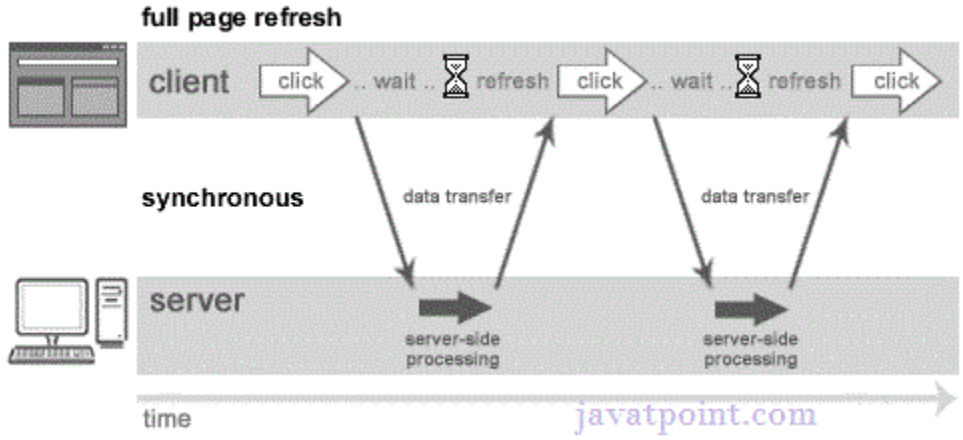
There are too many web applications running on the web that are using ajax technology like **gmail**, **facebook**,**twitter**, **google map**, **youtube** etc.

Understanding Synchronous vs Asynchronous

Before understanding AJAX, let's understand classic web application model and ajax web application model first.

Synchronous (Classic Web-Application Model)

A synchronous request blocks the client until operation completes i.e. browser is unresponsive. In such case, javascript engine of the browser is blocked.

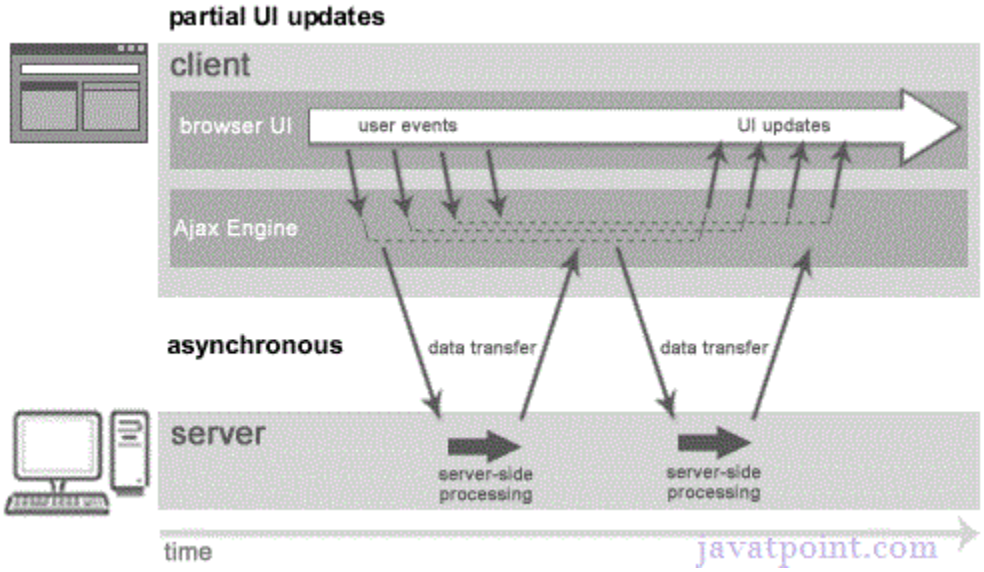


As you can see in the above image, full page is refreshed at request time and user is blocked until request completes.

Let's understand it another way.

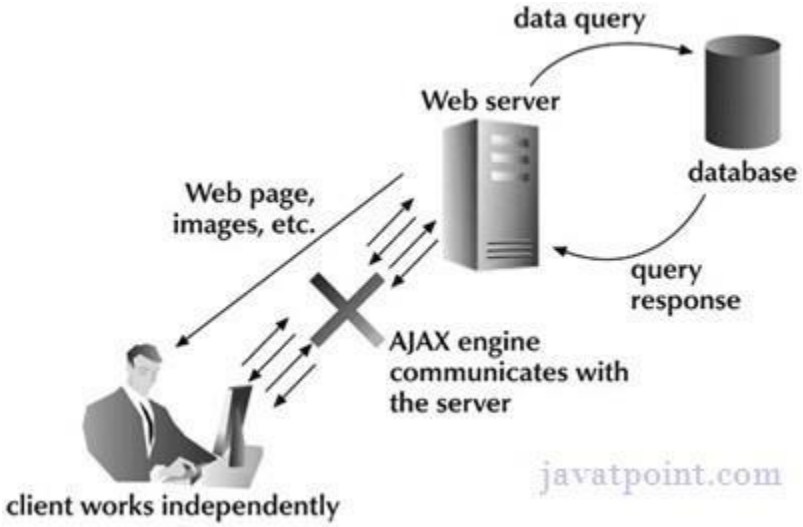
Asynchronous (AJAX Web-Application Model)

An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.



As you can see in the above image, full page is not refreshed at request time and user gets response from the ajax engine.

Let's try to understand asynchronous communication by the image given below.



AJAX Technologies

As describe earlier, ajax is not a technology but group of inter-related technologies. AJAX technologies includes:

- HTML/XHTML and CSS
- DOM
- XML or JSON
- XMLHttpRequest
- JavaScript

HTML/XHTML and CSS

These technologies are used for displaying content and style. It is mainly used for presentation.

DOM

It is used for dynamic display and interaction with data.

XML or JSON

For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.

XMLHttpRequest

For asynchronous communication between client and server. For more visit next page.

JavaScript

It is used to bring above technologies together.

Independently, it is used mainly for client-side validation.

Understanding XMLHttpRequest

1. XMLHttpRequest

An object of XMLHttpRequest is used for asynchronous communication between client and server.

It performs following operations:

1. Sends data from the client in the background
2. Receives the data from the server
3. Updates the webpage without reloading it.

Properties of XMLHttpRequest object

The common properties of XMLHttpRequest object are as follows:

Property	Description
onReadyStateChange	It is called whenever readystate attribute changes. It must not be used with synchronous requests.
readyState	represents the state of the request. It ranges from 0 to 4. **0** UNOPENED open() is not called. **1** OPENED open is called but send() is not called. **2** HEADERS_RECEIVED send() is called, and headers and status are available. **3** LOADING Downloading data; responseText holds the data. **4** DONE The operation is completed fully.
reponseText	returns response as text.
responseXML	returns response as XML

Methods of XMLHttpRequest object

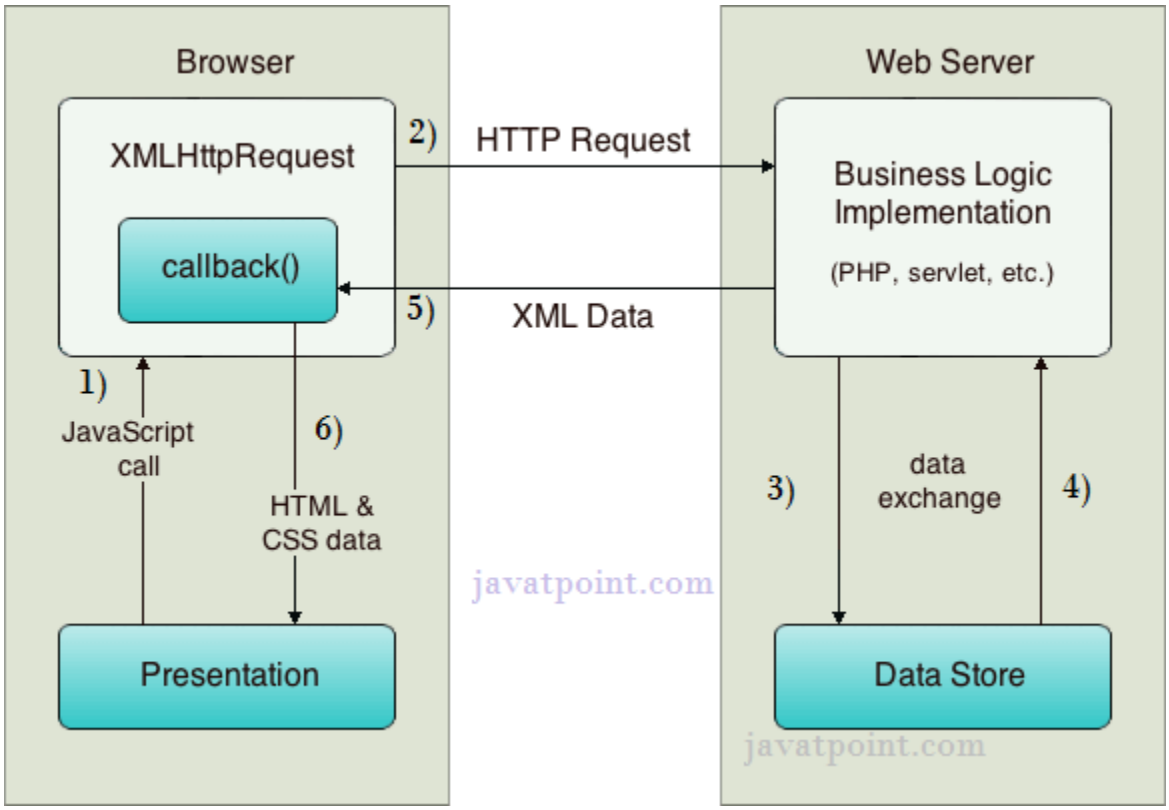
The important methods of XMLHttpRequest object are as follows:

Method	Description

void open(method, URL)	opens the request specifying get or post method and url.
void open(method, URL, async)	same as above but specifies asynchronous or not.
void open(method, URL, async, username, password)	same as above but specifies username and password.
void send()	sends get request.
void send(string)	send post request.
setRequestHeader(header,value)	it adds request headers.

How AJAX works?

AJAX communicates with the server using XMLHttpRequest object. Let's try to understand the flow of ajax or how ajax works by the image displayed below.



As you can see in the above example, XMLHttpRequest object plays a important role.

1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.

3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.

Ajax Java Example

To create ajax example, you need to use any server-side language e.g. Servlet, JSP, PHP, ASP.Net etc. Here we are using JSP for generating the server-side code.

In this example, we are simply printing the table of the given number.

Steps to create ajax example with jsp

You need to follow following steps:

1. load the org.json.jar file
2. create input page to receive any text or number
3. create server side page to process the request
4. provide entry in web.xml file

Load the org.json.jar file

download this example, we have included the org.json.jar file inside the WEB-INF/lib directory.

---

create input page to receive any text or number

In this page, we have created a form that gets input from the user. When user clicks on the showTable button, **sendInfo()** function is called. We have written all the ajax code inside this function.

We have called the **getInfo()** function whenever ready state changes. It writes the returned data in the web page dynamically by the help of **innerHTML** property.

table1.html

```
<html>
<head>
<script>
var request;
function sendInfo()
{
var v=document.vinform.t1.value;
```

```
8.  var url="index.jsp?val="+v;
9.
10. if(window.XMLHttpRequest){
11. request=new XMLHttpRequest();
12. }
13. else if(window.ActiveXObject){
14. request=new ActiveXObject("Microsoft.XMLHTTP");
15. }
16.
17. try
18. {
19. request.onreadystatechange=getInfo;
20. request.open("GET",url,true);
21. request.send();
22. }
23. catch(e)
24. {
25. alert("Unable to connect to server");
26. }
27. }
28.
29. function getInfo(){
30. if(request.readyState==4){
31. var val=request.responseText;
32. document.getElementById('amit').innerHTML=val;
33. }
34. }
35.
36. </script>
37. </head>
38. <body>
39.    <marquee><h1>This is an example of ajax</h1></marquee>
40. <form name="vinform">
41. <input type="text" name="t1">
42. <input type="button" value="ShowTable" onClick="sendInfo()">
43. </form>
44.
45. <span id="amit"> </span>
46.
47. </body>
48. </html>
```

---

create server side page to process the request

In this jsp page, we printing the table of given number.

index.jsp

```
<%
int n=Integer.parseInt(request.getParameter("val"));

for(int i=1;i<=10;i++)
out.print(i*n+"<br>");

%>
```

---

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>table1.html</welcome-file>
        </welcome-file-list>
    </web-app>
```

Ajax Java Example with Database

In this example, we are interacting with the database. You don't have to make any extra effort. Only write the database logic in you server side page.

In this example, we have written the server side code inside the index.jsp file.

Steps to create ajax example with database through jsp

You need to follow following steps:

1. load the org.json.jar file
2. create input page to receive any text or number
3. create server side page to process the request

create input page to receive any text or number

In this page, we have created a form that gets input from the user. When user press any key **sendInfo()** function is called. We have written all the ajax code inside this function.

We have called the **getInfo()** function whenever ready state changes. It writes the returned data in the web page dynamically by the help of **innerHTML** property.

table1.html

```
1.  <html>
2.  <head>
3.  <script>
4.  var request;
5.  function sendInfo()
6.  {
7.  var v=document.vinform.t1.value;
8.  var url="index.jsp?val="+v;
9.
10. if(window.XMLHttpRequest){
11. request=new XMLHttpRequest();
12. }
13. else if(window.ActiveXObject){
14. request=new ActiveXObject("Microsoft.XMLHTTP");
15. }
16.
17. try{
18. request.onreadystatechange=getInfo;
19. request.open("GET",url,true);
20. request.send();
21. }catch(e){alert("Unable to connect to server");}
22. }
23.
24. function getInfo(){
25. if(request.readyState==4){
26. var val=request.responseText;
27. document.getElementById('amit').innerHTML=val;
28. }
29. }
30.
31. </script>
32. </head>
33. <body>
34.    <marquee><h1>This is an example of ajax</h1></marquee>
35. <form name="vinform">
36. Enter id:<input type="text" name="t1" onkeyup="sendInfo()">
37. </form>
```

```
38.
39. <span id="amit"> </span>
40.
41. </body>
42. </html>
```

---

create server side page to process the request

In this jsp page, we printing the id and name of the employee for the given id.

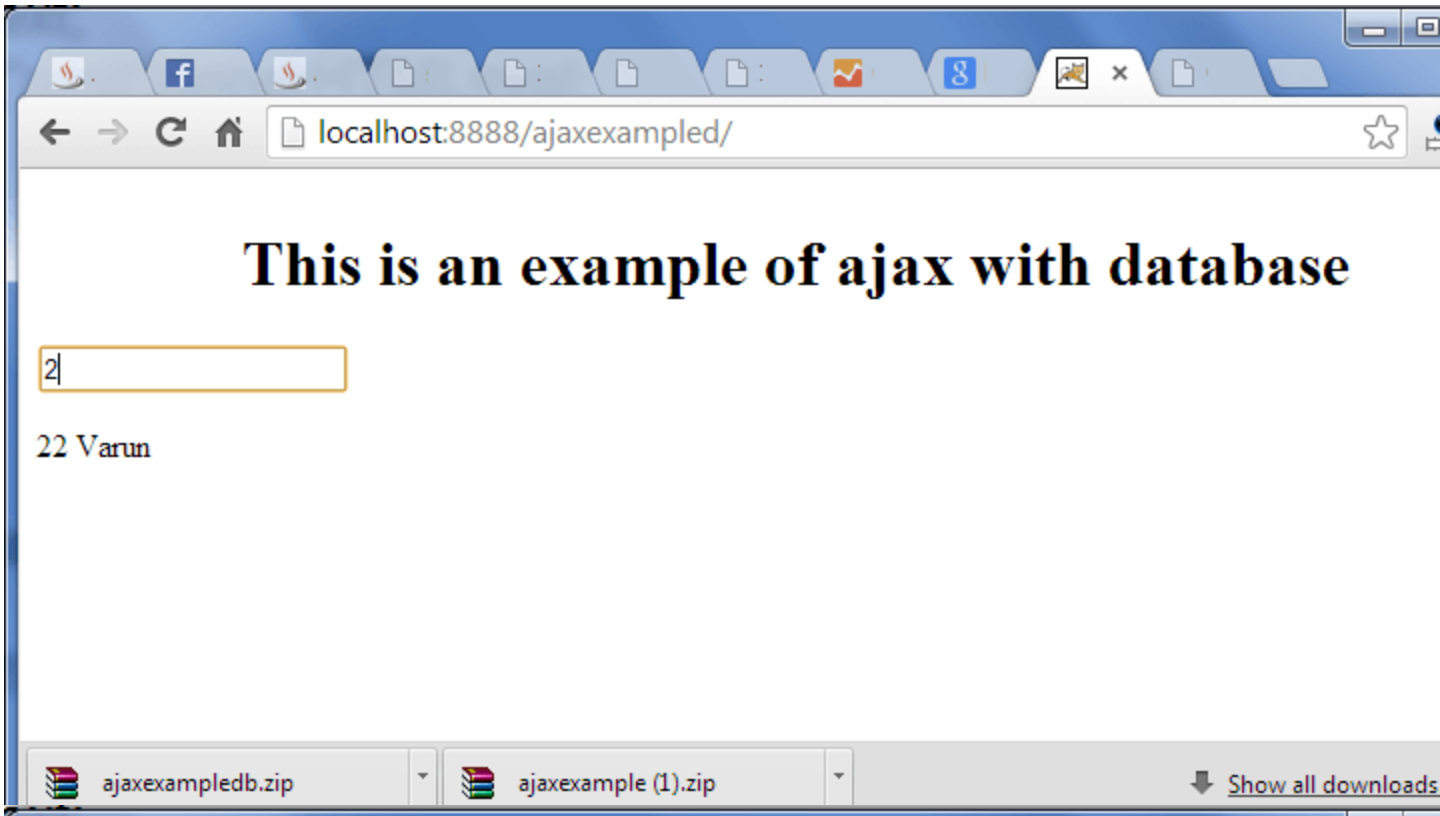
index.jsp

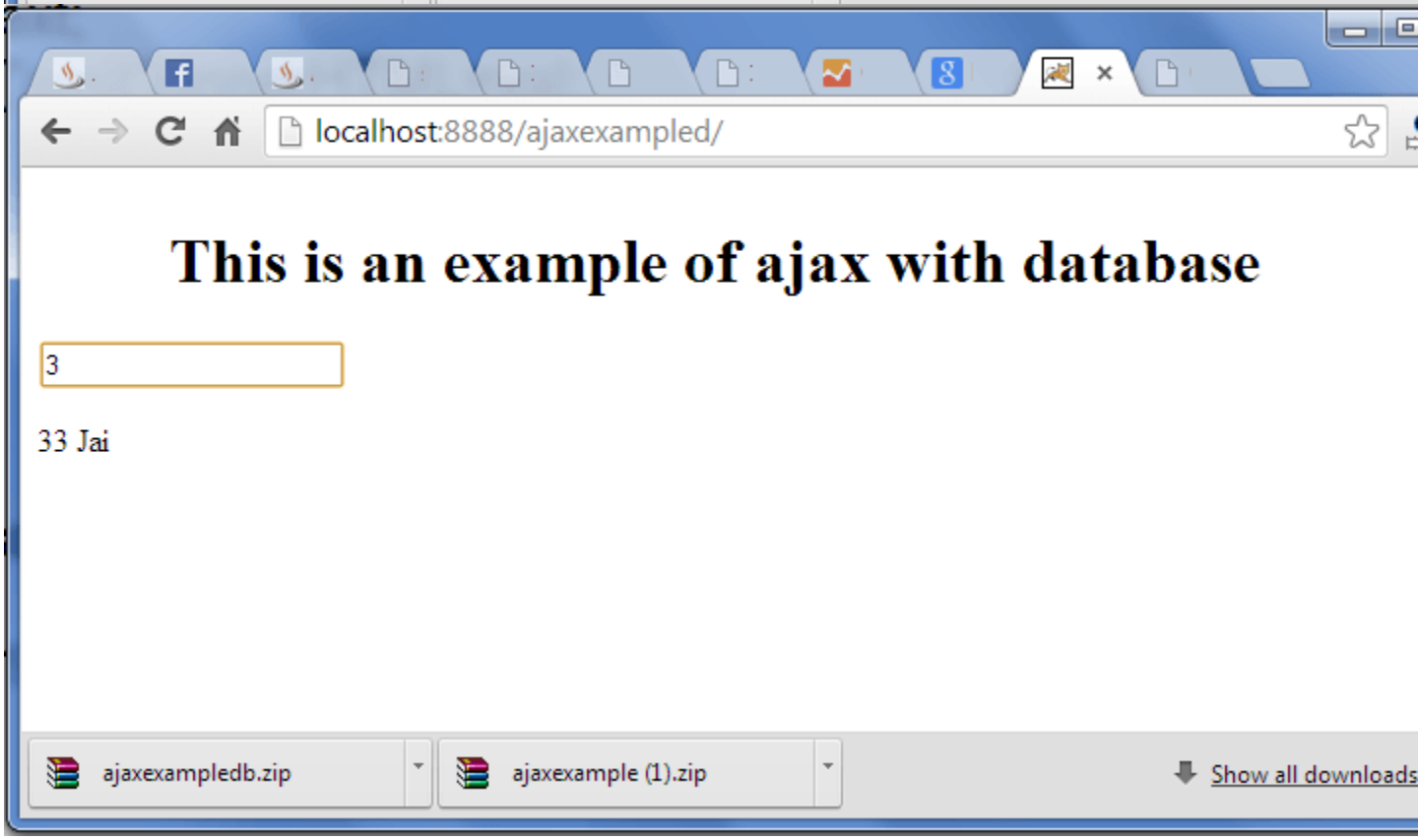
```
1.  <%@ page import="java.sql.*"%>
2.
3.  <%
4.  String s=request.getParameter("val");
5.  if(s==null || s.trim().equals("")){
6.  out.print("Please enter id");
7.  }else{
8.  int id=Integer.parseInt(s);
9.  out.print(id);
10. try{
11. Class.forName("com.mysql.jdbc.Driver");
12. Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mdb","root","root");
13. PreparedStatement ps=con.prepareStatement("select * from emp where id=?");
14. ps.setInt(1,id);
15. ResultSet rs=ps.executeQuery();
16. while(rs.next()){
17. out.print(rs.getInt(1)+" "+rs.getString(2));
18. }
19. con.close();
20. }catch(Exception e){e.printStackTrace();}
21. }
22. %>
```

---

download this ajax example

Output


localhost:8888/ajaxexampled/
This is an example of ajax with database
2
22 Varun
ajaxexampledb.zip
ajaxexample (1).zip
Show all downloads


localhost:8888/ajaxexampled/
This is an example of ajax with database
3
33 Jai
ajaxexampledb.zip
ajaxexample (1).zip
Show all downloads

AJAX JSON Example

We can get JSON data by AJAX code. AJAX provides facility to get response asynchronously. It doesn't reload the page and saves bandwidth.

AJAX JSON Example

Let's see a simple example of getting JSON data using AJAX code.

```
<html>
<head>
<meta content="text/html; charset=utf-8">
<title>AJAX JSON by Javatpoint</title>
<script type="application/javascript">
function load()
{
  var url = "http://date.jsontest.com/";//use any url that have json data
  var request;

  if(window.XMLHttpRequest){
   request=new XMLHttpRequest();//for Chrome, mozilla etc
  }
  else if(window.ActiveXObject){
   request=new ActiveXObject("Microsoft.XMLHTTP");//for IE only
  }
  request.onreadystatechange  = function(){
    if (request.readyState == 4  )
    {
      var jsonObj = JSON.parse(request.responseText);//JSON.parse() returns JSON object
      document.getElementById("date").innerHTML =  jsonObj.date;
      document.getElementById("time").innerHTML = jsonObj.time;
    }
  }
  request.open("GET", url, true);
  request.send();
}
</script>
</head>
<body>

Date: <span id="date"></span><br/>
Time: <span id="time"></span><br/>

<button type="button" onclick="load()">Load Information</button>
</body>
</html>
```

Output:

```
Date:
Time:
```

PHP Introduction

PHP code is executed on the server.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- CSS
- JavaScript

If you want to study these subjects first, find the tutorials on our Home page.

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

PHP is an amazing and popular language!

It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)!
It is deep enough to run large social networks!
It is also easy enough to be a beginner's first server side language!

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code is executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension "`.php`"

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server

- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images or PDF files. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

---

What's new in PHP 7

- PHP 7 is much faster than the previous popular stable release (PHP 5.6)
- PHP 7 has improved Error Handling
- PHP 7 supports stricter Type Declarations for function arguments
- PHP 7 supports new operators (like the spaceship operator: <=>)

PHP Installation

What Do I Need?

To start using PHP, you can:

- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL

Use a Web Host With PHP Support

If your server has activated support for PHP you do not need to do anything.

Just create some `.php` files, place them in your web directory, and the server will automatically parse them for you.

You do not need to compile anything or install any extra tools.

Because PHP is free, most web hosts offer PHP support.

Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

The official PHP website (PHP.net) has installation instructions for PHP: http://php.net/manual/en/install.php

---

PHP Online Compiler / Editor

With w3schools' online PHP compiler, you can edit PHP code, and view the result in your browser.

```
<?php
$txt = "PHP";
echo"I love $txt!";
?>
```

```
I love PHP!
```

PHP Syntax

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Basic PHP Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with `<?php` and ends with `?>`:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".`php`".

A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "`echo`" to output the text "Hello World!" on a web page:

Example[**Get your own PHP Server**]

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

<h1>My first PHP page</h1>

<?php
echo"Hello World!";
?>

</body>
</html>
```

Note: PHP statements end with a semicolon (`;`).

PHP Comments

Comments in PHP

A comment in PHP code is a line that is not executed as a part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand your code
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports several ways of commenting:

ExampleGet your own PHP Server

Syntax for single-line comments:

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

<?php
// This is a single-line comment

# This is also a single-line comment
?>

</body>
</html>
```

ExampleGet your own PHP Server

Syntax for multiple-line comments:

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

<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
?>

</body>
</html>
```

ExampleGet your own PHP Server

Using comments to leave out parts of the code:

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

<?php
// You can also use comments to leave out parts of a code line
$x = 5/* + 15 */ + 5;
echo $x;
?>

</body>
</html>
```

PHP Variables

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the `$` sign, followed by the name of the variable:

Example[Get your own PHP Server]

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

After the execution of the statements above, the variable `$txt` will hold the value `Hello world!`, the variable `$x` will hold the value `5`, and the variable `$y` will hold the value `10.5`.

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Think of variables as containers for storing data.

PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

- A variable starts with the `$` sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (`$age` and `$AGE` are two different variables)

Remember that PHP variable names are case-sensitive!

Output Variables

The PHP `echo` statement is often used to output data to the screen.

The following example will show how to output text and a variable:

Example **Get your own PHP Server**

```
<?php
$txt = "W3Schools.com";
echo"I love $txt!";
?>
```

The following example will produce the same output as the example above:

Example **Get your own PHP Server**

```
<?php
$txt = "W3Schools.com";
echo"I love " . $txt . "!";
?>
```

The following example will output the sum of two variables:

Example **Get your own PHP Server**

```
<?php
$x = 5;
```

```
$y = 4;
echo $x + $y;
?>
```

Note: You will learn more about the `echo` statement and how to output data to the screen in the next chapter.

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives an option to specify the data type expected when declaring a function, and by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

You will learn more about `strict` and `non-strict` requirements, and data type declarations in the PHP Functions chapter.

PHP echo and print Statements

With PHP, there are two basic ways to get output: `echo` and `print`.

In this tutorial we use `echo` or `print` in almost every example. So, this chapter contains a little more info about those two output statements.

PHP echo and print Statements

`echo` and `print` are more or less the same. They are both used to output data to the screen.

The differences are small: `echo` has no return value while `print` has a return value of 1 so it can be used in expressions. `echo` can take multiple parameters (although such usage is rare) while `print` can take one argument. `echo` is marginally faster than `print`.

The PHP echo Statement

The `echo` statement can be used with or without parentheses: `echo` or `echo()`.

Display Text

The following example shows how to output text with the `echo` command (notice that the text can contain HTML markup):

ExampleGet your own PHP Server

```
<?php
echo"<h2>PHP is Fun!</h2>";
echo"Hello world!<br>";
echo"I'm about to learn PHP!<br>";
echo"This ", "string ", "was ", "made ", "with multiple parameters.";
?>
```

Display Variables

The following example shows how to output text and variables with the `echo` statement:

ExampleGet your own PHP Server

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo"<h2>" . $txt1 . "</h2>";
echo"Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>
```

PHP Data Types

PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object

- NULL
- Resource

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

Example[**Get your own PHP Server**]

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo"<br>";
echo $y;
?>
```

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

Example[**Get your own PHP Server**]

```
<?php
$x = 5985;
var_dump($x);
?>
```

PHP Strings

A string is a sequence of characters, like "Hello world!".

PHP String Functions

In this chapter we will look at some commonly used functions to manipulate strings.

strlen() - Return the Length of a String

The PHP `strlen()` function returns the length of a string.

ExampleGet your own PHP Server

Return the length of the string "Hello world!":

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

str_word_count() - Count Words in a String

The PHP `str_word_count()` function counts the number of words in a string.

ExampleGet your own PHP Server

Count the number of word in the string "Hello world!":

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

PHP Numbers

In this chapter we will look in depth into Integers, Floats, and Number Strings.

PHP Numbers

One thing to notice about PHP is that it provides automatic data type conversion.

So, if you assign an integer value to a variable, the type of that variable will automatically be an integer. Then, if you assign a string to the same variable, the type will change to a string.

This automatic conversion can sometimes break your code.

PHP Integers

2, 256, -256, 10358, -179567 are all integers.

An integer is a number without any decimal part.

An integer data type is a non-decimal number between -2147483648 and 2147483647 in 32 bit systems, and between -9223372036854775808 and 9223372036854775807 in 64 bit systems. A value greater (or lower) than this, will be stored as float, because it exceeds the limit of an integer.

Note: Another important thing to know is that even if 4 * 2.5 is 10, the result is stored as float, because one of the operands is a float (2.5).

Here are some rules for integers:

- An integer must have at least one digit
- An integer must NOT have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

PHP has the following predefined constants for integers:

- PHP_INT_MAX - The largest integer supported
- PHP_INT_MIN - The smallest integer supported
- PHP_INT_SIZE - The size of an integer in bytes

PHP has the following functions to check if the type of a variable is integer:

- is_int()
- is_integer() - alias of is_int()
- is_long() - alias of is_int()

ExampleGet your own PHP Server

Check if the type of a variable is integer:

```
<?php
$x = 5985;
var_dump(is_int($x));

$x = 59.85;
var_dump(is_int($x));
?>
```

PHP Floats

A float is a number with a decimal point or a number in exponential form.

2.0, 256.4, 10.358, 7.64E+5, 5.56E-5 are all floats.

The float data type can commonly store a value up to 1.7976931348623E+308 (platform dependent), and have a maximum precision of 14 digits.

PHP has the following predefined constants for floats (from PHP 7.2):

- PHP_FLOAT_MAX - The largest representable floating point number
- PHP_FLOAT_MIN - The smallest representable positive floating point number
- PHP_FLOAT_MAX - The smallest representable negative floating point number
- PHP_FLOAT_DIG - The number of decimal digits that can be rounded into a float and back without precision loss
- PHP_FLOAT_EPSILON - The smallest representable positive number x, so that x + 1.0 != 1.0

PHP has the following functions to check if the type of a variable is float:

- is_float()
- is_double() - alias of is_float()

ExampleGet your own PHP Server

Check if the type of a variable is float:

```
<?php
$x = 10.365;
```

```
var_dump(is_float($x));
?>
```

PHP Infinity

A numeric value that is larger than PHP_FLOAT_MAX is considered infinite.

PHP has the following functions to check if a numeric value is finite or infinite:

- is_finite()
- is_infinite()

However, the PHP var_dump() function returns the data type and value:

Example **Get your own PHP Server**

Check if a numeric value is finite or infinite:

```
<?php
$x = 1.9e411;
var_dump($x);
?>
```

PHP NaN

NaN stands for Not a Number.

NaN is used for impossible mathematical operations.

PHP has the following functions to check if a value is not a number:

- is_nan()

However, the PHP var_dump() function returns the data type and value:

Example **Get your own PHP Server**

Invalid calculation will return a NaN value:

```
<?php
$x = acos(8);
```

```
var_dump($x);
?>
```

PHP Numerical Strings

The PHP is_numeric() function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

Example[Get your own PHP Server]

Check if the variable is numeric:

```
<?php
$x = 5985;
var_dump(is_numeric($x));

$x = "5985";
var_dump(is_numeric($x));

$x = "59.85" + 100;
var_dump(is_numeric($x));

$x = "Hello";
var_dump(is_numeric($x));
?>
```

Note: From PHP 7.0: The is_numeric() function will return FALSE for numeric strings in hexadecimal form (e.g. 0xf4c3b00c), as they are no longer considered as numeric strings.

PHP Casting Strings and Floats to Integers

Sometimes you need to cast a numerical value into another data type.

The (int), (integer), or intval() function are often used to convert a value to an integer.

Example[Get your own PHP Server]

Cast float and string to integer:

```php
<?php
// Cast float to int
$x = 23465.768;
$int_cast = (int)$x;
echo $int_cast;

echo"<br>";

// Cast string to int
$x = "23465.768";
$int_cast = (int)$x;
echo $int_cast;
?>
```

PHP Math

PHP has a set of math functions that allows you to perform mathematical tasks on numbers.

PHP pi() Function

The `pi()` function returns the value of PI:

Example[Get your own PHP Server]

```php
<?php
echo(pi()); // returns 3.1415926535898
?>
```

PHP min() and max() Functions

The `min()` and `max()` functions can be used to find the lowest or highest value in a list of arguments:

Example[Get your own PHP Server]

```php
<?php
echo(min(0, 150, 30, 20, -8, -200));  // returns -200
echo(max(0, 150, 30, 20, -8, -200));  // returns 150
?>
```

PHP abs() Function

The `abs()` function returns the absolute (positive) value of a number:

Example[Get your own PHP Server]

```
<?php
echo(abs(-6.7));  // returns 6.7
?>
```

PHP sqrt() Function

The `sqrt()` function returns the square root of a number:

Example[Get your own PHP Server]

```
<?php
echo(sqrt(64));  // returns 8
?>
```

PHP round() Function

The `round()` function rounds a floating-point number to its nearest integer:

Example[Get your own PHP Server]

```
<?php
echo(round(0.60));  // returns 1
echo(round(0.49));  // returns 0
?>
```

Random Numbers

The `rand()` function generates a random number:

ExampleGet your own PHP Server

```
<?php
echo(rand());
?>
```

To get more control over the random number, you can add the optional *min* and *max* parameters to specify the lowest integer and the highest integer to be returned.

For example, if you want a random integer between 10 and 100 (inclusive), use `rand(10, 100)`:

ExampleGet your own PHP Server

```
<?php
echo(rand(10, 100));
?>
```

Complete PHP Math Reference

For a complete reference of all math functions, go to our complete PHP Math Reference.

The PHP math reference contains description and example of use, for each function.

PHP Constants

Constants are like variables except that once they are defined they cannot be changed or undefined.

PHP Constants

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.

A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

Create a PHP Constant

To create a constant, use the `define()` function.

Syntax

define(*name*, *value*, *case-insensitive*)

Parameters:

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant
- *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

ExampleGet your own PHP Server

Create a constant with a **case-sensitive** name:

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

ExampleGet your own PHP Server

Create a constant with a **case-insensitive** name:

```
<?php
define("GREETING", "Welcome to W3Schools.com!", true);
echo greeting;
?>
```


PHP Constant Arrays

In PHP7, you can create an Array constant using the `define()` function.

ExampleGet your own PHP Server

Create an Array constant:

```php
<?php
define("cars", [
  "Alfa Romeo",
  "BMW",
  "Toyota"
]);
echo cars[0];
?>
```

Constants are Global

Constants are automatically global and can be used across the entire script.

Example[**Get your own PHP Server**]

This example uses a constant inside a function, even if it is defined outside the function:

```php
<?php
define("GREETING", "Welcome to W3Schools.com!");

function myTest() {
  echo GREETING;
}

myTest();
?>
```

PHP Operators

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators

- Logical operators
- String operators
- Array operators
- Conditional assignment operators

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result	Show it
+	Addition	$x + $y	Sum of $x and $y	
-	Subtraction	$x - $y	Difference of $x and $y	
*	Multiplication	$x * $y	Product of $x and $y	
/	Division	$x / $y	Quotient of $x and $y	
%	Modulus	$x % $y	Remainder of $x divided by $y	
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power	

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description	Show it
x = y	x = y	The left operand gets set to the value of the expression on the right	
x += y	x = x + y	Addition	
x -= y	x = x - y	Subtraction	

x *= y x = x * y Multiplication

x /= y x = x / y Division

x %= y x = x % y Modulus

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result	Show it
==	Equal	$x == $y	Returns true if $x is equal to $y	
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type	
!=	Not equal	$x != $y	Returns true if $x is not equal to $y	
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y	
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type	
>	Greater than	$x > $y	Returns true if $x is greater than $y	
<	Less than	$x < $y	Returns true if $x is less than $y	
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y	
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y	
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.	

PHP Increment / Decrement Operators

The PHP increment operators are used to increment a variable's value.

The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description	Show it
++$x	Pre-increment	Increments $x by one, then returns $x	
$x++	Post-increment	Returns $x, then increments $x by one	
--$x	Pre-decrement	Decrements $x by one, then returns $x	
$x--	Post-decrement	Returns $x, then decrements $x by one	

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result	Show it
and	And	$x and $y	True if both $x and $y are true	
or	Or	$x or $y	True if either $x or $y is true	
xor	Xor	$x xor $y	True if either $x or $y is true, but not both	
&&	And	$x && $y	True if both $x and $y are true	
||	Or	$x || $y	True if either $x or $y is true	
!	Not	!$x	True if $x is not true	

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result	Show it
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2	
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1	

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result	Show it
+	Union	$x + $y	Union of $x and $y	
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs	
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types	
!=	Inequality	$x != $y	Returns true if $x is not equal to $y	
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y	
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y	

PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions:

Operator	Name	Example	Result	Show it
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE	
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*.	

Introduced in PHP 7

PHP Exercises

Test Yourself With Exercises

Exercise:

Multiply `10` with `5`, and output the result.

```
echo 10 [ ] 5;
```

PHP if...else...elseif Statements

Conditional statements are used to perform different actions based on different conditions.

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- `if` statement - executes some code if one condition is true
- `if...else` statement - executes some code if a condition is true and another code if that condition is false
- `if...elseif...else` statement - executes different codes for more than two conditions
- `switch` statement - selects one of many blocks of code to be executed

---

PHP - The if Statement

The `if` statement executes some code if one condition is true.

Syntax

```
if (condition) {
  code to be executed if condition is true;
}
```

ExampleGet your own PHP Server

Output "Have a good day!" if the current time (HOUR) is less than 20:

```
<?php
$t = date("H");

if ($t <"20") {
  echo"Have a good day!";
}
?>
```

PHP - The if...else Statement

The `if...else` statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
if (condition) {
  code to be executed if condition is true;
} else {
  code to be executed if condition is false;
}
```

ExampleGet your own PHP Server

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
<?php
$t = date("H");

if ($t <"20") {
  echo"Have a good day!";
} else {
  echo"Have a good night!";
}
?>
```

PHP - The if...elseif...else Statement

The `if...elseif...else` statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
  code to be executed if this condition is true;
} elseif (condition) {
  code to be executed if first condition is false and this condition is true;
} else {
  code to be executed if all conditions are false;
}
```

ExampleGet your own PHP Server

Output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

```
<?php
$t = date("H");

if ($t <"10") {
  echo"Have a good morning!";
} elseif ($t <"20") {
  echo"Have a good day!";
} else {
  echo"Have a good night!";
}
?>
```

PHP - The switch Statement

The `switch` statement will be explained in the next chapter.

PHP Exercises

Test Yourself With Exercises

Exercise:

Output "Hello World" if `$a` is greater than `$b`.

```
$a = 50;
$b = 10;
[          ][          ]>[          ] {
  echo "Hello World";
}
```

PHP switch Statement

The `switch` statement is used to perform different actions based on different conditions.

The PHP switch Statement

Use the `switch` statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
 case label1:
   code to be executed if n=label1;
   break;
 case label2:
   code to be executed if n=label2;
   break;
 case label3:
   code to be executed if n=label3;
   break;
   ...
 default:
   code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use `break` to prevent the code from running into the next case automatically. The `default` statement is used if no match is found.

Example[**Get your own PHP Server**]

```
<?php
$favcolor = "red";

switch ($favcolor){
 case"red":
   echo"Your favorite color is red!";
   break;
 case"blue":
   echo"Your favorite color is blue!";
   break;
 case"green":
   echo"Your favorite color is green!";
  break;
 default:
   echo"Your favorite color is neither red, blue, nor green!";
}
?>
```

PHP Exercises

Test Yourself With Exercises

Exercise:

Create a `switch` statement that will output "Hello" if `$color` is "red", and "welcome" if `$color` is "green".

```
        ($color) {
        "red":
    echo "Hello";
    break;
        "green":
    echo "Welcome";
```

```
    break;
}
```

PHP Loops

In the following chapters you will learn how to repeat code by using loops in PHP.

PHP Loops

Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

- `while` - loops through a block of code as long as the specified condition is true
- `do...while` - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- `for` - loops through a block of code a specified number of times
- `foreach` - loops through a block of code for each element in an array

The following chapters will explain and give examples of each loop type.

PHP while Loop

The `while` loop - Loops through a block of code as long as the specified condition is true.

The PHP while Loop

The `while` loop executes a block of code as long as the specified condition is true.

Syntax

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

Examples

The example below displays the numbers from 1 to 5:

Example[Get your own PHP Server]

```
<?php
$x = 1;

while($x <= 5) {
  echo"The number is: $x <br>";
  $x++;
}
?>
```

Example Explained

- $x = 1; - Initialize the loop counter ($x), and set the start value to 1
- $x <= 5 - Continue the loop as long as $x is less than or equal to 5
- $x++; - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

Example[Get your own PHP Server]

```
<?php
$x = 0;

while($x <= 100) {
  echo"The number is: $x <br>";
  $x+=10;
}
?>
```

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 100 - Continue the loop as long as $x is less than or equal to 100
- $x+=10; - Increase the loop counter value by 10 for each iteration

PHP do while Loop

The `do...while` loop - Loops through a block of code once, and then repeats the loop as long as the specified condition is true.

The PHP do...while Loop

The `do...while` loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

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

Examples

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example **Get your own PHP Server**

```
<?php
$x = 1;

do {
  echo"The number is: $x <br>";
  $x++;
} while ($x <= 5);
?>
```

Note: In a `do...while` loop the condition is tested AFTER executing the statements within the loop. This means that the `do...while` loop will execute its statements at least once, even if the condition is false. See example below.

This example sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

Example **Get your own PHP Server**

```
<?php
$x = 6;

do {
  echo"The number is: $x <br>";
  $x++;
} while ($x <= 5);
?>
```

PHP for Loop

The `for` loop - Loops through a block of code a specified number of times.

The PHP for Loop

The `for` loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
  code to be executed for each iteration;
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

Examples

The example below displays the numbers from 0 to 10:

Example[Get your own PHP Server]

```
<?php
for ($x = 0; $x <= 10; $x++){
  echo"The number is: $x <br>";
}
?>
```

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 10; - Continue the loop as long as $x is less than or equal to 10
- $x++ - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

ExampleGet your own PHP Server

```
<?php
for ($x = 0; $x <= 100; $x+=10){
  echo"The number is: $x <br>";
}
?>
```

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 100; - Continue the loop as long as $x is less than or equal to 100
- $x+=10 - Increase the loop counter value by 10 for each iteration

PHP foreach Loop

The `foreach` loop - Loops through a block of code for each element in an array.

The PHP foreach Loop

The `foreach` loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as$value) {
  code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

Examples

The following example will output the values of the given array ($colors):

ExampleGet your own PHP Server

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
  echo"$value <br>";
```

```
}
?>
```

The following example will output both the keys and the values of the given array ($age):

Example[Get your own PHP Server]

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $val) {
  echo"$x = $val<br>";
}
?>
```

PHP Break and Continue

PHP Break

You have already seen the `break` statement used in an earlier chapter of this tutorial. It was used to "jump out" of a `switch` statement.

The `break` statement can also be used to jump out of a loop.

This example jumps out of the loop when **x** is equal to **4**:

Example[Get your own PHP Server]

```
<?php
for ($x = 0; $x <10; $x++) {
  if ($x == 4) {
    break;
  }
  echo"The number is: $x <br>";
}
?>
```

PHP Continue

The `continue` statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of **4**:

Example[**Get your own PHP Server**]

```
<?php
for ($x = 0; $x <10; $x++) {
 if ($x == 4) {
  continue;
 }
 echo"The number is: $x <br>";
}
?>
```

PHP Functions

The real power of PHP comes from its functions.

PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.

PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

Please check out our PHP reference for a complete overview of the PHP built-in functions.

PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user-defined function declaration starts with the word `function`:

Syntax

```
function functionName() {
  code to be executed;
}
```

Note: A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

Tip: Give the function a name that reflects what the function does!

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code, and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name followed by brackets ():

Example **Get your own PHP Server**

```
<?php
function writeMsg() {
  echo"Hello world!";
}

writeMsg(); // call the function
?>
```

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example **Get your own PHP Server**

```
<?php
function familyName($fname) {
  echo"$fname Refsnes.<br>";
}
```

```
familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

The following example has a function with two arguments ($fname and $year):

Example[Get your own PHP Server]

```
<?php
function familyName($fname, $year) {
  echo"$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and by adding the `strict` declaration, it will throw a "Fatal Error" if the data type mismatches.

In the following example we try to send both a number and a string to the function without using `strict`:

Example[Get your own PHP Server]

```
<?php
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is NOT enabled "5 days" is changed to int(5), and it will return 10
?>
```

To specify `strict` we need to set `declare(strict_types=1);`. This must be on the very first line of the PHP file.

In the following example we try to send both a number and a string to the function, but here we have added the `strict` declaration:

Example[Get your own PHP Server]

```
<?phpdeclare(strict_types=1); // strict requirement

function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an integer, an error will be thrown
?>
```

The `strict` declaration forces things to be used in the intended way.

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example[Get your own PHP Server]

```
<?phpdeclare(strict_types=1); // strict requirement
function setHeight(int $minheight = 50) {
  echo"The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

PHP Functions - Returning values

To let a function return a value, use the `return` statement:

Example[Get your own PHP Server]

```
<?phpdeclare(strict_types=1); // strict requirement
function sum(int $x, int $y) {
 $z = $x + $y;
 return $z;
}

echo"5 + 10 = " . sum(5, 10) . "<br>";
echo"7 + 13 = " . sum(7, 13) . "<br>";
echo"2 + 4 = " . sum(2, 4);
?>
```

PHP Return Type Declarations

PHP 7 also supports Type Declarations for the `return` statement. Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

To declare a type for the function return, add a colon (:) and the type right before the opening curly ({)bracket when declaring the function.

In the following example we specify the return type for the function:

Example[Get your own PHP Server]

```
<?phpdeclare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
 return $a + $b;
}
echo addNumbers(1.2, 5.2);
?>
```

You can specify a different return type, than the argument types, but make sure the return is the correct type:

Example[Get your own PHP Server]

```
<?phpdeclare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : int {
 return (int)($a + $b);
}
echo addNumbers(1.2, 5.2);
?>
```

Passing Arguments by Reference

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed.

When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the `&` operator is used:

Example[Get your own PHP Server]

Use a pass-by-reference argument to update a variable:

```
<?php
function add_five(&$value) {
 $value += 5;
}

$num = 2;
add_five($num);
echo $num;
?>
```

PHP Exercises

Test Yourself With Exercises

Exercise:

Create a function named `myFunction`.

```
        {
  echo "Hello World!";
}
```

PHP Arrays

An array stores multiple values in one single variable:

Example[Get your own PHP Server]

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo"I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the `array()` function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

---

Get The Length of an Array - The count() Function

The `count()` function is used to return the length (the number of elements) of an array:

Example[**Get your own PHP Server**]

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises

Exercise:

Use the correct function to output the number of items in an array.

```
$fruits = array("Apple", "Banana", "Orange");
echo ____;
```

PHP Indexed Arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example[**Get your own PHP Server**]

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo"I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a `for` loop, like this:

Example[**Get your own PHP Server**]

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
  echo $cars[$x];
  echo"<br>";
}
?>
```

Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises

Exercise:

Output the second item in the $fruits array.

```
$fruits = array("Apple", "Banana", "Orange");
echo ;
```

PHP Associative Arrays

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example[Get your own PHP Server]

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo"Peter is " . $age['Peter'] . " years old.";
?>
```

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a `foreach` loop, like this:

ExampleGet your own PHP Server

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
 echo"Key=" . $x . ", Value=" . $x_value;
 echo"<br>";
}
?>
```

Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises

Exercise:

Create an associative array containing the age of Peter, Ben and Joe.

```
$age = array("Peter"[ ]"35", "Ben"[ ]"37", "Joe"[ ]"43");
```

PHP Multidimensional Arrays

In the previous pages, we have described arrays that are a single list of key/value pairs.

However, sometimes you want to store values with more than one key. For this, we have multidimensional arrays.

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

---

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

ExampleGet your own PHP Server

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

We can also put a `for` loop inside another `for` loop to get the elements of the $cars array (we still have to point to the two indices):

ExampleGet your own PHP Server

```
<?php
for ($row = 0; $row <4; $row++) {
  echo"<p><b>Row number $row</b></p>";
  echo"<ul>";
  for ($col = 0; $col <3; $col++) {
    echo"<li>".$cars[$row][$col]."</li>";
  }
  echo"</ul>";
}
?>
```

Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Sorting Arrays

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- `sort()` - sort arrays in ascending order
- `rsort()` - sort arrays in descending order
- `asort()` - sort associative arrays in ascending order, according to the value
- `ksort()` - sort associative arrays in ascending order, according to the key
- `arsort()` - sort associative arrays in descending order, according to the value
- `krsort()` - sort associative arrays in descending order, according to the key

---

Sort Array in Ascending Order - sort()

The following example sorts the elements of the $cars array in ascending alphabetical order:

Example[Get your own PHP Server]

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
?>
```

The following example sorts the elements of the $numbers array in ascending numerical order:

Example[Get your own PHP Server]

```
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
?>
```

Sort Array in Descending Order - rsort()

The following example sorts the elements of the $cars array in descending alphabetical order:

Example[Get your own PHP Server]

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
?>
```

The following example sorts the elements of the $numbers array in descending numerical order:

ExampleGet your own PHP Server

```
<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);
?>
```

Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

ExampleGet your own PHP Server

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
?>
```

Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

ExampleGet your own PHP Server

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

ExampleGet your own PHP Server

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example[**Get your own PHP Server**]

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
?>
```

Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises

Exercise:

Use the correct array method to sort the `$colors` array alphabetically.

```
$colors = array("red", "green", "blue", "yellow");
        ;
```

PHP Global Variables - Superglobals

Superglobals were introduced in PHP 4.1.0, and are built-in variables that are always available in all scopes.

PHP Global Variables - Superglobals

Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

The next chapters will explain some of the superglobals, and the rest will be explained in later chapters.

PHP Superglobal - $GLOBALS

Super global variables are built-in variables that are always available in all scopes.

PHP $GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

The example below shows how to use the super global variable $GLOBALS:

Example[**Get your own PHP Server**]

```
<?php
$x = 75;
$y = 25;

function addition() {
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>
```

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function!

PHP Superglobal - $_SERVER

Super global variables are built-in variables that are always available in all scopes.

PHP $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

The example below shows how to use some of the elements in $_SERVER:

Example[Get your own PHP Server]

```
<?php
echo$_SERVER['PHP_SELF'];
echo"<br>";
echo$_SERVER['SERVER_NAME'];
echo"<br>";
echo$_SERVER['HTTP_HOST'];
echo"<br>";
echo$_SERVER['HTTP_REFERER'];
echo"<br>";
echo$_SERVER['HTTP_USER_AGENT'];
echo"<br>";
echo$_SERVER['SCRIPT_NAME'];
?>
```

The following table lists the most important elements that can go inside $_SERVER:

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using

$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-

	generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP Superglobal - $_REQUEST

Super global variables are built-in variables that are always available in all scopes.

PHP $_REQUEST

PHP $_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example **Get your own PHP Server**

```
<html>
<body>

<form method="post" action="<?phpecho$_SERVER['PHP_SELF'];?>">
 Name: <input type="text" name="fname">
 <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
 // collect value of input field
 $name = $_REQUEST['fname'];
 if (empty($name)) {
  echo"Name is empty";
 } else {
```

```
  echo $name;
 }
}
?>

</body>
</html>
```

PHP Superglobal - $_POST

Super global variables are built-in variables that are always available in all scopes.

PHP $_POST

PHP $_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example **Get your own PHP Server**

```
<html>
<body>

<form method="post" action="<?phpecho$_SERVER['PHP_SELF'];?>">
 Name: <input type="text" name="fname">
 <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
 // collect value of input field
 $name = $_POST['fname'];
 if (empty($name)) {
  echo"Name is empty";
 } else {
```

```
  echo $name;
 }
}
?>

</body>
</html>
```

PHP Superglobal - $_GET

Super global variables are built-in variables that are always available in all scopes.

PHP $_GET

PHP $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".

$_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>

<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>

</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "test_get.php", and you can then access their values in "test_get.php" with $_GET.

The example below shows the code in "test_get.php":

Example **Get your own PHP Server**

```
<html>
<body>

<?php
```

```
echo"Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

</body>
</html>
```

PHP Regular Expressions

What is a Regular Expression?

A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of text search and text replace operations.

Syntax

In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

```
$exp = "/w3schools/i";
```

In the example above, `/` is the **delimiter**, *w3schools* is the **pattern** that is being searched for, and `i` is a **modifier** that makes the search case-insensitive.

The delimiter can be any character that is not a letter, number, backslash or space. The most common delimiter is the forward slash (/), but when your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

Regular Expression Functions

PHP provides a variety of functions that allow you to use regular expressions. The `preg_match()`, `preg_match_all()` and `preg_replace()` functions are some of the most commonly used ones:

Function	Description

preg_match()	Returns 1 if the pattern was found in the string and 0 if not
preg_match_all()	Returns the number of times the pattern was found in the string, which may also be 0
preg_replace()	Returns a new string where matched patterns have been replaced with another string

---

Using preg_match()

The `preg_match()` function will tell you whether a string contains matches of a pattern.

Example[Get your own PHP Server]

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
<?php
$str = "Visit W3Schools";
$pattern = "/w3schools/i";
echo preg_match($pattern, $str); // Outputs 1
?>
```

Using preg_match_all()

The `preg_match_all()` function will tell you how many matches were found for a pattern in a string.

Example[Get your own PHP Server]

Use a regular expression to do a case-insensitive count of the number of occurrences of "ain" in a string:

```
<?php
$str = "The rain in SPAIN falls mainly on the plains.";
$pattern = "/ain/i";
echo preg_match_all($pattern, $str); // Outputs 4
?>
```

Using preg_replace()

The `preg_replace()` function will replace all of the matches of the pattern in a string with another string.

Example[Get your own PHP Server]

Use a case-insensitive regular expression to replace Microsoft with W3Schools in a string:

```
<?php
$str = "Visit Microsoft!";
$pattern = "/microsoft/i";
echo preg_replace($pattern, "W3Schools", $str); // Outputs "Visit W3Schools!"
?>
```

PHP Form Handling

The PHP superglobals $_GET and $_POST are used to collect form-data.

PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example[Get your own PHP Server]

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

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

```
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

The output could be something like this:

```
Welcome John
Your email address is john.doe@example.com
```

The same result could also be achieved using the HTTP GET method:

Example[**Get your own PHP Server**]

```
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

and "welcome_get.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

Think SECURITY when processing PHP forms!

This page does not contain any form validation, it just shows how you can send and retrieve form data.

However, the next pages will show how to process PHP forms with security in mind! Proper validation of form data is important to protect your form from hackers and spammers!

PHP Form Validation

This and the next chapters show how to use PHP to validate form data.

PHP Form Validation

Think SECURITY when processing PHP forms!

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:

The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

First we will look at the plain HTML code for the form:

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
```

The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?phpecho htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

When the form is submitted, the form data is sent with method="post".

What is the $_SERVER["PHP_SELF"] variable?

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the htmlspecialchars() function?

The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The $_SERVER["PHP_SELF"] variable can be used by hackers!

If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?phpecho$_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

So far, so good.

However, consider that a user enters the following URL in the address bar:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

```
<form method="post" action="<?phpecho htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.

When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

<script>location.href('http://www.hacked.com')</script>

- this would not be executed, because it would be saved as HTML escaped code, like this:

<script>location.href('http://www.hacked.com')</script>

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test_input().

Now, we can check each $_POST variable with the test_input() function, and the script looks like this:

ExampleGet your own PHP Server

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
```

```
 $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

Notice that at the start of the script, we check whether the form has been submitted using $_SERVER["REQUEST_METHOD"]. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

The next step is to make input fields required and create error messages if needed.

PHP Forms - Required Fields

This chapter shows how to make input fields required and create error messages if needed.

PHP - Required Fields

From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL

Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

In the previous chapter, all input fields were optional.

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an `if else` statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP `empty()` function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the `test_input()` function:

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
 if (empty($_POST["name"])) {
  $nameErr = "Name is required";
 } else {
  $name = test_input($_POST["name"]);
 }

 if (empty($_POST["email"])){
  $emailErr = "Email is required";
 } else {
  $email = test_input($_POST["email"]);
 }

 if (empty($_POST["website"])){
  $website = "";
 } else {
  $website = test_input($_POST["website"]);
 }

 if (empty($_POST["comment"])) {
  $comment = "";
 } else {
  $comment = test_input($_POST["comment"]);
 }
```

```
 if (empty($_POST["gender"])) {
   $genderErr = "Gender is required";
 } else{
   $gender = test_input($_POST["gender"]);
 }
}
?>
```

PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

Example[Get your own PHP Server]

```
<form method="post" action="<?phpecho htmlspecialchars($_SERVER["PHP_SELF"]);?>">

Name: <input type="text" name="name">
<span class="error">* <?phpecho $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?phpecho $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?phpecho $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
<span class="error">* <?phpecho $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">
```

```
</form>
```

PHP Forms - Validate E-mail and URL

This chapter shows how to validate names, e-mails, and URLs.

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters, dashes, apostrophes and whitespaces. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.

In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

PHP - Validate Name, E-mail, and URL

Now, the script looks like this:

Example[Get your own PHP Server]

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
```

```
  $website = "";
 } else {
  $website = test_input($_POST["website"]);
  // check if URL address syntax is valid (this regular expression also allows dashes in the URL)
  if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
    $websiteErr = "Invalid URL";
  }
 }

 if (empty($_POST["comment"])) {
  $comment = "";
 } else {
  $comment = test_input($_POST["comment"]);
 }

 if (empty($_POST["gender"])) {
  $genderErr = "Gender is required";
 } else {
  $gender = test_input($_POST["gender"]);
 }
}
?>
```

PHP Complete Form Example

This chapter shows how to keep the values in the input fields when the user hits the submit button.

PHP - Keep The Values in The Form

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the <textarea> and </textarea> tags. The little script outputs the value of the $name, $email, $website, and $comment variables.

Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

```
Name: <input type="text" name="name" value="<?phpecho $name;?>">

E-mail: <input type="text" name="email" value="<?phpecho $email;?>">

Website: <input type="text" name="website" value="<?phpecho $website;?>">

Comment: <textarea name="comment" rows="5" cols="40"><?phpecho $comment;?></textarea>

Gender:
<input type="radio" name="gender"
<?phpif (isset($gender) && $gender=="female") echo"checked";?>
value="female">Female
<input type="radio" name="gender"
<?phpif (isset($gender) && $gender=="male") echo"checked";?>
value="male">Male
<input type="radio" name="gender"
<?phpif (isset($gender) && $gender=="other") echo"checked";?>
value="other">Other
```

PHP - Complete Form Example

Here is the complete code for the PHP Form Validation Example:

PHP Date and Time

The PHP `date()` function is used to format a date and/or a time.

The PHP Date() Function

The PHP `date()` function formats a timestamp to a more readable date and time.

Syntax

date(*format*,*timestamp*)

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

Get a Date

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

The example below formats today's date in three different ways:

Example[Get your own PHP Server]

```
<?php
echo"Today is " . date("Y/m/d") . "<br>";
echo"Today is " . date("Y.m.d") . "<br>";
echo"Today is " . date("Y-m-d") . "<br>";
echo"Today is " . date("l");
?>
```

PHP Tip - Automatic Copyright Year

Use the `date()` function to automatically update the copyright year on your website:

Example[Get your own PHP Server]

```
&copy; 2010-<?phpecho date("Y");?>
```

Get a Time

Here are some characters that are commonly used for times:

- H - 24-hour format of an hour (00 to 23)
- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

Example **Get your own PHP Server**

```
<?php
echo"The time is " . date("h:i:sa");
?>
```

Note that the PHP date() function will return the current date/time of the server!

Get Your Time Zone

If the time you got back from the code is not correct, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set the timezone you want to use.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

Example **Get your own PHP Server**

```
<?php
date_default_timezone_set("America/New_York");
echo"The time is " . date("h:i:sa");
?>
```

Create a Date With mktime()

The optional *timestamp* parameter in the date() function specifies a timestamp. If omitted, the current date and time will be used (as in the examples above).

The PHP `mktime()` function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Syntax

mktime(*hour, minute, second, month, day, year*)

The example below creates a date and time with the `date()` function from a number of parameters in the `mktime()` function:

ExampleGet your own PHP Server

```
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo"Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

Create a Date From a String With strtotime()

The PHP `strtotime()` function is used to convert a human readable date string into a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).

Syntax

strtotime(*time, now*)

The example below creates a date and time from the `strtotime()` function:

ExampleGet your own PHP Server

```
<?php
$d=strtotime("10:30pm April 15 2014");
echo"Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

PHP is quite clever about converting a string to a date, so you can put in various values:

ExampleGet your own PHP Server

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("+3 Months");
```

```
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

However, `strtotime()` is not perfect, so remember to check the strings you put in there.

More Date Examples

The example below outputs the dates for the next six Saturdays:

Example[Get your own PHP Server]

```
<?php
$startdate = strtotime("Saturday");
$enddate = strtotime("+6 weeks", $startdate);

while ($startdate < $enddate) {
  echo date("M d", $startdate) . "<br>";
  $startdate = strtotime("+1 week", $startdate);
}
?>
```

The example below outputs the number of days until 4th of July:

Example[Get your own PHP Server]

```
<?php
$d1=strtotime("July 04");
$d2=ceil(($d1-time())/60/60/24);
echo"There are " . $d2 ." days until 4th of July.";
?>
```

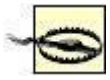
.

Calling System Calls

A system call is used by an application to request service from the operating system. System calls use machine code instructions, which causes the processor to change modes. Changing the mode gets the OS to perform restricted actions; for example, accessing hardware devices or server space available.

Every OS provides a library that sits between normal programs and the rest of the operating system, such as the Windows API. This library handles low-level details of passing information to the kernel and switching to supervisor mode.

You can use the `exec` function to call external functions.



For maximum security, use `exec` only when PHP code doesn't provide the same functionality.

For example, if you'd like to get information about how much space is available on the server, execute the `df` command, shown in Example 11-29. However, this is assuming you're on a Unix host.

Example 11-29. Executing df and displaying the results

```
<?php exec("df",$output_lines,$return_value); echo ("Command returned a value of
$return_value."); echo "</pre>"; foreach ($output_lines as $output) {   echo
"$o"; } echo "</pre>"; ?>
```

PHP Connect to MySQL

PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

Should I Use MySQLi or PDO?

If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

MySQL Examples in Both MySQLi and PDO Syntax

In this, and in the following chapters we demonstrate three ways of working with PHP and MySQL:

- MySQLi (object-oriented)
- MySQLi (procedural)
- PDO

MySQLi Installation

For Linux and Windows: The MySQLi extension is automatically installed in most cases, when php5 mysql package is installed.

For installation details, go to: http://php.net/manual/en/mysqli.installation.php

---

PDO Installation

For installation details, go to: http://php.net/manual/en/pdo.installation.php

---

Open a Connection to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the server:

Example (MySQLi Object-Oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
echo"Connected successfully";
?>
```

Note on the object-oriented example above:

$connect_error was broken until PHP 5.2.9 and 5.3.0. If you need to ensure compatibility with PHP versions prior to 5.2.9 and 5.3.0, use the following code instead:

```
// Check connection
if (mysqli_connect_error()) {
```

```
  die("Database connection failed: " . mysqli_connect_error());
}
```

Example (MySQLi Procedural)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
echo"Connected successfully";
?>
```

Example (PDO)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo"Connected successfully";
} catch(PDOException $e) {
  echo"Connection failed: " . $e->getMessage();
}
?>
```

Note: In the PDO example above we have also **specified a database (myDB)**. PDO require a valid database to connect to. If no database is specified, an exception is thrown.

Tip: A great benefit of PDO is that it has an exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

MySQLi Object-Oriented: Get your own PHP Server

```
$conn->close();
```

MySQLi Procedural: Get your own PHP Server

```
mysqli_close($conn);
```

PDO: Get your own PHP Server

```
$conn = null;
```

PHP Create a MySQL Database

A database consists of one or more tables.

You will need special CREATE privileges to create or to delete a MySQL database.

Create a MySQL Database Using MySQLi and PDO

The CREATE DATABASE statement is used to create a database in MySQL.

The following examples create a database named "myDB":

Example (MySQLi Object-oriented) Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```php
// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
  echo"Database created successfully";
} else {
  echo"Error creating database: " . $conn->error;
}

$conn->close();
?>
```

Note: When you create a new database, you must only specify the first three arguments to the mysqli object (servername, username and password).

Tip: If you have to use a specific port, add an empty string for the database-name argument, like this: new mysqli("localhost", "username", "password", "", port)

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
  echo"Database created successfully";
```

```
} else {
  echo"Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Note: The following PDO example create a database named "myDBPDO":

Example (PDO)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "CREATE DATABASE myDBPDO";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo"Database created successfully<br>";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Create Table

A database table has its own unique name and consists of columns and rows.

Create a MySQL Table Using MySQLi and PDO

The CREATE TABLE statement is used to create a table in MySQL.

We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
```

Notes on the table above:

The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go to our Data Types reference.

After the data type, you can specify other optional attributes for each column:

- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

The following examples shows how to create the table in PHP:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
```

```
// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
  echo"Table MyGuests created successfully";
} else {
  echo"Error creating table: " . $conn->error;
}

$conn->close();
?>
```

Example (MySQLi Procedural)[Get your own PHP Server]

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
```

```
  echo"Table MyGuests created successfully";
} else {
  echo"Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // sql to create table
 $sql = "CREATE TABLE MyGuests (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstname VARCHAR(30) NOT NULL,
  lastname VARCHAR(30) NOT NULL,
  email VARCHAR(50),
  reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
  )";

  // use exec() because no results are returned
  $conn->exec($sql);
  echo"Table MyGuests created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Insert Data

Insert Data Into MySQL Using MySQLi and PDO

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

To learn more about SQL, please visit our SQL tutorial.

In the previous chapter we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.

Note: If a column is AUTO_INCREMENT (like the "id" column) or TIMESTAMP with default update of current_timesamp (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

The following examples add a new record to the "MyGuests" table:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
```

```
  echo"New record created successfully";
} else {
  echo"Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```


Example (MySQLi Procedural)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
  echo"New record created successfully";
} else {
  echo"Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo"New record created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Get Last Inserted ID

Get ID of The Last Inserted Record

If we perform an INSERT or UPDATE on a table with an AUTO_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

In the table "MyGuests", the "id" column is an AUTO_INCREMENT field:

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
```

The following examples are equal to the examples from the previous page (PHP Insert Data Into MySQL), except that we have added one single line of code to retrieve the ID of the last inserted record. We also echo the last inserted ID:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
```

```
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
  $last_id = $conn->insert_id;
  echo"New record created successfully. Last inserted ID is: " . $last_id;
} else {
  echo"Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```


Example (MySQLi Procedural)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
```

```
  $last_id = mysqli_insert_id($conn);
  echo"New record created successfully. Last inserted ID is: " . $last_id;
} else {
  echo"Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  $last_id = $conn->lastInsertId();
  echo"New record created successfully. Last inserted ID is: " . $last_id;
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Insert Multiple Records

Insert Multiple Records Into MySQL Using MySQLi and PDO

Multiple SQL statements must be executed with the `mysqli_multi_query()` function.

The following examples add three new records to the "MyGuests" table:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if ($conn->multi_query($sql) === TRUE) {
  echo"New records created successfully";
} else {
  echo"Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

Note that each SQL statement must be separated by a semicolon.

PHP MySQL Prepared Statements

Prepared statements are very useful against SQL injections.

Prepared Statements and Bound Parameters

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Compared to executing SQL statements directly, prepared statements have three main advantages:

- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

Prepared Statements in MySQLi

The following example uses prepared statements and bound parameters in MySQLi:

Example (MySQLi with Prepared Statements)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");
```

```
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo"New records created successfully";

$stmt->close();
$conn->close();
?>
```

Code lines to explain from the example above:

"INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)"

In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double or blob value.

Then, have a look at the bind_param() function:

$stmt->bind_param("sss", $firstname, $lastname, $email);

This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.

The argument may be one of four types:

- i - integer
- d - double
- s - string
- b - BLOB

We must have one of these for each parameter.

By telling mysql what type of data to expect, we minimize the risk of SQL injections.

Note: If we want to insert any data from external sources (like user input), it is very important that the data is sanitized and validated.

PHP MySQL Select Data

Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

SELECT column_name(s) FROM table_name

or we can use the * character to select ALL columns from a table:

SELECT * FROM table_name

To learn more about SQL, please visit our SQL tutorial.

Select Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
```

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows >0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo"id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo"0 results";
}
$conn->close();
?>
```

Code lines to explain from the example above:

First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the function `num_rows()` checks if there are more than zero rows returned.

If there are more than zero rows returned, the function `fetch_assoc()` puts all the results into an associative array that we can loop through. The `while()` loop loops through the result set and outputs the data from the id, firstname and lastname columns.

The following example shows the same as the example above, in the MySQLi procedural way:

Example (MySQLi Procedural)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);
```

```
if (mysqli_num_rows($result) >0) {
  // output data of each row
  while($row = mysqli_fetch_assoc($result)) {
    echo"id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo"0 results";
}

mysqli_close($conn);
?>
```

PHP MySQL Use The WHERE Clause

Select and Filter Data From a MySQL Database

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

SELECT column_name(s) FROM table_name WHERE column_name operator value

To learn more about SQL, please visit our SQL tutorial.

Select and Filter Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table where the lastname is "Doe", and displays it on the page:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
```

```
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
$result = $conn->query($sql);

if ($result->num_rows >0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo"id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo"0 results";
}
$conn->close();
?>
```

Code lines to explain from the example above:

First, we set up the SQL query that selects the id, firstname and lastname columns from the MyGuests table where the lastname is "Doe". The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the `function num_rows()` checks if there are more than zero rows returned.

If there are more than zero rows returned, the function `fetch_assoc()` puts all the results into an associative array that we can loop through. The `while()` loop loops through the result set and outputs the data from the id, firstname and lastname columns.

The following example shows the same as the example above, in the MySQLi procedural way:

Example (MySQLi Procedural)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
```

```
$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) >0) {
  // output data of each row
  while($row = mysqli_fetch_assoc($result)) {
    echo"id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo"0 results";
}

mysqli_close($conn);
?>
```

PHP MySQL Use The ORDER BY Clause

Select and Order Data From a MySQL Database

The ORDER BY clause is used to sort the result-set in ascending or descending order.

The ORDER BY clause sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC

To learn more about SQL, please visit our SQL tutorial.

Select and Order Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table. The records will be ordered by the lastname column:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = $conn->query($sql);

if ($result->num_rows >0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo"id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo"0 results";
}
$conn->close();
?>
```

Code lines to explain from the example above:

First, we set up the SQL query that selects the id, firstname and lastname columns from the MyGuests table. The records will be ordered by the lastname column. The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the `function num_rows()` checks if there are more than zero rows returned.

If there are more than zero rows returned, the function `fetch_assoc()` puts all the results into an associative array that we can loop through. The `while()` loop loops through the result set and outputs the data from the id, firstname and lastname columns.

The following example shows the same as the example above, in the MySQLi procedural way:

Example (MySQLi Procedural)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
```

```
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) >0) {
  // output data of each row
  while($row = mysqli_fetch_assoc($result)) {
    echo"id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo"0 results";
}

mysqli_close($conn);
?>
```

PHP MySQL Delete Data

Delete Data From a MySQL Table Using MySQLi and PDO

The DELETE statement is used to delete records from a table:

```
DELETE FROM table_name
WHERE some_column = some_value
```

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

To learn more about SQL, please visit our SQL tutorial.

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

The following examples delete the record with id=3 in the "MyGuests" table:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
  echo"Record deleted successfully";
} else {
  echo"Error deleting record: " . $conn->error;
}

$conn->close();
?>
```

PHP MySQL Update Data

Update Data In a MySQL Table Using MySQLi and PDO

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

To learn more about SQL, please visit our SQL tutorial.

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in the "MyGuests" table:

Example (MySQLi Object-oriented)Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
  echo"Record updated successfully";
} else {
  echo"Error updating record: " . $conn->error;
}

$conn->close();
?>
```

PHP MySQL Limit Data Selections

Limit Data Selections From a MySQL Database

MySQL provides a LIMIT clause that is used to specify the number of records to return.

The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.

Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

$sql = "SELECT * FROM Orders LIMIT 30";

When the SQL query above is run, it will return the first 30 records.

What if we want to select records 16 - 25 (inclusive)?

Mysql also provides a way to handle this: by using OFFSET.

The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":

$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";

You could also use a shorter syntax to achieve the same result:

$sql = "SELECT * FROM Orders LIMIT 15, 10";

Notice that the numbers are reversed when you use a comma.

PHP Form Handling

The PHP superglobals $_GET and $_POST are used to collect form-data.

PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example[Get your own PHP Server]

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
```

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

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

The output could be something like this:

Welcome John
Your email address is john.doe@example.com

The same result could also be achieved using the HTTP GET method:

Example[Get your own PHP Server]

```
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

and "welcome_get.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
```

</body>
</html>

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

Think SECURITY when processing PHP forms!

This page does not contain any form validation, it just shows how you can send and retrieve form data.

However, the next pages will show how to process PHP forms with security in mind! Proper validation of form data is important to protect your form from hackers and spammers!

GET vs. POST

Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the URL parameters.

$_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Developers prefer POST for sending form data.

Next, lets see how we can process PHP forms the secure way!

PHP Exercises

Test Yourself With Exercises

Exercise:

If the form in the white section below gets submitted, how can you, in welcome.php, output the value from the "first name" field?

```
<form action="welcome.php" method="get">
First name: <input type="text" name="fname">
</form>

<html>
<body>

Welcome <?php echo [          ]; ?>
</body>
</html>
```

PHP Cookies

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the `setcookie()` function.

Syntax

setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the `isset()` function to find out if the cookie is set:

Example[Get your own PHP Server]

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo"Cookie named '" . $cookie_name . "' is not set!";
} else {
  echo"Cookie '" . $cookie_name . "' is set!<br>";
  echo"Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Note: The `setcookie()` function must appear BEFORE the <html> tag.

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use `setrawcookie()` instead).

PHP Sessions

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

A session is started with the `session_start()` function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

ExampleGet your own PHP Server

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
```

```
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo"Session variables are set.";
?>

</body>
</html>
```

HTTP authentication with PHP ¶

It is possible to use the header() function to send an "Authentication Required" message to the client browser causing it to pop up a Username/Password input window. Once the user has filled in a username and a password, the URL containing the PHP script will be called again with the predefined variables*PHP_AUTH_USER*, *PHP_AUTH_PW*, and *AUTH_TYPE* set to the user name, password and authentication type respectively. These predefined variables are found in the *$_SERVER* array. *Only* "Basic" and "Digest" authentication methods are supported. See the header() function for more information.

An example script fragment which would force client authentication on a page is as follows:

Example #1 Basic HTTP Authentication example

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
header('WWW-Authenticate: Basic realm="My Realm"');
header('HTTP/1.0 401 Unauthorized');
echo 'Text to send if user hits Cancel button';
exit;
} else {
echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
}
?>
```

Example #2 Digest HTTP Authentication example

This example shows you how to implement a simple Digest HTTP authentication script. For more information read the » RFC 2617.

```
<?php
$realm = 'Restricted area';

//user => password
$users = array('admin' =>'mypass', 'guest' =>'guest');


if (empty($_SERVER['PHP_AUTH_DIGEST'])) {
```

```
header('HTTP/1.1 401 Unauthorized');
header('WWW-Authenticate: Digest realm="'.$realm.
'",qop="auth",nonce="'.uniqid().'",opaque="'.md5($realm).'"');

die('Text to send if user hits Cancel button');
}


// analyze the PHP_AUTH_DIGEST variable
if (!($data = http_digest_parse($_SERVER['PHP_AUTH_DIGEST'])) ||
!isset($users[$data['username']]))
die('Wrong Credentials!');


// generate the valid response
$A1 = md5($data['username'] . ':' . $realm . ':' . $users[$data['username']]);
$A2 = md5($_SERVER['REQUEST_METHOD'].':'.$data['uri']);
$valid_response =
md5($A1.':'.$data['nonce'].':'.$data['nc'].':'.$data['cnonce'].':'.$data['qop'].':'.$A2);

if ($data['response'] != $valid_response)
die('Wrong Credentials!');

// ok, valid username & password
echo 'You are logged in as: ' . $data['username'];


// function to parse the http auth header
function http_digest_parse($txt)
{
// protect against missing data
$needed_parts = array('nonce'=>1, 'nc'=>1, 'cnonce'=>1, 'qop'=>1, 'username'=>1,
'uri'=>1, 'response'=>1);
$data = array();
$keys = implode('|', array_keys($needed_parts));

preg_match_all('@(' . $keys . ')=(?:([\'"])([^\2]+?)\2|([^\s,]+))@', $txt, $matches,
PREG_SET_ORDER);

foreach ($matches as $m) {
$data[$m[1]] = $m[3] ? $m[3] : $m[4];
unset($needed_parts[$m[1]]);
}

return $needed_parts ? false : $data;
}
?>
```

Note: **Compatibility Note**

Please be careful when coding the HTTP header lines. In order to guarantee maximum compatibility with all clients, the keyword "Basic" should be written with an uppercase "B", the realm string must be enclosed in double (not single) quotes, and exactly one space should precede the *401* code in the *HTTP/1.0 401* header line. Authentication parameters have to be comma-separated as seen in the digest example above.

Instead of simply printing out *PHP_AUTH_USER* and *PHP_AUTH_PW*, as done in the above example, you may want to check the username and password for validity. Perhaps by sending a query to a database, or by looking up the user in a dbm file.

Watch out for buggy Internet Explorer browsers out there. They seem very picky about the order of the headers. Sending the *WWW-Authenticate* header before the `HTTP/1.0 401` header seems to do the trick for now.

Note: **Configuration Note**

PHP uses the presence of an `AuthType` directive to determine whether external authentication is in effect.

Note, however, that the above does not prevent someone who controls a non-authenticated URL from stealing passwords from authenticated URLs on the same server.

Both Netscape Navigator and Internet Explorer will clear the local browser window's authentication cache for the realm upon receiving a server response of 401. This can effectively "log out" a user, forcing them to re-enter their username and password. Some people use this to "time out" logins, or provide a "log-out" button.

Example #3 HTTP Authentication example forcing a new name/password

```
<?php
function authenticate() {
header('WWW-Authenticate: Basic realm="Test Authentication System"');
header('HTTP/1.0 401 Unauthorized');
echo "You must enter a valid login ID and password to access this resource\n";
exit;
}

if (!isset($_SERVER['PHP_AUTH_USER']) ||
($_POST['SeenBefore'] == 1 &&$_POST['OldAuth'] == $_SERVER['PHP_AUTH_USER'])) {
authenticate();
} else {
echo "<p>Welcome: " . htmlspecialchars($_SERVER['PHP_AUTH_USER']) . "<br />";
echo "Old: " . htmlspecialchars($_REQUEST['OldAuth']);
echo "<form action='' method='post'>\n";
echo "<input type='hidden' name='SeenBefore' value='1' />\n";
echo "<input type='hidden' name='OldAuth' value=\"" .
htmlspecialchars($_SERVER['PHP_AUTH_USER']) . "\" />\n";
echo "<input type='submit' value='Re Authenticate' />\n";
echo "</form></p>\n";
}
?>
```

PHP Error Handling

Error handling in PHP is simple. An error message with filename, line number and a message describing the error is sent to the browser.

PHP Error Handling

When creating scripts and web applications, error handling is an important part. If your code lacks error checking code, your program may look very unprofessional and you may be open to security risks.

This tutorial contains some of the most common error checking methods in PHP.

We will show different error handling methods:

- Simple "die()" statements
- Custom errors and error triggers
- Error reporting

Basic Error Handling: Using the die() function

The first example shows a simple script that opens a text file:

Example

```
<?php
$file=fopen("mytestfile.txt","r");
?>
```

If the file does not exist you might get an error like this:

Warning: fopen(mytestfile.txt) [function.fopen]: failed to open stream:
No such file or directory in **C:\webfolder\test.php** on line **2**

To prevent the user from getting an error message like the one above, we test whether the file exist before we try to access it:

Example

```
<?php
if(file_exists("mytestfile.txt")) {
  $file = fopen("mytestfile.txt", "r");
} else {
  die("Error: The file does not exist.");
}
?>
```

Now if the file does not exist you get an error like this:

Error: The file does not exist.

The code above is more efficient than the earlier code, because it uses a simple error handling mechanism to stop the script after the error.

However, simply stopping the script is not always the right way to go. Let's take a look at alternative PHP functions for handling errors.

Creating a Custom Error Handler

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.

This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax

error_function(error_level,error_message,
error_file,error_line,error_context)

Parameter	Description
error_level	Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels
error_message	Required. Specifies the error message for the user-defined error
error_file	Optional. Specifies the filename in which the error occurred
error_line	Optional. Specifies the line number in which the error occurred
error_context	Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Error Report levels

These error report levels are the different types of error the user-defined error handler can be used for:

Value	Constant	Description
1	E_ERROR	A fatal run-time error. Execution of the script is stopped
2	E_WARNING	A non-fatal run-time error. Execution of the script is not stopped
8	E_NOTICE	A run-time notice. The script found something that might be an error, but could also happen when running a script normally
256	E_USER_ERROR	A fatal user-generated error. This is like an E_ERROR, except it is generated by the PHP script using the function trigger_error()
512	E_USER_WARNING	A non-fatal user-generated warning. This is like an E_WARNING, except it is generated by the PHP script using the function

		trigger_error()
1024	E_USER_NOTICE	A user-generated notice. This is like an E_NOTICE, except it is generated by the PHP script using the function trigger_error()
2048	E_STRICT	Not strictly an error.
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

Now lets create a function to handle errors:

```
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}
```

The code above is a simple error handling function. When it is triggered, it gets the error level and an error message. It then outputs the error level and message and terminates the script.

Now that we have created an error handling function we need to decide when it should be triggered.

Set Error Handler

The default error handler for PHP is the built in error handler. We are going to make the function above the default error handler for the duration of the script.

It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. However, in this example we are going to use our custom error handler for all errors:

```
set_error_handler("customError");
```

Since we want our custom function to handle all errors, the `set_error_handler()` only needed one parameter, a second parameter could be added to specify an error level.

Example

Testing the error handler by trying to output variable that does not exist:

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo"<b>Error:</b> [$errno] $errstr";
}
```

```
//set error handler
set_error_handler("customError");

//trigger error
echo($test);
?>
```

The output of the code above should be something like this:

Error: [8] Undefined variable: test

Trigger an Error

In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this is done by the `trigger_error()` function.

Example

In this example an error occurs if the "test" variable is bigger than "1":

```
<?php
$test=2;
if ($test>=1){
  trigger_error("Value must be 1 or below");
}
?>
```

The output of the code above should be something like this:

Notice: Value must be 1 or below
in **C:\webfolder\test.php** on line **6**

An error can be triggered anywhere you wish in a script, and by adding a second parameter, you can specify what error level is triggered.

Possible error types:

- E_USER_ERROR - Fatal user-generated run-time error. Errors that can not be recovered from. Execution of the script is halted
- E_USER_WARNING - Non-fatal user-generated run-time warning. Execution of the script is not halted
- E_USER_NOTICE - Default. User-generated run-time notice. The script found something that might be an error, but could also happen when running a script normally

Example

In this example an E_USER_WARNING occurs if the "test" variable is bigger than "1". If an E_USER_WARNING occurs we will use our custom error handler and end the script:

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo"<b>Error:</b> [$errno] $errstr<br>";
  echo"Ending Script";
  die();
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

The output of the code above should be something like this:

Error: [512] Value must be 1 or below
Ending Script

Now that we have learned to create our own errors and how to trigger them, lets take a look at error logging.

Error Logging

By default, PHP sends an error log to the server's logging system or a file, depending on how the error_log configuration is set in the php.ini file. By using the `error_log()` function you can send error logs to a specified file or a remote destination.

Sending error messages to yourself by e-mail can be a good way of getting notified of specific errors.

Send an Error Message by E-Mail

In the example below we will send an e-mail with an error message and end the script, if a specific error occurs:

```php
<?php
//error handler function
function customError($errno, $errstr) {
  echo"<b>Error:</b> [$errno] $errstr<br>";
  echo"Webmaster has been notified";
  error_log("Error: [$errno] $errstr",1,
  "someone@example.com","From: webmaster@example.com");
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

The output of the code above should be something like this:

Error: [512] Value must be 1 or below
Webmaster has been notified

And the mail received from the code above looks like this:

Error: [512] Value must be 1 or below

This should not be used with all errors. Regular errors should be logged on the server using the default PHP logging system.

jQuery Tutorial

jQuery is a JavaScript Library.

jQuery greatly simplifies JavaScript programming.

jQuery is easy to learn.

Examples in Each Chapter

Our "Try it Yourself" editor makes it easy to learn jQuery. You can edit code and view the result in your browser:

Example

```
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
```

Click on the "Try it Yourself" button to see how it works.

---

jQuery Exercises

Test Yourself With Exercises

Exercise:

Use the correct **selector** to hide all <p> elements.

```
$("          ").hide();
```

Start the Exercise

---

jQuery Examples

Learn by examples! At W3Schools you will find a lot of jQuery examples to edit and test yourself.

---

jQuery Quiz Test

Test your jQuery skills at W3Schools!

My Learning

Track your progress with the free "My Learning" program here at W3Schools.

Log in to your account, and start earning points!

This is an optional feature. You can study W3Schools without using My Learning.

---

jQuery References

At W3Schools you will find a complete reference of all jQuery selectors, methods, properties and events.

jQuery Reference

jQuery Introduction

The purpose of jQuery is to make it much easier to use JavaScript on your website.

What You Should Already Know

Before you start studying jQuery, you should have a basic knowledge of:

- HTML
- CSS
- JavaScript

If you want to study these subjects first, find the tutorials on our Home page.

What is jQuery?

jQuery is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Tip: In addition, jQuery has plugins for almost any task out there.

Why jQuery?

There are lots of other JavaScript libraries out there, but jQuery is probably the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

- Google
- Microsoft
- IBM
- Netflix

Will jQuery work in all browsers?

The jQuery team knows all about cross-browser issues, and they have written this knowledge into the jQuery library. jQuery will run exactly the same in all major browsers.

jQuery Get Started

Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

- Download the jQuery library from jQuery.com

- Include jQuery from a CDN, like Google

Downloading jQuery

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed
- Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from jQuery.com.

The jQuery library is a single JavaScript file, and you reference it with the HTML `<script>` tag (notice that the `<script>` tag should be inside the `<head>` section):

```
<head>
<script src="jquery-3.6.3.min.js"></script>
</head>
```

Tip: Place the downloaded file in the same directory as the pages where you wish to use it.

---

jQuery CDN

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).

Google is an example of someone who host jQuery:

Google CDN:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
</head>
```

One big advantage of using the hosted jQuery from Google:

Many users already have downloaded jQuery from Google when visiting another site. As a result, it will be loaded from cache when they visit your site, which leads to faster loading time. Also, most CDN's will make sure that once a user requests a file from it, it will be served from the server closest to them, which also leads to faster loading time

jQuery Syntax

With jQuery you select (query) HTML elements and perform "actions" on them.

---

jQuery Syntax

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **$(*selector*).*action*()**

- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

Examples:

`$(this).hide()` - hides the current element.

`$("p").hide()` - hides all <p> elements.

`$(".test").hide()` - hides all elements with class="test".

`$("#test").hide()` - hides the element with id="test".

Are you familiar with CSS selectors?

jQuery uses CSS syntax to select elements. You will learn more about the selector syntax in the next chapter of this tutorial.

Tip: If you don't know CSS, you can read our CSS Tutorial.

The Document Ready Event

You might have noticed that all jQuery methods in our examples, are inside a document ready event:

```
$(document).ready(function(){

  // jQuery methods go here...
```

});

This is to prevent any jQuery code from running before the document is finished loading (is ready).

It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.

Here are some examples of actions that can fail if methods are run before the document is fully loaded:

- Trying to hide an element that is not created yet
- Trying to get the size of an image that is not loaded yet

Tip: The jQuery team has also created an even shorter method for the document ready event:

```
$(function(){

  // jQuery methods go here...

});
```

Use the syntax you prefer. We think that the document ready event is easier to understand when reading the code.

jQuery Selectors

---

jQuery selectors are one of the most important parts of the jQuery library.

---

jQuery Selectors

jQuery selectors allow you to select and manipulate HTML element(s).

jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.

All selectors in jQuery start with the dollar sign and parentheses: $().

The element Selector

The jQuery element selector selects elements based on the element name.

You can select all <p> elements on a page like this:

```
$("p")
```

Example

When a user clicks on a button, all <p> elements will be hidden:

Example

```
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide();
  });
});
```

The #id Selector

The jQuery *#id* selector uses the id attribute of an HTML tag to find the specific element.

An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific id, write a hash character, followed by the id of the HTML element:

```
$("#test")
```

Example

When a user clicks on a button, the element with id="test" will be hidden:

Example

```
$(document).ready(function(){
  $("button").click(function(){
    $("#test").hide();
  });
});
```

The .class Selector

The jQuery `.class` selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

```
$(".test")
```

Example

When a user clicks on a button, the elements with class="test" will be hidden:

Example

```
$(document).ready(function(){
  $("button").click(function(){
    $(".test").hide();
  });
});
```

More Examples of jQuery Selectors

Syntax	Description	Example
$("*")	Selects all elements	
$(this)	Selects the current HTML element	
$("p.intro")	Selects all <p> elements with class="intro"	
$("p:first")	Selects the first <p> element	
$("ul li:first")	Selects the first element of the first	
$("ul li:first-child")	Selects the first element of every	
$("[href]")	Selects all elements with an href attribute	
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"	
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"	
$(":button")	Selects all <button> elements and <input> elements of type="button"	
$("tr:even")	Selects all even <tr> elements	
$("tr:odd")	Selects all odd <tr> elements	

Use our jQuery Selector Tester to demonstrate the different selectors.

For a complete reference of all the jQuery selectors, please go to our jQuery Selectors Reference.

Functions In a Separate File

If your website contains a lot of pages, and you want your jQuery functions to be easy to maintain, you can put your jQuery functions in a separate .js file.

When we demonstrate jQuery in this tutorial, the functions are added directly into the `<head>` section. However, sometimes it is preferable to place them in a separate file, like this (use the src attribute to refer to the .js file):

Example

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
<script src="my_jquery_functions.js"></script>
</head>
```

jQuery Exercises

Test Yourself With Exercises

Exercise:

Use the correct **selector** to hide all <p> elements.

```
$("          ").hide();
```

Start the Exercise

jQuery Event Methods

jQuery is tailor-made to respond to events in an HTML page.

What are Events?

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

Examples:

- moving a mouse over an element
- selecting a radio button
- clicking on an element

The term **"fires/fired"** is often used with events. Example: "The keypress event is fired, the moment you press a key".

Here are some common DOM events:

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

---

jQuery Syntax For Event Methods

In jQuery, most DOM events have an equivalent jQuery method.

To assign a click event to all paragraphs on a page, you can do this:

```
$("p").click();
```

The next step is to define what should happen when the event fires. You must pass a function to the event:

```
$("p").click(function(){
  // action goes here!!
});
```

Commonly Used jQuery Event Methods

$(document).ready()

The `$(document).ready()` method allows us to execute a function when the document is fully loaded. This event is already explained in the jQuery Syntax chapter.

click()

The `click()` method attaches an event handler function to an HTML element.

The function is executed when the user clicks on the HTML element.

The following example says: When a click event fires on a `<p>` element; hide the current `<p>` element:

Example

```
$("p").click(function(){
  $(this).hide();
});
```

dblclick()

The `dblclick()` method attaches an event handler function to an HTML element.

The function is executed when the user double-clicks on the HTML element:

Example

```
$("p").dblclick(function(){
  $(this).hide();
});
```

mouseenter()

The `mouseenter()` method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer enters the HTML element:

Example

```
$("#p1").mouseenter(function(){
  alert("You entered p1!");
});
```

mouseleave()

The `mouseleave()` method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer leaves the HTML element:

Example

```
$("#p1").mouseleave(function(){
  alert("Bye! You now leave p1!");
});
```

mousedown()

The `mousedown()` method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element:

Example

```
$("#p1").mousedown(function(){
  alert("Mouse down over p1!");
});
```

mouseup()

The `mouseup()` method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element:

Example

```
$("#p1").mouseup(function(){
  alert("Mouse up over p1!");
});
```

hover()

The `hover()` method takes two functions and is a combination of the `mouseenter()` and `mouseleave()` methods.

The first function is executed when the mouse enters the HTML element, and the second function is executed when the mouse leaves the HTML element:

Example

```
$("#p1").hover(function(){
  alert("You entered p1!");
},
function(){
```

```
  alert("Bye! You now leave p1!");
});
```

focus()

The `focus()` method attaches an event handler function to an HTML form field.

The function is executed when the form field gets focus:

Example

```
$("input").focus(function(){
  $(this).css("background-color", "#cccccc");
});
```

blur()

The `blur()` method attaches an event handler function to an HTML form field.

The function is executed when the form field loses focus:

Example

```
$("input").blur(function(){
  $(this).css("background-color", "#ffffff");
});
```

The on() Method

The `on()` method attaches one or more event handlers for the selected elements.

Attach a click event to a `<p>` element:

Example

```
$("p").on("click", function(){
  $(this).hide();
});
```

Attach multiple event handlers to a `<p>` element:

Example

```
$("p").on({
  mouseenter: function(){
    $(this).css("background-color", "lightgray");
  },
  mouseleave: function(){
    $(this).css("background-color", "lightblue");
  },
  click: function(){
    $(this).css("background-color", "yellow");
  }
});
```

---

jQuery Exercises

Test Yourself With Exercises

Exercise:

Use the correct **event** to hide all <p> elements with a "click".

```
$("p").          (function(){
  $(this).hide();
});
```

Start the Exercise

---

jQuery Event Methods

For a full jQuery event reference, please go to our jQuery Events Reference.

jQuery Effects - Hide and Show

Hide, Show, Toggle, Slide, Fade, and Animate. WOW!

Examples

jQuery hide()
Demonstrates a simple jQuery hide() method.

jQuery hide()
Another hide() demonstration. How to hide parts of text.

---

jQuery hide() and show()

With jQuery, you can hide and show HTML elements with the `hide()` and `show()` methods:

Example

```
$("#hide").click(function(){
  $("p").hide();
});

$("#show").click(function(){
  $("p").show();
});
```

Syntax:

$(*selector*).hide(*speed,callback*);

$(*selector*).show(*speed,callback*);

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the `hide()` or `show()` method completes (you will learn more about callback functions in a later chapter).

The following example demonstrates the speed parameter with `hide()`:

Example

```
$("button").click(function(){
  $("p").hide(1000);
});
```

jQuery Effects - Fading

With jQuery you can fade elements in and out of visibility.

Examples

jQuery fadeIn()
Demonstrates the jQuery fadeIn() method.

jQuery fadeOut()
Demonstrates the jQuery fadeOut() method.

jQuery fadeToggle()
Demonstrates the jQuery fadeToggle() method.

jQuery fadeTo()
Demonstrates the jQuery fadeTo() method.

---

jQuery Fading Methods

With jQuery you can fade an element in and out of visibility.

jQuery has the following fade methods:

- `fadeIn()`
- `fadeOut()`
- `fadeToggle()`
- `fadeTo()`

---

jQuery fadeIn() Method

The jQuery `fadeIn()` method is used to fade in a hidden element.

Syntax:

$(*selector*).fadeIn(*speed*,*callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the `fadeIn()` method with different parameters:

Example

```
$("button").click(function(){
  $("#div1").fadeIn();
  $("#div2").fadeIn("slow");
  $("#div3").fadeIn(3000);
});
```

jQuery Effects - Sliding

The jQuery slide methods slide elements up and down.

Examples

jQuery slideDown()
Demonstrates the jQuery slideDown() method.

jQuery slideUp()
Demonstrates the jQuery slideUp() method.

jQuery slideToggle()
Demonstrates the jQuery slideToggle() method.

---

jQuery Sliding Methods

With jQuery you can create a sliding effect on elements.

jQuery has the following slide methods:

- `slideDown()`
- `slideUp()`

- `slideToggle()`

---

jQuery slideDown() Method

The jQuery `slideDown()` method is used to slide down an element.

Syntax:

$(*selector*).slideDown(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the `slideDown()` method:

Example

```
$("#flip").click(function(){
  $("#panel").slideDown();
});
```

---

jQuery slideUp() Method

The jQuery `slideUp()` method is used to slide up an element.

Syntax:

$(*selector*).slideUp(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the `slideUp()` method:

Example

```
$("#flip").click(function(){
  $("#panel").slideUp();
});
```

---

jQuery slideToggle() Method

The jQuery `slideToggle()` method toggles between the `slideDown()` and `slideUp()` methods.

If the elements have been slid down, `slideToggle()` will slide them up.

If the elements have been slid up, `slideToggle()` will slide them down.

$(*selector*).slideToggle(*speed*,*callback*);

The optional speed parameter can take the following values: "slow", "fast", milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the `slideToggle()` method:

Example

```
$("#flip").click(function(){
  $("#panel").slideToggle();
});
```

---

jQuery Exercises

Test Yourself With Exercises

Exercise:

Use a jQuery method to **slide up** a element.

```
$("div").          ();
```

Start the Exercise

---

jQuery Effects Reference

For a complete overview of all jQuery effects, please go to our jQuery Effect Referenc

jQuery Effects - Animation

With jQuery, you can create custom animations.

jQuery

---

jQuery Animations - The animate() Method

The jQuery `animate()` method is used to create custom animations.

Syntax:

$(*selector*).animate({*params*},*speed*,*callback*);

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

The following example demonstrates a simple use of the `animate()` method; it moves a element to the right, until it has reached a left property of 250px:

Example

```
$("button").click(function(){
  $("div").animate({left: '250px'});
});
```

By default, all HTML elements have a static position, and cannot be moved.
To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

---

jQuery animate() - Manipulate Multiple Properties

Notice that multiple properties can be animated at the same time:

Example

```
$("button").click(function(){
  $("div").animate({
    left: '250px',
    opacity: '0.5',
    height: '150px',
    width: '150px'
  });
});
```

Is it possible to manipulate ALL CSS properties with the animate() method?

Yes, almost! However, there is one important thing to remember: all property names must be camel-cased when used with the animate() method: You will need to write paddingLeft instead of padding-left, marginRight instead of margin-right, and so on.

Also, color animation is not included in the core jQuery library.
If you want to animate color, you need to download the Color Animations plugin from jQuery.com.

---

jQuery animate() - Using Relative Values

It is also possible to define relative values (the value is then relative to the element's current value). This is done by putting += or -= in front of the value:

Example

```
$("button").click(function(){
  $("div").animate({
    left: '250px',
    height: '+=150px',
    width: '+=150px'
  });
});
```

---

jQuery animate() - Using Pre-defined Values

You can even specify a property's animation value as "`show`", "`hide`", or "`toggle`":

Example

```
$("button").click(function(){
  $("div").animate({
    height: 'toggle'
  });
});
```

---

jQuery animate() - Uses Queue Functionality

By default, jQuery comes with queue functionality for animations.

This means that if you write multiple `animate()` calls after each other, jQuery creates an "internal" queue with these method calls. Then it runs the animate calls ONE by ONE.

So, if you want to perform different animations after each other, we take advantage of the queue functionality:

Example 1

```
$("button").click(function(){
  var div = $("div");
  div.animate({height: '300px', opacity: '0.4'}, "slow");
  div.animate({width: '300px', opacity: '0.8'}, "slow");
  div.animate({height: '100px', opacity: '0.4'}, "slow");
  div.animate({width: '100px', opacity: '0.8'}, "slow");
});
```

The example below first moves the `<div>` element to the right, and then increases the font size of the text:

Example 2

```
$("button").click(function(){
  var div = $("div");
  div.animate({left: '100px'}, "slow");
  div.animate({fontSize: '3em'}, "slow");
});
```

---

jQuery Exercises

Test Yourself With Exercises

Exercise:

Use the `animate()` method to move a element 250 pixels to the right.

```
$("div").animate({          : '          '});
```

Start the Exercise

---

jQuery Effects Reference

For a complete overview of all jQuery effects, please go to our jQuery Effect Reference.

jQuery Stop Animations

The jQuery stop() method is used to stop animations or effects before it is finished.

Click to slide down/up the panel

Examples

jQuery stop() sliding
Demonstrates the jQuery stop() method.

jQuery stop() animation (with parameters)
Demonstrates the jQuery stop() method.

---

jQuery stop() Method

The jQuery `stop()` method is used to stop an animation or effect before it is finished.

The `stop()` method works for all jQuery effect functions, including sliding, fading and custom animations.

Syntax:

$(*selector*).stop(*stopAll*,*goToEnd*);

The optional stopAll parameter specifies whether also the animation queue should be cleared or not. Default is false, which means that only the active animation will be stopped, allowing any queued animations to be performed afterwards.

The optional goToEnd parameter specifies whether or not to complete the current animation immediately. Default is false.

So, by default, the `stop()` method kills the current animation being performed on the selected element.

The following example demonstrates the `stop()` method, with no parameters:

Example

```
$("#stop").click(function(){
  $("#panel").stop();
});
```

---

jQuery Exercises

Test Yourself With Exercises

Exercise:

Use a jQuery method to **stop** the animation effect of a element.

```
$("div").          ();
```

Start the Exercise

---

jQuery Effects Reference

For a complete overview of all jQuery effects, please go to our jQuery Effect Reference.

jQuery Callback Functions

A callback function is executed after the current effect is 100% finished.

---

jQuery Callback Functions

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: **$(*selector*).hide(*speed,callback*);**

Examples

The example below has a callback parameter that is a function that will be executed after the hide effect is completed:

Example with Callback

```
$("button").click(function(){
  $("p").hide("slow", function(){
    alert("The paragraph is now hidden");
  });
});
```

The example below has no callback parameter, and the alert box will be displayed before the hide effect is completed:

Example without Callback

```
$("button").click(function(){
  $("p").hide(1000);
  alert("The paragraph is now hidden");
});
```