

I HTML:

HTML-Introduction-tag basics- page structure-adding comments working with texts, paragraphs and line break. Emphasizing test- heading and horizontal rules-list-font size, face and color-alignment links-tables-frames.

II

Forms & Images Using Html: Graphics: Introduction-How to work efficiently with images in web pages, image maps, GIF animation, adding multimedia, data collection with html forms textbox, password, list box, combo box, text area, tools for building web page front page.

III

XML & DHTML: Cascading style sheet (CSS)-what is CSS-Why we use CSS-adding CSS to your web pages-Grouping styles-extensible markup language (XML).

IV

Dynamic HTML: Document object model (DCOM)-Accessing HTML & CSS through DCOM Dynamic content styles & positioning-Event bubbling-data binding. JavaScript: Client-side scripting, What is JavaScript, How to develop JavaScript, simple JavaScript, variables, functions, conditions, loops and repetition

V

Advance script, JavaScript and objects, JavaScript own objects, the DOM and web browser environments, forms and validations.

Text Book

1 Pankaj Sharma, “Web Technology”, SkKataria& Sons Bangalore 2011. 2 Mike Mcgrath, “Java Script”, Dream Tech Press 2006, 1st Edition.

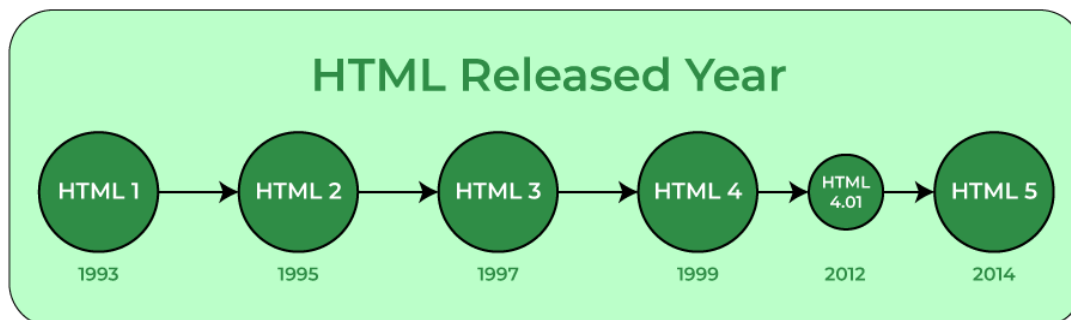
UNIT 1

HTML Introduction

HTML stands for HyperText Markup Language. It is used to design web pages using a markup language. HTML is a combination of Hypertext and Markup language. Hypertext defines the link between web pages. A markup language is used to define the text document within the tag which defines the

structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

HTML is a markup language used by the browser to manipulate text, images, and other content, in order to display it in the required format. HTML was created by Tim Berners-Lee in 1991. The first-ever version of HTML was HTML 1.0, but the first standard version was HTML 2.0, published in 1995.



Elements and Tags:

HTML uses predefined [tags](#) and [elements](#) which tell the browser how to properly display the content. Remember to include closing tags. If omitted, the browser applies the effect of the opening tag until the end of the page.



HTML page structure:

The basic structure of an HTML page is laid out below. It contains the essential building-block elements (i.e. doctype declaration, HTML, head, title, and body elements) upon which all web pages are created.

HTML Page Structure

`<!DOCTYPE html>` ← Tells version of HTML

`<html>` ← HTML Root Element

`<head>` ← Used to contain page HTML metadata

`<title>Page Title</title>` ← Title of HTML page

`</head>`

`<body>` ← Hold content of HTML

`<h2>Heading Content</h2>` ← HTML heading tag

`<p>Paragraph Content</p>` ← HTML paragraph tag

`</body>`

`</html>`

<!DOCTYPE html>: This is the document type declaration (not technically a tag). It declares a document as being an HTML document. The doctype declaration is not case-sensitive.

<html>: This is called the HTML root element. All other elements are contained within it.

<head>: The head tag contains the “behind the scenes” elements for a webpage. Elements within the head aren’t visible on the front-end of a webpage. HTML elements used inside the `<head>` element include:

- **<style>**-This html tag allows us to insert styling into our webpages and make them appealing to look at with the help of CSS.
- **<title>**-The title is what is displayed on the top of your browser when you visit a website and contains the title of the webpage that you are viewing.
- **<base>**-It specifies the base URL for all relative URL’s in a document.
- **<noscript>**– Defines a section of HTML that is inserted when the scripting has been turned off in the users browser.
- **<script>**-This tag is used to add functionality in the website with the help of JavaScript.
- **<meta>**-This tag encloses the meta data of the website that must be loaded every time the website is visited. For eg:- the metadata charset

allows you to use the standard UTF-8 encoding in your website. This in turn allows the users to view your webpage in the language of their choice. It is a self closing tag.

- [<link>](#)– The 'link' tag is used to tie together HTML, CSS, and JavaScript. It is self closing.

[<body>](#): The body tag is used to enclose all the visible content of a webpage. In other words, the body content is what the browser will show on the front-end.

An HTML document can be created using any text editor. Save the text file using **.html** or **.htm**. Once saved as an HTML document, the file can be opened as a webpage in the browser.

Note: Basic/built-in text editors are Notepad (Windows) and TextEdit (Macs). Basic text editors are entirely sufficient for when you're just getting started. As you progress, there are many feature-rich text editors available which allow for greater function and flexibility.

Example: This example illustrates the basic structure of HTML code.

- HTML

```
<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!--The above meta characteristics make a website compatible with different
    devices. -->

    <title>Demo Web Page</title>

</head>

<body>

    <h1>GeeksforGeeks</h1>
```

```
<p>A computer science portal for geeks</p>
```

```
</body>
```

```
</html>
```

Output:

GeeksforGeeks

A computer science portal for geeks

Features of HTML:

- It is easy to learn and easy to use.
- It is platform-independent.
- Images, videos, and audio can be added to a web page.
- Hypertext can be added to the text.
- It is a markup language.

Why learn HTML?

- It is a simple markup language. Its implementation is easy.
- It is used to create a website.
- Helps in developing fundamentals about web programming.
- Boost professional career.

Advantages:

- HTML is used to build websites.
- It is supported by all browsers.
- It can be integrated with other languages like CSS, JavaScript, etc.

Disadvantages:

- HTML can only create static web pages. For dynamic web pages, other languages have to be used.
- A large amount of code has to be written to create a simple web page.
- The security feature is not good.

HTML Comments

HTML comments are not displayed in the browser, but they can help document your HTML source code.

HTML Comment Tag

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 start tag, but not in the end tag.

Note: Comments are not displayed by the browser, but they can help document your HTML source code.

Add Comments

With comments you can place notifications and reminders in your HTML code:

Example

```
<!-- This is a comment -->
```

```
<p>This is a paragraph.</p>
```

```
<!-- Remember to add more information here -->
```

[Try it Yourself »](#)

Hide Content

Comments can be used to hide content.

This can be helpful if you hide content temporarily:

Example

```
<p>This is a paragraph.</p>
```

```
<!-- <p>This is another paragraph </p> -->
```

```
<p>This is a paragraph too.</p>
```

[Try it Yourself »](#)

You can also hide more than one line. Everything between the `<!--` and the `-->` will be hidden from the display.

Example

Hide a section of HTML code:

```
<p>This is a paragraph.</p>
<!--
<p>Look at this cool image:</p>

-->
<p>This is a paragraph too.</p>
```

[Try it Yourself »](#)

Exercise:

Use the HTML comment tag to make a comment out of the "This is a comment" text.

```
<h1>This is a heading</h1>
<!-- This is a comment -->
<p>This is a paragraph.</p>
```

HTML Comments

HTML Comments are used to insert comments in the HTML code. It is a good practice of coding so that the coder and the reader can get help to understand the code. It is a simple piece of code that is wiped off (ignored) by web browsers i.e., not displayed by the browser.

Comment Types

There are three types of comments in HTML which are:

Table of Content

- [Comment Types](#)
- [Single-line Comment](#)
- [Multi-line Comment](#)
- [Using comment tag](#)

We will explore different types of Comments with suitable examples.

Single-line Comment

The single-line comment is given inside the (<!-- comment -->) tag.

Syntax

```
<!-- comment -->
```

Example: In this example, we will see the implementation of the above approach.

- HTML

```
<!DOCTYPE html>

<html>

<body>

    <!--This is heading Tag, It wont be displayed by the browser -->

    <h1>GeeksforGeeks</h1>


    <!--This is single line comment,It wont be displayed by the browser -->

    <h2>This is single line comment</h2>


</body>

</html>
```

Output:

Multi-line Comment

Multiple lines can be given by the syntax (`<!-- -->`), Basically it's the same as we used in single line comment, difference is half part of the comment (`" -->`), is appended where the intended comment line ends.

Syntax

```
<!-- Multi
Line
Comment -->
```

Example: In this example, we will see the implementation of above approach.

- HTML

```
<!DOCTYPE html>

<html>

<body>


    <!-- This is

        heading tag -->

    <h1>GeeksforGeeks</h1>


    <!-- This is

        multi-line

        comment -->

    <h2>This is multi-line comment</h2>
```

```
</body>
```

```
</html>
```

Output:

Using comment tag

There used to be an HTML **<comment>** tag, but currently it is not supported by any modern browser.

Syntax

```
<comment> Some Comment </comment>
```

Example: In this example, we will see the implementation of above approach.

- HTML

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
    <comment>This is heading tag</comment>
```

```
    <h1>GeeksforGeeks</h1>
```

```
    <comment>This is multi-line
```

```
        comment
```

```
    </comment>
```

```
<h2>This is a comment using</h2>

</body>

</html>
```

Output:

GeeksforGeeks

This is a comment using comment Tag

Comments

Sometimes while writing HTML, you want to leave a comment, either for yourself or for another person. Keep in mind that, being comments, they will **not be rendered at all** and will in fact be completely ignored by the browser. You can think of comments as just little notes that do not affect anything else.

You can achieve this in HTML like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Comments</title>
  </head>
  <body>
    This is content.
    <!-- This is a comment. -->
  </body>
</html>
```

You can also comment in the middle of a sentence, or have comments that span multiple lines, like so:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Multi-line Comments</title>
  </head>
  <body>
    This is <!-- Comment inside. --> some content.
    <!--
    I am
    a multiple line
    comment -->
  </body>
</html>
```

Comments are useful depending on how you use them, but they are completely optional.



Multiline Content

Working with multiline content in HTML is pretty easy, but it is not like simply pressing enter on a Word document. For example, this will not give you the expected outcome:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Multiline Content</title>
  </head>
  <body>
    I am on the first line
    and I am on the second line.
  </body>
</html>
```

They will render **on the same line!**

render inline.

Without tags, content will

⌚ Paragraph Tags

- ⌚ The reason for this is the way HTML is eventually parsed by the browser. With no tags separating the two lines, the content is essentially treated as being together, and are thus rendered together, on the same line.

To get the desired effect in this case, you will need to use the paragraph tag, or `p` tag.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Paragraph Tags</title>
  </head>
  <body>
    <p>I am on the first line</p>
    <p>and I am on the second line.</p>
  </body>
</html>
```

Paragraph tags start on a new line.

With each line being encompassed by their own paragraph tag, the browser treats them as separate pieces of content and renders them as such. Paragraph tags are used for writing, well, paragraphs, which each paragraph being neatly separated from each other as you would expect.

Line Break Tags

There is another way to accomplish a similar effect as above. It is usually discouraged unless in very specific cases, but that case might come up for you.

If you use a **line break** tag, or `br` tag, you can tell the browser to start a new line and render the rest of the content there instead.

Using a line break tag, we get this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Line Break Tags</title>
  </head>
  <body>
    I am on the first line<br>
    and I am on the second line.
  </body>
</html>
```

Line break tags force a new line.

Our content renders on separate lines, and we are now happy campers! We can now write content that is more readable and easier to read.

Emphasis

Adding emphasis to content is a great way to highlight that selection over the surrounding content. HTML gives you two ways to give **emphasis** to content.

You have the em tag, which stands for **emphasis**, and the strong tag, which is to give content a **strong importance**. You can use both of these tags to emphasize content.

Load this up in your browser to see for yourself:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Emphasis</title>
  </head>
  <body>
    I <em>really</em> think that HTML is
    <strong>awesome!</strong>
  </body>
</html>
```

An example using emphasis and strong.

How to render as emphasized text using HTML ?

The **** tag in HTML is a phrase tag and used to emphasize the text content. It is similar to **<italic>** tag. The main difference between these two tag is the **** tag semantically emphasizes on the important word or section of words while **<i>** tag is just offset text conventionally styled in italic to show alternative mood or voice.

Note: This effect can be achieved by using CSS property.

Syntax:

` Contents... `

Example 1: This example uses `` tag to create emphasized text.

```
<!DOCTYPE html>

<html>

<head>

    <title>

        How to render as emphasized

        text using HTML?

    </title>

</head>

<body style="text-align: center;">

    <h1 style="color: green;">

        GeeksforGeeks

    </h1>

    <h3>

        How to render as emphasized

        text using HTML?
```



```
</h3>

<em>Emphasized text content</em>

</body>

</html>
```

Output:

GeeksforGeeks

How to render as emphasized text using HTML?

Emphasized text content

Example 2: This example uses tag with title attribute to create emphasized text.

- HTML

```
<!DOCTYPE html>

<html>

<head>
```

```
<title>

    How to render as emphasized

    text using HTML?

</title>

</head>


<body style="text-align: center;">

    <h1 style="color:green;">

        GeeksforGeeks

    </h1>


    <h3>

        How to render as emphasized

        text using HTML?

    </h3>


    <em title="Emphasized text">

        Emphasized text content

    </em>

</body>


</html>
```

Output:

GeeksforGeeks

How to render as emphasized text using HTML?

Emphasized text content

Supported Browsers:

- Google Chrome
- Internet Explorer
- Firefox
- Safari
- Opera
-

HTML Heading

An HTML heading tag is used to define the headings of a page. There are six levels of headings defined by HTML. These 6 heading elements are h1, h2, h3, h4, h5, and h6; with h1 being the highest level and h6 being the least.

- **<h1>** is used for the main heading. (Biggest in size)
- **<h2>** is used for subheadings, if there are further sections under the subheadings then the **<h3>** elements are used.
- **<h6>** for the small heading (smallest one).

Browsers display the contents of headings in different sizes. The exact size at which each browser shows the heading can vary slightly. Users can also adjust the size of the text in their browser.

Syntax:

// the 'h' inside the tag should be in small case only.

`<h1>Heading1</h1>`

`<h2>Heading2</h2>`

•
•
•
•

`<h6>Heading6</h6>`

Importance of Heading:

- Search Engines use headings for indexing the structure and organizing the content of the webpage.
- Headings are used for highlighting important topics.

- They provide valuable information and tell us about the structure of the document.

Example 1: This example illustrates the HTML heading tags.

- HTML

```
<!DOCTYPE html>

<html>

<body>


    <h1>H1 Heading</h1>


    <!-- With the help of Style attribute you can
    customize

        the size of the heading, As done below-->


    <h1 style="font-size: 50px">H1 with new size.</h1>


    <!-- Here font-size is the property by which we
    can

        modify the heading. Here we kept it 50px
    i.e. 50 pixels -->


</body>

</html>
```

```
<!DOCTYPE html>

<html>

<body>

    <h1>Welcome to GeeksforGeeks</h1>

    <h2>A computer science portal for geeks</h2>

    <h5>Tutorial</h5>

    <h6>Geeks</h6>

</body>

</html>
```

Output:

Welcome to GeeksforGeeks

A computer science portal for geeks

Tutorial

Geeks

Changing the size of HTML Headings: The default size of HTML headings can be changed, using the style attribute.

Example: This example explains the HTML heading tags by specifying the size of the font.

- HTML

```
<!DOCTYPE html>
```

```
<html>

<body>


    <h1>H1 Heading</h1>


    <!-- With the help of Style attribute you can customize
           the size of the heading, As done below-->


    <h1 style="font-size: 50px">H1 with new size.</h1>


    <!-- Here font-size is the property by which we can
           modify the heading. Here we kept it 50px i.e. 50 pixels -->


</body>

</html>
```

Output:

H1 Heading

H1 with new size.

Styling the <h1> tag with different font-size

Horizontal rule: The <hr> tag which stands for the horizontal rule is used to define a thematic break in an HTML page. The <hr> tag is an empty tag, and

it does not require any end tag. It is basically used to separate content. Please refer to the [HTML <hr> Tag](#) article for more detailed information.

Example: This example explains the HTML Headings with horizontal rules.

- HTML

```
<!DOCTYPE html>

<html>

<body>

    <h1>Heading 1</h1>

    <p>I like HTML.</p>

    <!-- hr Tag is used here-->

    <hr />

    <h2>Heading 2</h2>

    <p>I like CSS.</p>

    <!-- hr Tag is used here-->

    <hr />

    <h2>Heading 3</h2>

    <p>I like Javascript.</p>

</body>

</html>
```

Output:

Heading 1

I like HTML.

Heading 2

I like CSS.

Heading 3

I like Javascript.

HTML Heading with the horizontal line

Supported Browsers:

- Google Chrome 93.0
- Microsoft Edge 93.0
- Internet Explorer 11.0
- Firefox 92.0
- Opera 78.0
- Safari 14.1

HTML Tag

The ** tag** in HTML plays an important role in the web page to create an attractive and readable web page. The font tag is used to change the color, size, and style of a text. The base font tag is used to set all the text to the same size, color and face.

Syntax:

```
<font attribute = "value"> Content </font>
```

Example: In this example, we have used the tag with a font size as 5.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <h2>GeeksforGeeks</h2>
```



```
<!--Normal paragraph tag-->

<p>Hello Geeks!.</p>

<!--font tag-->

<font size="5"> Welcome to GeeksforGeeks </font>

</body>

</html>
```

Output:

GeeksforGeeks

Hello Geeks!.

Welcome to GeeksforGeeks

HTML tag

The font tag has basically three attributes which are given below:

- [Font Size attribute](#)
- [Face/Type attribute](#)
- [Color attribute](#)

Note: Font tag is not supported in HTML5.

font Size: This attribute is used to adjust the size of the text in the HTML document using a font tag with the size attribute. The range of size of the font in HTML is from 1 to 7 and the default size is 3.

Syntax:

```
<font size="number">
```

Example: This example uses the tag where different font sizes are specified.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <!--HTML font size tag starts here-->

    <font size="1">GeeksforGeeks!</font><br />

    <font size="2">GeeksforGeeks!</font><br />

    <font size="3">GeeksforGeeks!</font><br />

    <font size="4">GeeksforGeeks!</font><br />

    <font size="5">GeeksforGeeks!</font><br />

    <font size="6">GeeksforGeeks!</font><br />

    <font size="7">GeeksforGeeks!</font>

    <!--HTML font size tag ends here-->

</body>

</html>
```

Output:

GeeksforGeeks!
GeeksforGeeks!
GeeksforGeeks!
GeeksforGeeks!
GeeksforGeeks!
GeeksforGeeks!
GeeksforGeeks!

font size attribute

Font Type: Font type can be set by using face attribute with font tag in HTML document. But the fonts used by the user need to be installed in the system first.

Syntax:

```
<font face="font_family">
```

Example: This example describes the tag with different font type & font size.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <!--HTML font face tag starts here-->

    <font face="Times New Roman" size="6">

        GeeksforGeeks!!

    </font>    <br />

    <font face="Verdana" size="6">

        GeeksforGeeks!!
```

```
</font><br />

<font face="Comic sans MS" size=" 6">

    GeeksforGeeks!!

</font><br />

<font face="WildWest" size="6">

    GeeksforGeeks!!

</font><br />

<font face="Bedrock" size="6">

    GeeksforGeeks!!

</font><br />

<!--HTML font face tag ends here-->

</body>

</html>
```

Output:

GeeksforGeeks!!
GeeksforGeeks!!
GeeksforGeeks!!
GeeksforGeeks!!
GeeksforGeeks!!

font type attribute

Font Color: Font color is used to set the text color using a font tag with the color attribute in an HTML document. Color can be specified either with its name or with its hex code.

Syntax:

```
<font color="color_name|hex_number|rgb_number">
```

Example: This example describes the tag with different font colors.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <!--HTML font color tag starts here-->

    <font color="#009900">GeeksforGeeks</font><br />

    <font color="green">GeeksforGeeks</font>

    <!--HTML font color tag ends here-->

</body>

</html>
```

Output:

GeeksforGeeks
GeeksforGeeks

font color attribute

Supported Browsers:

- Google Chrome
- Microsoft Edge 12 and above

- Internet Explorer
- Firefox
- Opera
- Safari

HTML Lists

A list is a record of short pieces of related information or used to display the data or any information on web pages in the ordered or unordered form. For instance, to purchase the items, we need to prepare a list that can either be ordered or unordered list which helps us to organize the data & easy to find the item. Please refer to the [HTML type Attribute](#) article for the various types of attributes that can be used with the ordered & unordered list.

Example: The below example illustrates the use of the unordered & ordered list in HTML.

HTML

```
<!DOCTYPE html>
<html>

<head>
  <title>GeeksforGeeks</title>
</head>

<body>
  <h2>Welcome To GeeksforGeeks Learning</h2>
  <h5>List of available courses</h5>
  <ul>
    <li>Data Structures & Algorithm</li>
    <li>Web Technology</li>
    <li>Aptitude & Logical Reasoning</li>
    <li>Programming Languages</li>
  </ul>
  <h5>Data Structures topics</h5>
  <ol>
    <li>Array</li>
    <li>Linked List</li>
    <li>Stacks</li>
    <li>Queues</li>
    <li>Trees</li>
    <li>Graphs</li>
  </ol>
</body>

</html>
```

Output:

Welcome To GeeksforGeeks Learning

List of available courses

- Data Structures & Algorithm
- Web Technology
- Aptitude & Logical Reasoning
- Programming Languages

Data Structures topics

1. Array
2. Linked List
3. Stacks
4. Queues
5. Trees
6. Graphs

HTML List

Supported Tags: These tags are used in HTML listing.

- [HTML Tag](#)
- [HTML Tag](#)
- [HTML <dl> Tag](#)

The HTML Unordered List: An unordered list starts with the “ul” tag. Each list item starts with the “li” tag. The list items are marked with bullets i.e small black circles by default.

Syntax:

```
<ul> list of items </ul>
```

Attribute: This tag contains two attributes which are listed below:

- **compact:** It will render the list smaller.
- **type:** It specifies which kind of marker is used in the list.

Note: The attributes are not supported by HTML5.

Example: This example describes the unordered list.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <h2>Grocery list</h2>

    <ul>

        <li>Bread</li>
```

```
<li>Eggs</li>

<li>Milk</li>

<li>Coffee</li>

</ul>

</body>

</html>
```

Output:

Grocery list

- Bread
- Eggs
- Milk
- Coffee

Unordered List

HTML unordered list has various list item markers:

Example 1: The Disc can be used to set the list item marker to a bullet i.e default.

HTML

```
<!DOCTYPE html>

<html>

<head>

    <title>HTML ul tag</title>

</head>
```



```
<body>

    <h1>GeeksforGeeks</h1>

    <h2>Unordered List with Disc Bullets</h2>

<p>GeeksforGeeks courses List:</p>

<ul style="list-style-type:disc">

    <li>Geeks</li>

    <li>Sudo</li>

    <li>Gfg</li>

    <li>Gate</li>

    <li>Placement</li>

</ul>

</body>

</html>
```

Output:

GeeksforGeeks

Unordered List with Disc Bullets

GeeksforGeeks courses List:

- Geeks
- Sudo
- Gfg
- Gate
- Placement

Example 2: The Circle can be used to set the list item marker to a circle. **HTML**

```
<!DOCTYPE html>

<html>

<body>

    <h1>GeeksforGeeks</h1>

    <h2>Unordered List with Circle
    Bullets</h2>

<p>GeeksforGeeks courses List:</p>

<ul style="list-style-type: circle">

    <li>Geeks</li>

    <li>Sudo</li>

    <li>Gfg</li>

    <li>Gate</li>

    <li>Placement</li>

</ul></body>

</html>
```

Output:

GeeksforGeeks

Unordered List with Circle Bullets

GeeksforGeeks courses List:

- Geeks
- Sudo
- Gfg
- Gate
- Placement

Unordered List with circle item maker

Example 3: The Square can be used to set the list item marker to a square.
HTML

```
<!DOCTYPE html>

<html>

<body>

    <h1>GeeksforGeeks</h1>

    <h2>Unordered List with Square Bullets</h2>

<p>GeeksforGeeks courses List:</p>


    <ul style="list-style-type: square">

        <li>Geeks</li>

        <li>Sudo</li>

        <li>Gfg</li>

        <li>Gate</li>
```

```
<li>Placement</li>

</ul>

</body>

</html>
```

Output:

GeeksforGeeks

Unordered List with Square Bullets

GeeksforGeeks courses List:

- Geeks
- Sudo
- Gfg
- Gate
- Placement

Unordered List with square item maker

Example 4: It's none that can be used to set the list item marker with no mark.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <h1>GeeksforGeeks</h1>

    <h2>Unordered List with No Bullets</h2>
```

```
<p>GeeksforGeeks courses List:</p>
```

```
<ul style="list-style-type: none">
```

```
<li>Geeks</li>
```

```
<li>Sudo</li>
```

```
<li>Gfg</li>
```

```
<li>Gate</li>
```

```
<li>Placement</li>
```

```
</ul>
```

```
</body>
```

```
</html>
```

Output:

GeeksforGeeks

Unordered List with No Bullets

GeeksforGeeks courses List:

- Geeks
- Sudo
- Gfg
- Gate
- Placement

Unordered List with none item maker

Example: Nested Unordered List, It is used to nest the list items ie., a list inside another list.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <h1>GeeksforGeeks</h1>

    <h2>Nested Unordered List</h2>

<p>GeeksforGeeks courses List:</p>

<ul>

    <li>DSA</li>

    <ul>

        <li>Array</li>

        <li>Linked List</li>

        <li>stack</li>

        <li>Queue</li>

    </ul>

</ul>
```

```
<li>Web Technologies</li>

<ul>

    <li>HTML</li>

    <li>CSS</li>

    <li>JavaScript</li>

</ul>

<li>Aptitude</li>

<li>Gate</li>

<li>Placement</li>

</ul>

</body>

</html>
```

Output:

GeeksforGeeks

Nested Unordered List

GeeksforGeeks courses List:

- DSA
 - Array
 - Linked List
 - stack
 - Queue
- Web Technologies
 - HTML
 - CSS
 - JavaScript
- Aptitude
- Gate
- Placement

Nested Unordered List

HTML Ordered List: An ordered list starts with the “ol” tag. Each list item starts with the “li” tag. The list items are marked with numbers by default.

Syntax:

```
<ol>

  <li>Item1</li>

  <li>Item2</li>

  <li>Item3</li>

</ol>
```

Attributes:

- **compact**: It defines the list should be compacted (compact attribute is not supported in HTML5. Use CSS instead.).
- **reversed**: It defines that the order will be descending.
- **start**: It defines from which number or alphabet the order will start.
- **type**: It defines which type(1, A, a, I, and i) of the order you want in your list of numeric, alphabetic, or roman numbers.

Example: This example illustrates the use of the reverse attribute, control list counting & type attribute.

HTML

```
<!DOCTYPE html>

<html>
```



```
<head>
```

```
    <title>HTML ol tag</title>
```

```
</head>
```

```
<body>
```

```
    <h1 style="color: green">GeeksforGeeks</h1>
```

```
    <h3>HTML ol tag</h3>
```

```
<p>reversed attribute</p>
```

```
    <ol reversed>
```

```
        <li>HTML</li>
```

```
        <li>CSS</li>
```

```
        <li>JS</li>
```

```
    </ol>
```

```
<p>start attribute</p>
```

```
    <ol start="5">
```

```
        <li>HTML</li>
```

```
        <li>CSS</li>
```

```

        <li>JS</li>

    </ol>

    <p>type attribute</p>

    <ol type="i">

        <li>HTML</li>

        <li>CSS</li>

        <li>JS</li>

    </ol>

</body>

</html>

```

Output:



HTML tag

reversed attribute

3. HTML
2. CSS
1. JS

start attribute

5. HTML
6. CSS
7. JS

type attribute

- i. HTML
- ii. CSS
- iii. JS

Ordered List with different list style

HTML ordered list has various list item markers: The type attribute of the tag defines the type of the list item marker.

Example 1: The list items will be numbered with numbers i.e default.

HTML

```
<!DOCTYPE html>

<html>

<body>

  <h2>Ordered List with Numbers</h2>

  <ol type="1">

    <li>Bread</li>

    <li>Eggs</li>

    <li>Milk</li>

    <li>Coffee</li>

  </ol>

</body>

</html>
```

Output:

Ordered List with Numbers

1. Bread
2. Eggs
3. Milk
4. Coffee

Ordered List with numeric item maker

Example 2: Type="A", this list of items will be numbered with uppercase letters.

HTML

```
<!DOCTYPE html>

<html>

<body>

  <h2>Ordered List with Letters</h2>

  <ol type="A">

    <li>Bread</li>

    <li>Eggs</li>

    <li>Milk</li>

    <li>Coffee</li>

  </ol>

</body>

</html>
```

Output:

Ordered List with Letters

- A. Bread
- B. Eggs
- C. Milk
- D. Coffee

Ordered List with capital alphabetic item maker

Example 3: Type="a", this list of items will be numbered with lowercase letters.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <h2>Ordered List with Lowercase Letters</h2>

    <ol type="a">

        <li>Bread</li>

        <li>Eggs</li>

        <li>Milk</li>

        <li>Coffee</li>

    </ol>

</body>

</html>
```

Output:

Ordered List with Lowercase Letters

- a. Bread
- b. Eggs
- c. Milk
- d. Coffee

Ordered List with small alphabetic item maker

Example 4: Type="I", this list of items will be numbered with uppercase roman numbers.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <h2>Ordered List with Roman Numbers</h2>

    <ol type="I">

        <li>Bread</li>

        <li>Eggs</li>

        <li>Milk</li>

        <li>Coffee</li>

    </ol>

</body>

</html>
```

Output:

Ordered List with Roman Numbers

- I. Bread
- II. Eggs
- III. Milk
- IV. Coffee

Ordered List with uppercase roman numbers

Example 5: Type="i", this list of items will be numbered with lowercase roman numbers.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <h2>Ordered List with Lowercase Roman Numbers</h2>

    <ol type="i">

        <li>Bread</li>

        <li>Eggs</li>

        <li>Milk</li>

        <li>Coffee</li>

    </ol>

</body>

</html>
```

Output:

Ordered List with Lowercase Roman Numbers

- i. Bread
- ii. Eggs
- iii. Milk
- iv. Coffee

Ordered List with lowercase roman numbers

Example 6: Nested ordered list, a nested ordered list is a list that has a list inside another list.

HTML

```
<!DOCTYPE html>

<html>

<body>

    <h1>GeeksforGeeks</h1>

    <h2>Nested Ordered List</h2>

    <ol>

        <li>Coffee</li>

        <li> Tea

            <ol>

                <li>Black tea</li>

                <li>Green tea</li>

            </ol>

        </li>

        <li>Milk</li>

    </ol>

</body>

</html>
```

Output:

GeeksforGeeks

Nested Ordered List

1. Coffee
2. Tea
 1. Black tea
 2. Green tea
3. Milk

Nested Ordered List

There is another attribute that is specifically defined for a list item, which is used in with the “li” tag and that is the [value attribute](#). Below is a little description of the value attribute specifically used with the “li” tag. Though it is used with various other HTML elements.

Value attribute:

The value attribute may be used on an individual element within an ordered list to change its value within the list. You define the value of a list item and the number of any list item appearing after it will be recalculated accordingly.

Example: This example illustrates the use of the “value attribute” used on the element.

HTML

```
<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

</head>

<body>

<h2>Welcome To GFG</h2>

<ol>
```

```
<li>Item One</li>

<li value="10">Item Two</li>

<li>Item Three</li>

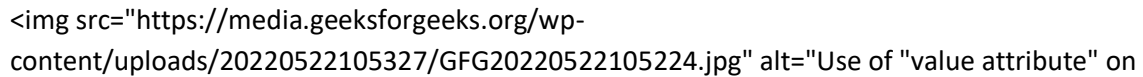
<li>Item Four</li>

</ol>

</body>

</html>
```

Output:

The screenshot shows a web browser displaying the output of the HTML code. It shows a list with four items: 'Item One', 'Item Two', 'Item Three', and 'Item Four'. The 'Item Two' has a value attribute set to '10'.

- " srcset="https://media.geeksforgeeks.org/wp-content/uploads/20220522105327/GFG20220522105224.jpg, " sizes="100vw" width="237">Value attribute

HTML Description List: 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. Please refer to the [How to add description list of an element using HTML?](#) article for further details.

Syntax:

```
<dl> Contents... </dl>
```

Example: This example describes the HTML Description List.

HTML

```
<!DOCTYPE html>

<html>

<body>
```

```
<h2>A Description List</h2>

<dl>

    <dt>Coffee</dt>

    <dd>- 500 gms</dd>

    <dt>Milk</dt>

    <dd>- 1 ltr Tetra Pack</dd>

</dl>

</body>

</html>
```

Output:

A Description List

Coffee
- 500 gms

Milk
- 1 ltr Tetra Pack

Description List

Supported Browser:

- Google Chrome 94.0 & above
- Microsoft Edge 93.0
- Firefox 92.0 & above
- Opera 78.0
- Safari 14.1
- IE 11.0

HTML | face Attribute

The **HTML face Attribute** is used to specify *the font family of the text inside element.*

Syntax:

```
<font face="font_family">
```

Attribute Values: It contains single value **font_family** which is used to specify the font family. Several font family can be used by separating comma.

Note: The face attribute is not supported by HTML5.

Example:

```
<!DOCTYPE html>

<html>

  <head>

    <title>

      HTML font face Attribute

    </title>

  </head>

  <body>

    <font size="6" face="verdana">

      GeeksforGeeks!    </font>

    <br>

    <font size="6" face="arial">

      GeeksforGeeks!

    </font>    <br>

    <font size="6">

      GeeksforGeeks!

    </font></body>

</html>
```

Output:

GeeksforGeeks!
GeeksforGeeks!
GeeksforGeeks!

Supported Browsers: The browser supported by **HTML face attribute** are listed below:

- Google Chrome
- Internet Explorer
- Firefox
- Safari
- Opera

Example:

```
<!DOCTYPE html>

<html>

<head>

    <title>

        HTML font face Attribute

    </title>

</head>

<body>

    <font size="6" face="verdana">

        GeeksforGeeks!

    </font>
```

```
<br>

<font size="6" face="arial">

    GeeksforGeeks!

</font>

<br>

<font size="6">

    GeeksforGeeks!

</font>

</body>

</html>
```

Output:

GeeksforGeeks!
GeeksforGeeks!
GeeksforGeeks!

Supported Browsers: The browser supported by **HTML face attribute** are listed below:

- Google Chrome
- Internet Explorer
- Firefox
- Safari
- Opera

HTML | <col> bgcolor Attribute

The **HTML <col> bgcolor attribute** is used to specify the background color of a column element. It is not supported by HTML 5.

Syntax:

```
<col bgcolor= "color_name | hex_number | rgb_number">
```

Attribute Values:

- **color_name:** It sets the text color by using the color name. For example *“red”*.
- **hex_number:** It sets the text color by using the color hex code. For example *“#0000ff”*.
- **rgb_number:** It sets the text color by using the rgb code. For example: *“RGB(0, 153, 0)”*.

Example:

```
<!DOCTYPE html>

<html>

<head>

    <title>

        HTML col bgcolor Attribute

    </title>

</head>

<body>

    <h1>GeeksforGeeks</h1>

    <h2>HTML col bgcolor Attribute</h2>

    <table border="1">
```

```
<colgroup>

    <col span="2" bgcolor="green">

        <col bgcolor="blue">

</colgroup>

<tr>

    <th>Name</th>

    <th>Branch</th>

    <th>Expenses</th>

</tr>


<tr>

    <td>BITTU</td>

    <td>CSE</td>

    <td>2500.00</td>

</tr>


<tr>

    <td>RAKESH</td>

    <td>ECE</td>

    <td>1400.00</td>

</tr>

</table>
```



```
</body>
```

```
</html>
```

Output:

GeeksforGeeks

HTML col bgcolor Attribute

Name	Branch	Expenses
BITTU	CSE	2500.00
RAKESH	ECE	1400.00

Supported Browsers: The browser supported by **HTML <col> bgcolor attribute** are listed below:

- Google Chrome
- Internet Explorer
- Firefox
- Safari
- Opera

HTML align Attribute

• [Read](#)

• [Discuss](#)

• [Courses](#)

•

HTML **align Attribute** in HTML is used to specify the alignment of the text content of The Element. this attribute is used in all elements. The Align attribute can also be set using the CSS property “text-align: ” or in “vertical-align: “.

Supported Tags:

- [<applet>](#)
- [<caption>](#)
- [<col>](#)
- [<colgroup>](#)
- [<hr>](#)
- [<iframe>](#)

- [](#)
- [<legend>](#)
- [<table>](#)
- [<tbody>](#)
- [<td>](#)
- [<tfoot>](#)
- [<th>](#)
- [<thead>](#)
- [<tr>](#)

Syntax:

```
<element_name align="left | right | center | justify">
```

Attribute Values:

- **left:** It sets the text left-align.
- **right:** It sets the text right-align.
- **center:** It sets the text center-align.
- **justify:** It stretches the text of a paragraph to set the width of all lines equal.

Example 1: This example shows the use of the above approach.

- html

```
<!DOCTYPE html>

<html>

<head>

    <title>

        HTML p align Attribute

    </title>

</head>

<body>

    <h1>GeeksforGeeks</h1>
```

```
<h2>HTML p align Attribute</h2>
```

```
<p align="left">
```

```
    Left align content
```

```
</p>
```

```
<p align="center">
```

```
    center align content
```

```
</p>
```

```
<p align="right">
```

```
    Right align content
```

```
</p>
```

```
</body>
```

```
</html>
```

Output:

GeeksforGeeks

HTML align Attribute

Left align content

center align content

Right align content

Example 2: This example shows the use of the above approach.

- **html**

```
<!DOCTYPE html>

<html>

<head>

    <title>gfg</title>

</head>

<body>

    <h1>GeeksForGeeks</h1>

    <h2>

        div align Attribute

    </h2>

    <div align="center">

        div align="center"
```

```
</div>

<div align="left">

    div align="left"

</div>

<div align="right">

    div align="right"

</div>

<div align="justify">

    div align="justify"

</div>

</body>

</html>
```

Output:

GeeksForGeeks

div align Attribute

div align="left"

div align="justify"

div align="center"

div align="right"

HTML align attribute

Supported Browsers:

- Google Chrome
- Internet Explorer
- Firefox
- Apple Safari
- Opera

Design Text with HTML

[Download Article](#)

1

Surround each section that will have changed alignment with a "div". That means, you need to add "div" inside the "less than" and "greater than" symbols (<>) before the first HTML tag that will have its alignment changed, and add "/div" inside these symbols after the last HTML tag that will have its alignment changed.

2

Determine how you need to change the alignment of the text in that "div".

3

If you need left-align the text, change the "div" tag so that the following text is inside the "<>" symbols: `div style='text-align:left'`.

- If you need to right-align the text, change the "div" tag to `div style='text-align:right'` within the "<>" symbols.
- If you need center-align the text, change the "div" tag to `div style='text-align:center'` within the "<>" symbols.
- If you need to justify the text, change the "div" tag to `div style='text-align:justify'` within the "<>" symbols.

4

Save your changes.

5

Verify your content's appearance to make sure it worked.

- If it didn't work, then the website has specific coding in its style sheet that overrides your "div". Override the site style-sheet by adding the appropriate version of `style='text-align:right'` inside the opening tag of each element to have its alignment changed. For example, a "p" tag would become `p style='text-align:right'` within the "<>" symbols.

6

Enjoy seeing your text display exactly how you wanted.

Align Images with HTML

[Download Article](#)

1

Find the HTML code for the image you want to align.

2

Edit the "img" tag to add the appropriate "float" property to it.

- If you need the image to hang to the left, add "style='float:left'" to the tag, as in "img style='float:right'" within the "<>" symbols.
- If you need the image to hang to the right, add "style='float:right'" to the tag, as in "img style='float:right'" within the "<>" symbols.
- If you need the image to hang in the center, the code gets a bit more complicated. There is not "float:center" property, so you have to add the work-around "style='align:center;text-align:center'" to the tag, as in "img style='align:center;text-align:center'" again within the "<>" symbols.

3

Save your code.

4

Enjoy your aligned images.

HTML <hr> Tag

The <hr> tag in HTML stands for horizontal rule and is used to insert a horizontal rule or a thematic break in an HTML page to divide or separate document sections. The <hr> tag is an empty tag, and it does not require an end tag.

Tag Attributes: The table given below describe the <hr> tag attributes. **These attributes are not supported in HTML5:**

Attribute	Value	Description
align	left center right	Used to specify the alignment of the horizontal rule.
noshade	noshade	Used to specify the bar without shading effect.
size	pixels	Used to specify the height of the horizontal rule.
width	pixels	Used to specify the width of the horizontal rule.

Syntax :

<hr> ...

Below programs illustrate the <hr> tag in HTML:

Example 1:

HTML

```
<!DOCTYPE html>

<html>

<body>

<p>There is a horizontal rule below this paragraph.</p>

<!--HTML hr tag is used here-->

    <hr>
```

```
<p>This is a horizontal rule above this paragraph.</p>
```

```
</body>
```

```
</html>
```

Output:

There is a horizontal rule below this paragraph.

This is a horizontal rule above this paragraph.

Example 2(hr tag with attributes):

HTML

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Normal horizontal line.</p>
```

```
<!--HTML hr tag is used here-->
```

```
<hr>
```

```
<p>Horizontal line with height of 30 pixels</p>
```

```
<hr size="30">
```

```
<p>Horizontal line with height of 30 pixels  
and noshade.</p>
```

```
<hr size="30" noshade>
```

```
</body>
```

```
</html>
```

Output:

image widget

Supported Browsers:

- Google Chrome 1
- Edge 12
- Internet Explorer 5.5
- Firefox 1
- Opera 12.1
- Safari 3

HTML is the foundation of webpages, is used for webpage development by structuring websites and web apps. You can learn HTML from the ground up by following this [HTML Tutorial](#) and [HTML Examples](#).

HTML Tables

HTML tables allow web developers to arrange data into rows and columns.

Define an HTML Table

A table in HTML consists of table cells inside rows and columns.

Example

A simple HTML table:

```
<table>
  <tr>
    <th>Company</th>
    <th>Contact</th>
```

```
    <th>Country</th>
</tr>
<tr>
    <td>Alfreds Futterkiste</td>
    <td>Maria Anders</td>
    <td>Germany</td>
</tr>
<tr>
    <td>Centro comercial Moctezuma</td>
    <td>Francisco Chang</td>
    <td>Mexico</td>
</tr>
</table>
```

Table Cells

Each table cell is defined by a `<td>` and a `</td>` tag.

`td` stands for table data.

Everything between `<td>` and `</td>` are the content of the table cell.

Example

```
<table>
  <tr>
    <td>Emil</td>
    <td>Tobias</td>
    <td>Linus</td>
  </tr>
</table>
```

Note: A table cell can contain all sorts of HTML elements: text, images, lists, links, other tables, etc.

Table Rows

Each table row starts with a `<tr>` and ends with a `</tr>` tag.

`tr` stands for table row.

Example

```
<table>
  <tr>
    <td>Emil</td>
    <td>Tobias</td>
    <td>Linus</td>
  </tr>
  <tr>
    <td>16</td>
    <td>14</td>
    <td>10</td>
  </tr>
</table>
```

You can have as many rows as you like in a table; just make sure that the number of cells are the same in each row.

Note: There are times when a row can have less or more cells than another. You will learn about that in a later chapter.

Table Headers

Sometimes you want your cells to be table header cells. In those cases use the `<th>` tag instead of the `<td>` tag:

`th` stands for table header.

Example

Let the first row be table header cells:

```
<table>
  <tr>
    <th>Person 1</th>
    <th>Person 2</th>
    <th>Person 3</th>
  </tr>
  <tr>
    <td>Emil</td>
    <td>Tobias</td>
    <td>Linus</td>
```

```
</tr>
<tr>
  <td>16</td>
  <td>14</td>
  <td>10</td>
</tr>
</table>
```

By default, the text in `<th>` elements are bold and centered, but you can change that with CSS.

HTML Table Tags

Tag	Description
<code><table></code>	Defines a table
<code><th></code>	Defines a header cell in a table
<code><tr></code>	Defines a row in a table
<code><td></code>	Defines a cell in a table
<code><caption></code>	Defines a table caption
<code><colgroup></code>	Specifies a group of one or more columns in a table for formatting

<code><col></code>	Specifies column properties for each column within a <code><colgroup></code> element
--	--

<code><thead></code>	Groups the header content in a table
--	--------------------------------------

<code><tbody></code>	Groups the body content in a table
--	------------------------------------

<code><tfoot></code>	Groups the footer content in a table
--	--------------------------------------

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.

Example

Following is the example to create three horizontal frames –

[Live Demo](#)

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Frames</title>
  </head>

  <frameset rows = "10%,80%,10%">
    <frame name = "top" src = "/html/top_frame.htm"
  />
    <frame name = "main" src = "/html/main_frame.htm"
  />
    <frame name = "bottom" src =
"/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support
frames.</body>
    </noframes>

  </frameset>

</html>
```

This will produce the following result –

Example

Let's put the above example as follows, here we replaced rows attribute by cols and changed their width. This will create all the three frames vertically –

[Live Demo](#)

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Frames</title>
  </head>

  <frameset cols = "25%,50%,25%">
    <frame name = "left" src = "/html/top_frame.htm"
  />
    <frame name = "center" src =
"/html/main_frame.htm" />
    <frame name = "right" src =
"/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support
frames.</body>
    </noframes>
  </frameset>

</html>
```

This will produce the following result –

The <frameset> Tag Attributes

Following are important attributes of the <frameset> tag –

Sr.No	Attribute & Description
	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 <i>cols</i> = "100, 500, 100".
1	A percentage of the browser window. For example, to create three vertical frames, use <i>cols</i> = "10%, 80%, 10%". Using a wildcard symbol. For example, to create three vertical frames, use <i>cols</i> = "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 <i>cols</i> = "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.

rows

- 2 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.

border

- 3 This attribute specifies the width of the border of each frame in pixels. For example, border = "5". A value of zero means no border.

frameborder

- 4 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.

framespacing

- 5 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
	src
1	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.
	name
2	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.
	frameborder
3	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"`.

marginheight

- 5 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"`.

noresize

- 6 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"`.

scrolling

- 7 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.

longdesc

- 8 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"`

Browser Support for Frames

If a user is using any old browser or any browser, which does not support frames then `<noframes>` element should be displayed to the user.

So you must place a `<body>` element inside the `<noframes>` element because the `<frameset>` element is supposed to replace the `<body>` element, but if a browser does not understand `<frameset>` element then it should understand what is inside the `<body>` element which is contained in a `<noframes>` element.

You can put some nice message for your user having old browsers. For example, *Sorry!! your browser does not support frames.* as shown in the above example.

Frame's name and target attributes

One of the most popular uses of frames is to place navigation bars in one frame and then load main pages into a separate frame.

Let's see following example where a test.htm file has following code –

[Live Demo](#)

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Target Frames</title>
  </head>

  <frameset cols = "200, *">
    <frame src = "/html/menu.htm" name = "menu_page"
  />
    <frame src = "/html/main.htm" name = "main_page"
  />

  <noframes>
    <body>Your browser does not support
frames.</body>
  </noframes>
</frameset>

</html>
```

Here, we have created two columns to fill with two frames. The first frame is 200 pixels wide and will contain the navigation menu bar implemented by **menu.htm** file. The second column fills in remaining space and will contain the main part of the page and it is implemented by **main.htm** file. For all the three links available in menu bar, we have mentioned target frame as **main_page**, so whenever you click any of the links in menu bar, available link will open in main page.

Following is the content of menu.htm file

```
<!DOCTYPE html>
<html>

  <body bgcolor = "#4a7d49">
    <a href = "http://www.google.com" target =
"main_page">Google</a>
```

```
<br />
<br />

    <a href = "http://www.microsoft.com" target =
"main_page">Microsoft</a>
    <br />
    <br />

    <a href = "http://news.bbc.co.uk" target =
"main_page">BBC News</a>
</body>

</html>
```

Following is the content of main.htm file –

[Live Demo](#)

```
<!DOCTYPE html>
<html>

    <body bgcolor = "#b5dcb3">
        <h3>This is main page and content from any link
will be displayed here.</h3>
        <p>So now click any link and see the result.</p>
    </body>

</html>
```

When we load **test.htm** file, it produces following result –

Now you can try to click links available in the left panel and see the result. The *targetattribute* can also take one of the following values –

Sr. No	Option & Description
1	_self Loads the page into the current frame.
2	_blank Loads a page into a new browser window. Opening a new window.

- 3 **_parent**
Loads the page into the parent window, which in the case of a single frameset is the main browser window.
 - 4 **_top**
Loads the page into the browser window, replacing any current frames.
- targetframe**
Loads the page into a named targetframe.

UNIT 2

HTML Forms

[Previous](#)[Next](#)

An HTML form is used to collect user input. The user input is most often sent to a server for processing.

Example

First name:

Last name:

[Try it Yourself »](#)

The <form> Element

The HTML `<form>` element is used to create an HTML form for user input:

`<form>`

.

form elements

.

`</form>`

The `<form>` element is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc.

All the different form elements are covered in this chapter: [HTML Form Elements](#).

The `<input>` Element

The HTML `<input>` element is the most used form element.

An `<input>` element can be displayed in many ways, depending on the `type` attribute.

Here are some examples:

Type	Description
<code><input type="text"></code>	Displays a single-line text input field
<code><input type="radio"></code>	Displays a radio button (for selecting c
<code><input type="checkbox"></code>	Displays a checkbox (for selecting zero
<code><input type="submit"></code>	Displays a submit button (for submittir

`<input type="button">`

Displays a clickable button

All the different input types are covered in this chapter: [HTML Input Types](#).

ADVERTISEMENT

Text Fields

The `<input type="text">` defines a single-line input field for text input.

Example

A form with input fields for text:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

[Try it Yourself »](#)

This is how the HTML code above will be displayed in a browser:

First name:

Last name:

Note: The form itself is not visible. Also note that the default width of an input field is 20 characters.

The <label> Element

Notice the use of the `<label>` element in the example above.

The `<label>` tag defines a label for many form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focuses on the input element.

The `<label>` element also helps users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the `<label>` element, it toggles the radio button/checkbox.

The `for` attribute of the `<label>` tag should be equal to the `id` attribute of the `<input>` element to bind them together.

Radio Buttons

The `<input type="radio">` defines a radio button.

Radio buttons let a user select ONE of a limited number of choices.

Example

A form with radio buttons:

```
<p>Choose your favorite Web language:</p>
```

```
<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

Choose your favorite Web language:

- ☐ HTML
- ☐ CSS
- ☐ JavaScript

Checkboxes

The `<input type="checkbox">` defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

Example

A form with checkboxes:

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike"
">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car"
>
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat"
">
  <label for="vehicle3"> I have a boat</label>
</form>
```

[Try it Yourself »](#)

This is how the HTML code above will be displayed in a browser:

- ☐ I have a bike
- ☐ I have a car
- ☐ I have a boat

The Submit Button

The `<input type="submit">` defines a button for submitting the form data to a form-handler.

The form-handler is typically a file on the server with a script for processing input data.

The form-handler is specified in the form's `action` attribute.

Example

A form with a submit button:

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

First name:

Last name:

The Name Attribute for <input>

Notice that each input field must have a `name` attribute to be submitted.

If the `name` attribute is omitted, the value of the input field will not be sent at all.

Example

This example will not submit the value of the "First name" input field:

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" value="John"><br><br>
  <input type="submit" value="Submit">
</form>
```

[Try it Yourself »](#)

HTML Exercises

Exercise:

In the form below, add an input field with the type "button" and the value "OK".

```
form>

</form>
```

[Submit Answer »](#)

[Start the Exercise](#)

HTML Form Attributes

[Previous](#) [Next](#)

This chapter describes the different attributes for the HTML `<form>` element.

The Action Attribute

The `action` attribute defines the action to be performed when the form is submitted.

Usually, the form data is sent to a file on the server when the user clicks on the submit button.

In the example below, the form data is sent to a file called "action_page.php". This file contains a server-side script that handles the form data:

Example

On submit, send form data to "action_page.php":

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

[Try it Yourself »](#)

Tip: If the `action` attribute is omitted, the action is set to the current page.

The Target Attribute

The `target` attribute specifies where to display the response that is received after submitting the form.

The `target` attribute can have one of the following values:

Value	Description
<code>_blank</code>	The response is displayed in a new window or tab
<code>_self</code>	The response is displayed in the current window
<code>_parent</code>	The response is displayed in the parent frame
<code>_top</code>	The response is displayed in the full body of the w
<code>framename</code>	The response is displayed in a named iframe

The default value is `_self` which means that the response will open in the current window.

Example

Here, the submitted result will open in a new browser tab:

```
<form action="/action_page.php" target="_blank">
```

[Try it Yourself »](#)

The Method Attribute

The `method` attribute specifies the HTTP method to be used when submitting the form data.

The form-data can be sent as URL variables (with `method="get"`) or as HTTP post transaction (with `method="post"`).

The default HTTP method when submitting form data is GET.

Example

This example uses the GET method when submitting the form data:

```
<form action="/action_page.php" method="get">
```

[Try it Yourself »](#)

Example

This example uses the POST method when submitting the form data:

```
<form action="/action_page.php" method="post">
```

[Try it Yourself »](#)

Notes on GET:

- Appends the form data to the URL, in name/value pairs
- NEVER use GET to send sensitive data! (the submitted form data is visible in the URL!)
- The length of a URL is limited (2048 characters)
- Useful for form submissions where a user wants to bookmark the result
- GET is good for non-secure data, like query strings in Google

Notes on POST:

- Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL)
- POST has no size limitations, and can be used to send large amounts of data.
- Form submissions with POST cannot be bookmarked

Tip: Always use POST if the form data contains sensitive or personal information!

The Autocomplete Attribute

The `autocomplete` attribute specifies whether a form should have autocomplete on or off.

When autocomplete is on, the browser automatically complete values based on values that the user has entered before.

Example

A form with autocomplete on:

```
<form action="/action_page.php" autocomplete="on">
```

[Try it Yourself »](#)

The Novalidate Attribute

The `novalidate` attribute is a boolean attribute.

When present, it specifies that the form-data (input) should not be validated when submitted.

Example

A form with a novalidate attribute:

```
<form action="/action_page.php" novalidate>
```

[Try it Yourself »](#)

HTML Exercises

Exercise:

Add a submit button, and specify that the form should go to 'action_page.php'.

```
form <input type="submit" value="Submit" />="/action_page.php">  
name: <input type="text" name="name">  
<input type="submit" value="Submit" />  
</form>
```

Submit Answer »

[Start the Exercise](#)

List of All <form> Attributes

Attribute	Description
accept-charset	Specifies the character encodings used for form submission
action	Specifies where to send the form-data when a form is submitted
autocomplete	Specifies whether a form should have autocomplete on or off
enctype	Specifies how the form-data should be encoded when submitting it to the server
method	Specifies the HTTP method to use when sending form-data

name	Specifies the name of the form
novalidate	Specifies that the form should not be validated when submitted
rel	Specifies the relationship between a linked resource and the current document
target	Specifies where to display the response that is received after submitting the form

HTML Form Elements

[Previous](#)[Next](#)

This chapter describes all the different HTML form elements.

The HTML `<form>` Elements

The HTML `<form>` element can contain one or more of the following form elements:

- `<input>`
- `<label>`
- `<select>`
- `<textarea>`
- `<button>`
- `<fieldset>`
- `<legend>`
- `<datalist>`
- `<output>`
- `<option>`
- `<optgroup>`

The <input> Element

One of the most used form elements is the `<input>` element.

The `<input>` element can be displayed in several ways, depending on the `type` attribute.

Example

```
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname">
```

Try it Yourself »

All the different values of the `type` attribute are covered in the next chapter: [HTML Input Types](#).

The <label> Element

The `<label>` element defines a label for several form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.

The `<label>` element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the `<label>` element, it toggles the radio button/checkbox.

The `for` attribute of the `<label>` tag should be equal to the `id` attribute of the `<input>` element to bind them together.

The <select> Element

The `<select>` element defines a drop-down list:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

[Try it Yourself »](#)

The `<option>` element defines an option that can be selected.

By default, the first item in the drop-down list is selected.

To define a pre-selected option, add the `selected` attribute to the option:

Example

```
<option value="fiat" selected>Fiat</option>
```

[Try it Yourself »](#)

Visible Values:

Use the `size` attribute to specify the number of visible values:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="3">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

[Try it Yourself »](#)

Allow Multiple Selections:

Use the `multiple` attribute to allow the user to select more than one value:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="4" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

[Try it Yourself »](#)

The <textarea> Element

The <textarea> element defines a multi-line input field (a text area):

Example

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

[Try it Yourself »](#)

The **rows** attribute specifies the visible number of lines in a text area.

The **cols** attribute specifies the visible width of a text area.

This is how the HTML code above will be displayed in a browser:



You can also define the size of the text area by using CSS:

Example

```
<textarea name="message" style="width:200px; height:600px;">
The cat was playing in the garden.
</textarea>
```

[Try it Yourself »](#)

ADVERTISEMENT

The <button> Element

The `<button>` element defines a clickable button:

Example

```
<button type="button" onclick="alert('Hello World!')">Click
Me!</button>
```

[Try it Yourself »](#)

This is how the HTML code above will be displayed in a browser:

Click Me!

Note: Always specify the `type` attribute for the button element. Different browsers may use different default types for the button element.

The <fieldset> and <legend> Elements

The `<fieldset>` element is used to group related data in a form.

The `<legend>` element defines a caption for the `<fieldset>` element.

Example

```

<form action="/action_page.php">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname" value="John"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname" value="Doe"><br><br>
  >
  <input type="submit" value="Submit">
</fieldset>
</form>

```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

Personalia:First name:

Last name:

The <datalist> Element

The `<datalist>` element specifies a list of pre-defined options for an `<input>` element.

Users will see a drop-down list of the pre-defined options as they input data.

The `list` attribute of the `<input>` element, must refer to the `id` attribute of the `<datalist>` element.

Example

```

<form action="/action_page.php">
  <input list="browsers">
  <datalist id="browsers">
    <option value="Edge">

```



```
<option value="Firefox">
<option value="Chrome">
<option value="Opera">
<option value="Safari">
</datalist>
</form>
```

Try it Yourself »

The <output> Element

The <output> element represents the result of a calculation (like one performed by a script).

Example

Perform a calculation and show the result in an <output> element:

```
<form action="/action_page.php"
oninput="x.value=parseInt(a.value)+parseInt(b.value)">
  0
  <input type="range" id="a" name="a" value="50">
  100 +
  <input type="number" id="b" name="b" value="50">
  =
  <output name="x" for="a b"></output>
  <br><br>
  <input type="submit">
</form>
```

Try it Yourself »

HTML Exercises

Exercise:

In the form below, add an empty drop down list with the name "cars".

```
<form action="/action_page.php">  
<input type="text" value="Name" />  
<input type="text" value="Email" />  
</form>
```

Submit Answer »

[Start the Exercise](#)

HTML Form Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control
<textarea>	Defines a multiline input control (text area)
<label>	Defines a label for an <input> element
<fieldset>	Groups related elements in a form

<u><legend></u>	Defines a caption for a <fieldset> element
<u><select></u>	Defines a drop-down list
<u><optgroup></u>	Defines a group of related options in a drop-down list
<u><option></u>	Defines an option in a drop-down list
<u><button></u>	Defines a clickable button
<u><datalist></u>	Specifies a list of pre-defined options for input controls
<u><output></u>	Defines the result of a calculation

HTML Input Types

[❏ Previous](#)[Next ❏](#)

This chapter describes the different types for the HTML `<input>` element.

HTML Input Types

Here are the different input types you can use in HTML:

- `<input type="button">`
- `<input type="checkbox">`
- `<input type="color">`
- `<input type="date">`
- `<input type="datetime-local">`
- `<input type="email">`
- `<input type="file">`
- `<input type="hidden">`
- `<input type="image">`
- `<input type="month">`
- `<input type="number">`
- `<input type="password">`
- `<input type="radio">`
- `<input type="range">`
- `<input type="reset">`
- `<input type="search">`
- `<input type="submit">`
- `<input type="tel">`
- `<input type="text">`
- `<input type="time">`
- `<input type="url">`
- `<input type="week">`

Tip: The default value of the `type` attribute is "text".

Input Type Text

`<input type="text">` defines a **single-line text input field**:

Example

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

[Try it Yourself »](#)

This is how the HTML code above will be displayed in a browser:

First name:

Last name:

Input Type Password

`<input type="password">` defines a **password field**:

Example

```
<form>
  <label for="username">Username:</label><br>
  <input type="text" id="username" name="username"><br>
  <label for="pwd">Password:</label><br>
  <input type="password" id="pwd" name="pwd">
</form>
```

[Try it Yourself »](#)

This is how the HTML code above will be displayed in a browser:

Username:

Password:

The characters in a password field are masked (shown as asterisks or circles).

ADVERTISEMENT

Input Type Submit

`<input type="submit">` defines a button for **submitting** form data to a **form-handler**.

The form-handler is typically a server page with a script for processing input data.

The form-handler is specified in the form's `action` attribute:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

First name:

Last name:

If you omit the submit button's value attribute, the button will get a default text:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit">
</form>
```

Try it Yourself »

Input Type Reset

`<input type="reset">` defines a **reset button** that will reset all form values to their default values:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
  <input type="reset" value="Reset">
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

First name:

Last name:

If you change the input values and then click the "Reset" button, the form-data will be reset to the default values.

Input Type Radio

`<input type="radio">` defines a **radio button**.

Radio buttons let a user select ONLY ONE of a limited number of choices:

Example

<p>Choose your favorite Web language:</p>

```
<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

- ☐ HTML
- ☐ CSS
- ☐ JavaScript

Input Type Checkbox

<input type="checkbox"> defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

Example

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```


[Try it Yourself »](#)

This is how the HTML code above will be displayed in a browser:

- ☐ I have a bike
- ☐ I have a car
- ☐ I have a boat

Input Type Button

`<input type="button">` defines a **button**:

Example

```
<input type="button" onclick="alert('Hello World!')" value="Click Me!">
```

[Try it Yourself »](#)

This is how the HTML code above will be displayed in a browser:

Input Type Color

The `<input type="color">` is used for input fields that should contain a color.

Depending on browser support, a color picker can show up in the input field.

Example

```
<form>  
  <label for="favcolor">Select your favorite color:</label>  
  <input type="color" id="favcolor" name="favcolor">  
</form>
```

[Try it Yourself »](#)

Input Type Date

The `<input type="date">` is used for input fields that should contain a date.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="birthday">Birthday:</label>
  <input type="date" id="birthday" name="birthday">
</form>
```

[Try it Yourself »](#)

You can also use the `min` and `max` attributes to add restrictions to dates:

Example

```
<form>
  <label for="datemax">Enter a date before 1980-01-01:</label>
  <input type="date" id="datemax" name="datemax" max="1979-12-31"><br><br>
  <label for="datemin">Enter a date after 2000-01-01:</label>
  <input type="date" id="datemin" name="datemin" min="2000-01-02">
</form>
```

[Try it Yourself »](#)

Input Type Datetime-local

The `<input type="datetime-local">` specifies a date and time input field, with no time zone.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="birthdaytime">Birthday (date and time):</label>
  <input type="datetime-
local" id="birthdaytime" name="birthdaytime">
</form>
```

[Try it Yourself »](#)

Input Type Email

The `<input type="email">` is used for input fields that should contain an e-mail address.

Depending on browser support, the e-mail address can be automatically validated when submitted.

Some smartphones recognize the email type, and add ".com" to the keyboard to match email input.

Example

```
<form>
  <label for="email">Enter your email:</label>
  <input type="email" id="email" name="email">
</form>
```

[Try it Yourself »](#)

Input Type Image

The `<input type="image">` defines an image as a submit button.

The path to the image is specified in the `src` attribute.

Example

```
<form>
<input type="image" src="img_submit.gif" alt="Submit" width="48" height="48">
</form>
```

Try it Yourself »

Input Type File

The `<input type="file">` defines a file-select field and a "Browse" button for file uploads.

Example

```
<form>
  <label for="myfile">Select a file:</label>
  <input type="file" id="myfile" name="myfile">
</form>
```

Try it Yourself »

Input Type Hidden

The `<input type="hidden">` defines a hidden input field (not visible to a user).

A hidden field lets web developers include data that cannot be seen or modified by users when a form is submitted.

A hidden field often stores what database record that needs to be updated when the form is submitted.

Note: While the value is not displayed to the user in the page's content, it is visible (and can be edited) using any browser's developer tools or "View Source" functionality. Do not use hidden inputs as a form of security!

Example

```
<form>
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <input type="hidden" id="custId" name="custId" value="3487">
  <input type="submit" value="Submit">
</form>
```

[Try it Yourself »](#)

Input Type Month

The `<input type="month">` allows the user to select a month and year.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="bdaymonth">Birthday (month and year):</label>
  <input type="month" id="bdaymonth" name="bdaymonth">
</form>
```

[Try it Yourself »](#)

Input Type Number

The `<input type="number">` defines a **numeric** input field.

You can also set restrictions on what numbers are accepted.

The following example displays a numeric input field, where you can enter a value from 1 to 5:

Example

```
<form>
  <label for="quantity">Quantity (between 1 and 5):</label>
```

```
<input type="number" id="quantity" name="quantity" min="1" max="5">
</form>
```

[Try it Yourself »](#)

Input Restrictions

Here is a list of some common input restrictions:

Attribute	Description
checked	Specifies that an input field should be pre-selected (type="checkbox" or type="radio")
disabled	Specifies that an input field should be disabled
max	Specifies the maximum value for an input field
maxlength	Specifies the maximum number of character for an input field
min	Specifies the minimum value for an input field
pattern	Specifies a regular expression to check the input value
readonly	Specifies that an input field is read only (cannot be edited)

required	Specifies that an input field is required (must be filled out)
size	Specifies the width (in characters) of an input field
step	Specifies the legal number intervals for an input field
value	Specifies the default value for an input field

You will learn more about input restrictions in the next chapter.

The following example displays a numeric input field, where you can enter a value from 0 to 100, in steps of 10. The default value is 30:

Example

```
<form>
  <label for="quantity">Quantity:</label>
  <input type="number" id="quantity" name="quantity" min="0" max="
100" step="10" value="30">
</form>
```

[Try it Yourself »](#)

Input Type Range

The `<input type="range">` defines a control for entering a number whose exact value is not important (like a slider control). Default range is 0 to 100. However, you can set restrictions on what numbers are accepted with the `min`, `max`, and `step` attributes:

Example

```
<form>
```

```
<label for="vol">Volume (between 0 and 50):</label>
<input type="range" id="vol" name="vol" min="0" max="50">
</form>
```

[Try it Yourself »](#)

Input Type Search

The `<input type="search">` is used for search fields (a search field behaves like a regular text field).

Example

```
<form>
  <label for="gsearch">Search Google:</label>
  <input type="search" id="gsearch" name="gsearch">
</form>
```

[Try it Yourself »](#)

Input Type Tel

The `<input type="tel">` is used for input fields that should contain a telephone number.

Example

```
<form>
  <label for="phone">Enter your phone number:</label>
  <input type="tel" id="phone" name="phone" pattern="[0-9]{3}-[0-9]{2}-[0-9]{3}">
</form>
```

[Try it Yourself »](#)

Input Type Time

The `<input type="time">` allows the user to select a time (no time zone).

Depending on browser support, a time picker can show up in the input field.

Example

```
<form>
  <label for="appt">Select a time:</label>
  <input type="time" id="appt" name="appt">
</form>
```

Try it Yourself »

Input Type Url

The `<input type="url">` is used for input fields that should contain a URL address.

Depending on browser support, the url field can be automatically validated when submitted.

Some smartphones recognize the url type, and adds ".com" to the keyboard to match url input.

Example

```
<form>
  <label for="homepage">Add your homepage:</label>
  <input type="url" id="homepage" name="homepage">
</form>
```

Try it Yourself »

Input Type Week

The `<input type="week">` allows the user to select a week and year.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="week">Select a week:</label>
  <input type="week" id="week" name="week">
</form>
```

[Try it Yourself »](#)

HTML Exercises

Exercise:

In the form below, add an input field for text, with the name "username" .

```
form action="/action_page.php">
  <input type="text" name="username">
</form>
```

[Submit Answer »](#)

[Start the Exercise](#)

HTML Input Type Attribute

Tag	Description
<code><input type=""></code>	Specifies the input type to display

HTML Input Attributes

[Previous](#)[Next](#)

This chapter describes the different attributes for the HTML `<input>` element.

The value Attribute

The input `value` attribute specifies an initial value for an input field:

Example

Input fields with initial (default) values:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe">
</form>
```

[Try it Yourself »](#)

The readonly Attribute

The input `readonly` attribute specifies that an input field is read-only.

A read-only input field cannot be modified (however, a user can tab to it, highlight it, and copy the text from it).

The value of a read-only input field will be sent when submitting the form!

Example

A read-only input field:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John" readonly>
</br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe">
</form>
```

[Try it Yourself »](#)

The disabled Attribute

The input `disabled` attribute specifies that an input field should be disabled.

A disabled input field is unusable and un-clickable.

The value of a disabled input field will not be sent when submitting the form!

Example

A disabled input field:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John" disabled>
</br>
  <label for="lname">Last name:</label><br>
```

```
<input type="text" id="lname" name="lname" value="Doe">
</form>
```

[Try it Yourself »](#)

ADVERTISEMENT

The size Attribute

The input `size` attribute specifies the visible width, in characters, of an input field.

The default value for `size` is 20.

Note: The `size` attribute works with the following input types: text, search, tel, url, email, and password.

Example

Set a width for an input field:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" size="50"><br>
  <label for="pin">PIN:</label><br>
  <input type="text" id="pin" name="pin" size="4">
</form>
```

[Try it Yourself »](#)

The maxlength Attribute

The input `maxlength` attribute specifies the maximum number of characters allowed in an input field.

Note: When a `maxlength` is set, the input field will not accept more than the specified number of characters. However, this attribute does not

provide any feedback. So, if you want to alert the user, you must write JavaScript code.

Example

Set a maximum length for an input field:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" size="50"><br>
  <label for="pin">PIN:</label><br>
  <input type="text" id="pin" name="pin" maxlength="4" size="4">
</form>
```

[Try it Yourself »](#)

The min and max Attributes

The input `min` and `max` attributes specify the minimum and maximum values for an input field.

The `min` and `max` attributes work with the following input types: number, range, date, datetime-local, month, time and week.

Tip: Use the max and min attributes together to create a range of legal values.

Example

Set a max date, a min date, and a range of legal values:

```
<form>
  <label for="datemax">Enter a date before 1980-01-01:</label>
  <input type="date" id="datemax" name="datemax" max="1979-12-31"><br><br>

  <label for="datemin">Enter a date after 2000-01-01:</label>
  <input type="date" id="datemin" name="datemin" min="2000-01-02"><br><br>

  <label for="quantity">Quantity (between 1 and 5):</label>
```

```
<input type="number" id="quantity" name="quantity" min="1" max="5">
</form>
```

[Try it Yourself »](#)

The multiple Attribute

The input `multiple` attribute specifies that the user is allowed to enter more than one value in an input field.

The `multiple` attribute works with the following input types: email, and file.

Example

A file upload field that accepts multiple values:

```
<form>
  <label for="files">Select files:</label>
  <input type="file" id="files" name="files" multiple>
</form>
```

[Try it Yourself »](#)

The pattern Attribute

The input `pattern` attribute specifies a regular expression that the input field's value is checked against, when the form is submitted.

The `pattern` attribute works with the following input types: text, date, search, url, tel, email, and password.

Tip: Use the global [title](#) attribute to describe the pattern to help the user.

Tip: Learn more about [regular expressions](#) in our JavaScript tutorial.

Example

An input field that can contain only three letters (no numbers or special characters):

```
<form>
  <label for="country_code">Country code:</label>
  <input type="text" id="country_code" name="country_code"
    pattern="[A-Za-z]{3}" title="Three letter country code">
</form>
```

[Try it Yourself »](#)

The placeholder Attribute

The input `placeholder` attribute specifies a short hint that describes the expected value of an input field (a sample value or a short description of the expected format).

The short hint is displayed in the input field before the user enters a value.

The `placeholder` attribute works with the following input types: text, search, url, tel, email, and password.

Example

An input field with a placeholder text:

```
<form>
  <label for="phone">Enter a phone number:</label>
  <input type="tel" id="phone" name="phone"
    placeholder="123-45-678"
    pattern="[0-9]{3}-[0-9]{2}-[0-9]{3}">
</form>
```

[Try it Yourself »](#)

The required Attribute

The input **required** attribute specifies that an input field must be filled out before submitting the form.

The **required** attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

Example

A required input field:

```
<form>
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" required>
</form>
```

[Try it Yourself »](#)

The step Attribute

The input **step** attribute specifies the legal number intervals for an input field.

Example: if `step="3"`, legal numbers could be -3, 0, 3, 6, etc.

Tip: This attribute can be used together with the max and min attributes to create a range of legal values.

The **step** attribute works with the following input types: number, range, date, datetime-local, month, time and week.

Example

An input field with a specified legal number intervals:

```
<form>
  <label for="points">Points:</label>
  <input type="number" id="points" name="points" step="3">
</form>
```

[Try it Yourself »](#)

Note: Input restrictions are not foolproof, and JavaScript provides many ways to add illegal input. To safely restrict input, it must also be checked by the receiver (the server)!

The autofocus Attribute

The input `autofocus` attribute specifies that an input field should automatically get focus when the page loads.

Example

Let the "First name" input field automatically get focus when the page loads:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" autofocus><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

[Try it Yourself »](#)

The height and width Attributes

The input `height` and `width` attributes specify the height and width of an `<input type="image">` element.

Tip: Always specify both the height and width attributes for images. If height and width are set, the space required for the image is reserved when the page is loaded. Without these attributes, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

Example

Define an image as the submit button, with height and width attributes:

```
<form>
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="image" src="img_submit.gif" alt="Submit" width="48"
    height="48">
</form>
```

[Try it Yourself »](#)

The list Attribute

The input `list` attribute refers to a `<datalist>` element that contains pre-defined options for an `<input>` element.

Example

An `<input>` element with pre-defined values in a `<datalist>`:

```
<form>
  <input list="browsers">
  <datalist id="browsers">
    <option value="Edge">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
  </datalist>
</form>
```

[Try it Yourself »](#)

The autocomplete Attribute

The input `autocomplete` attribute specifies whether a form or an input field should have autocomplete on or off.

Autocomplete allows the browser to predict the value. When a user starts to type in a field, the browser should display options to fill in the field, based on earlier typed values.

The `autocomplete` attribute works with `<form>` and the following `<input>` types: text, search, url, tel, email, password, datepickers, range, and color.

Example

An HTML form with autocomplete on, and off for one input field:

```
<form action="/action_page.php" autocomplete="on">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" autocomplete="off"><br><br>
  <input type="submit" value="Submit">
</form>
```

[Try it Yourself »](#)

Tip: In some browsers you may need to activate an autocomplete function for this to work (Look under "Preferences" in the browser's menu).

HTML Exercises

Exercise:

1. In the input field below, add placeholder that says "Your name here".

```
<form action="/action_page.php">
  <input type="text" <input type="text" >
```

```
/form>
```

Submit Answer »

[Start the Exercise](#)

HTML Form and Input Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control

HTML Input form* Attributes

[Previous](#)[Next](#)

This chapter describes the different **form*** attributes for the HTML **<input>** element.

The form Attribute

The input **form** attribute specifies the form the **<input>** element belongs to.

The value of this attribute must be equal to the id attribute of the

<form> element it belongs to.

Example

An input field located outside of the HTML form (but still a part of the form):

```
<form action="/action_page.php" id="form1">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <input type="submit" value="Submit">
</form>

<label for="lname">Last name:</label>
<input type="text" id="lname" name="lname" form="form1">
```

Try it Yourself »

The formaction Attribute

The input `formaction` attribute specifies the URL of the file that will process the input when the form is submitted.

Note: This attribute overrides the `action` attribute of the `<form>` element.

The `formaction` attribute works with the following input types: submit and image.

Example

An HTML form with two submit buttons, with different actions:

```
<form action="/action_page.php">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
  <input type="submit" formaction="/action_page2.php" value="Submit as Admin">
</form>
```

Try it Yourself »

The formenctype Attribute

The input `formenctype` attribute specifies how the form-data should be encoded when submitted (only for forms with `method="post"`).

Note: This attribute overrides the `enctype` attribute of the `<form>` element.

The `formenctype` attribute works with the following input types: submit and image.

Example

A form with two submit buttons. The first sends the form-data with default encoding, the second sends the form-data encoded as "multipart/form-data":

```
<form action="/action_page_binary.asp" method="post">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <input type="submit" value="Submit">
  <input type="submit" formenctype="multipart/form-data"
    value="Submit as Multipart/form-data">
</form>
```

Try it Yourself »

ADVERTISEMENT

The formmethod Attribute

The input `formmethod` attribute defines the HTTP method for sending form-data to the action URL.

Note: This attribute overrides the `method` attribute of

the `<form>` element.

The `formmethod` attribute works with the following input types: submit and image.

The form-data can be sent as URL variables (method="get") or as an HTTP post transaction (method="post").

Notes on the "get" method:

- This method appends the form-data to the URL in name/value pairs
- This method is useful for form submissions where a user want to bookmark the result
- There is a limit to how much data you can place in a URL (varies between browsers), therefore, you cannot be sure that all of the form-data will be correctly transferred
- Never use the "get" method to pass sensitive information! (password or other sensitive information will be visible in the browser's address bar)

Notes on the "post" method:

- This method sends the form-data as an HTTP post transaction
- Form submissions with the "post" method cannot be bookmarked
- The "post" method is more robust and secure than "get", and "post" does not have size limitations

Example

A form with two submit buttons. The first sends the form-data with method="get". The second sends the form-data with method="post":

```
<form action="/action_page.php" method="get">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit using GET">
  <input type="submit" formmethod="post" value="Submit using
POST">
</form>
```

[Try it Yourself »](#)

The formtarget Attribute

The input `formtarget` attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.

Note: This attribute overrides the target attribute of the `<form>` element.

The `formtarget` attribute works with the following input types: submit and image.

Example

A form with two submit buttons, with different target windows:

```
<form action="/action_page.php">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
  <input type="submit" formtarget="_blank" value="Submit to a new
window/tab">
</form>
```

[Try it Yourself »](#)

The formnovalidate Attribute

The input `formnovalidate` attribute specifies that an `<input>` element should not be validated when submitted.

Note: This attribute overrides the novalidate attribute of the `<form>` element.

The `formnovalidate` attribute works with the following input types: submit.

Example

A form with two submit buttons (with and without validation):

```
<form action="/action_page.php">
  <label for="email">Enter your email:</label>
  <input type="email" id="email" name="email"><br><br>
  <input type="submit" value="Submit">
  <input type="submit" formnovalidate="formnovalidate"
    value="Submit without validation">
</form>
```

[Try it Yourself »](#)

The novalidate Attribute

The `novalidate` attribute is a `<form>` attribute.

When present, `novalidate` specifies that all of the form-data should not be validated when submitted.

Example

Specify that no form-data should be validated on submit:

```
<form action="/action_page.php" novalidate>
  <label for="email">Enter your email:</label>
  <input type="email" id="email" name="email"><br><br>
  <input type="submit" value="Submit">
</form>
```

[Try it Yourself »](#)

HTML Form and Input Elements

Tag	Description
-----	-------------

[`<form>`](#)

Defines an HTML form for user input

[`<input>`](#)

Defines an input control

HTML Graphics

Google Maps

[Google Maps](#) lets you embed maps in HTML.

Google Maps Tutorial

Google Maps API

This tutorial is about the Google Maps API (Application **P**rogramming **I**nterface).

An API is a set of methods and tools that can be used for building software applications.

Google Maps in HTML

This example creates a Google Map in HTML:

Example

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

```
<h1>My First Google Map</h1>
```

```
<div id="googleMap" style="width:100%;height:400px;"></div>

<script>
function myMap() {
var mapProp= {
  center:new google.maps.LatLng(51.508742,-0.120850),
  zoom:5,
};
var map
= new google.maps.Map(document.getElementById("googleMap"),mapProp);
}
</script>

<script src="https://maps.googleapis.com/maps/api/js?key=YOUR_KEY&callb
ack=myMap"></script>

</body>
</html>
```

SVG

The [HTML <svg> element](#) allows vector based graphics in HTML:

SVG Tutorial

SVG stands for Scalable Vector Graphics.

SVG defines vector-based graphics in XML format.

Examples in Each Chapter

With our "Try it Yourself" editor, you can edit the SVG, and click on a button to view the result.

SVG Example

```
<html>
<body>

<h1>My first SVG</h1>

<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-
width="4" fill="yellow" />
</svg>

</body>
</html>
```

Try it Yourself »

What you should already know

Before you continue, you should have some basic understanding of the following:

- HTML
- Basic XML

If you want to study these subjects first, find the tutorials on our [Home page](#).

What is SVG?

- SVG stands for Scalable Vector Graphics
- SVG is used to define vector-based graphics for the Web
- SVG defines the graphics in XML format
- Every element and every attribute in SVG files can be animated
- SVG is a W3C recommendation
- SVG integrates with other W3C standards such as the DOM and XSL

ADVERTISEMENT

SVG is a W3C Recommendation

SVG 1.0 became a W3C Recommendation on 4 September 2001.

SVG 1.1 became a W3C Recommendation on 14 January 2003.

SVG 1.1 (Second Edition) became a W3C Recommendation on 16 August 2011.

SVG Advantages

Advantages of using SVG over other image formats (like JPEG and GIF) are:

- SVG images can be created and edited with any text editor
- SVG images can be searched, indexed, scripted, and compressed
- SVG images are scalable
- SVG images can be printed with high quality at any resolution
- SVG images are zoomable
- SVG graphics do NOT lose any quality if they are zoomed or resized
- SVG is an open standard
- SVG files are pure XML

Creating SVG Images

SVG images can be created with any text editor, but it is often more convenient to create SVG images with a drawing program, like [Inkscape](#).

HTML Canvas

The [HTML <canvas> element](#) can be used to draw graphics on a web page:

The graphic below is created with <canvas>:

It shows four elements: a red rectangle, a gradient rectangle, a multicolor rectangle, and a multicolor text.

HTML Canvas Tutorial

The HTML `<canvas>` element is used to draw graphics on a web page.

The graphic above is created with `<canvas>`.

It shows four elements: a red rectangle, a gradient rectangle, a multicolor rectangle, and a multicolor text.

What is HTML Canvas?

The HTML `<canvas>` element is used to draw graphics, on the fly, via scripting (usually JavaScript).

The `<canvas>` element is only a container for graphics. You must use a script to actually draw the graphics.

Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

HTML Canvas Can Draw Text

Canvas can draw colorful text, with or without animation.

HTML Canvas Can Draw Graphics

Canvas has great features for graphical data presentation with an imagery of graphs and charts.

HTML Canvas Can be Animated

Canvas objects can move. Everything is possible: from simple bouncing balls to complex animations.

HTML Canvas Can be Interactive

Canvas can respond to JavaScript events.

Canvas can respond to any user action (key clicks, mouse clicks, button clicks, finger movement).

HTML Canvas Can be Used in Games

Canvas' methods for animations, offer a lot of possibilities for HTML gaming applications.

Canvas Example

In HTML, a `<canvas>` element looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

The `<canvas>` element must have an `id` attribute so it can be referred to by JavaScript.

The width and height attribute is necessary to define the size of the canvas.

Tip: You can have multiple `<canvas>` elements on one HTML page.

By default, the `<canvas>` element has no border and no content.

To add a border, use a `style` attribute:

Example

```
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;"></canvas>
```

[Try it Yourself »](#)

The next chapters show how to draw on the canvas.

See Also:

[W3Schools' Full Canvas Reference](#)

Browser Support

The `<canvas>` element is an HTML5 standard (2014).

`<canvas>` is supported in all modern browsers:

Chrome	Edge	Firefox	Safari	Opera
Yes	Yes	Yes	Yes	Yes

Unit 3

CSS Tutorial

CSS is an acronym for **Cascading Style Sheet** and it was created by **Håkon Wium Lie** in 1996 for the web developers to change the layout, colors, and fonts of their websites.

CSS is a style sheet or presentational language that is used to layout, format, and style documents that are written in HTML to

make them look beautiful. CSS is generally used with HTML to change the style of web pages and user interfaces. You can use CSS to change the color, backgrounds, borders, paddings, margins, fonts, icons, position and various other properties of HTML elements in a web document.

CSS Examples

```
<html>
<head>
  <style>
    div{ background-color: lightgrey; width:100%; padding:5px}

    h1{color: #40a944; text-align: center;}

    p{ font-family: verdana;font-size: 20px;}
  </style>
</head>
<body>
  <div>
    <h1>CSS Example</h1>
    <p>This is a paragraph.</p>
  </div>
</body>
</html>
```

Why to Learn CSS?

Cascading Style Sheets, fondly referred to as **CSS**, is a simple design language intended to simplify the process of making web pages presentable.

CSS is a **MUST** for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.

Create Stunning Web site - CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are

sized and laid out, what background images or colors are used, layout designs, variations in display for different devices and screen sizes as well as a variety of other effects.

- **Become a web designer** - If you want to start a career as a professional web designer, HTML and CSS designing is a must skill.
- **Control web** - CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.
- **Learn other languages** - Once you understand the basics of HTML and CSS then other related technologies like javascript, php, or angular are become easier to understand.

Applications of CSS

As mentioned before, CSS is one of the most widely used style language over the web.

CSS saves time - You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.

- **Pages load faster** - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
- **Easy maintenance** - To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
- **Superior styles to HTML** - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- **Multiple Device Compatibility** - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
- **Global web standards** - Now HTML attributes are being deprecated and it is being recommended to use CSS. So its

a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

Audiences

This **CSS tutorial** will help both students as well as professionals who want to make their websites or personal blogs more attractive. We recommend reading this tutorial, in the sequence listed in the left side menu.

Today, HTML and CSS are essential languages to learn for anyone involved in the web development process including Web Developers, Web Designers, Graphic Designers, Content Managers, and Project Managers etc.

Prerequisites

Don't worry, you really don't need to be a computer geek to learn CSS. It's very easy styling and presentational language which anyone can learn on the go. You simply needs to be familiar with:

- Basic word processing using any text editor.
- How to create directories and files.
- How to navigate through different directories.
- Internet browsing using popular browsers like Internet Explorer or Firefox.
- Developing simple Web Pages using HTML or XHTML.

If you are new to HTML and XHTML, then we would suggest you to go through our HTML Tutorial or XHTML Tutorial first.

Include CSS in a webpage

CSS is used for styling the look and formatting of a document written in HTML. There are three ways to add CSS in HTML

- **Inline CSS:** Styles added directly to the HTML element.
- **Internal CSS:** Styles defined at the `<head>` section of the document.
- **External CSS:** Styles defined in a separate file.

Inline Style

Inline style is the approach of adding CSS rules directly to the HTML element using the `style` attribute. For example,

```
<p
style
="color: red;">Hello, Good Morning</p>
```

Here,

- `style` - defines the CSS for the element `<p>`
- `color: red` - changes the text of the `<p>` element to the color `red`

The `style` attribute can be added to any element but the styles will only be applied to that particular element. For example,

```
<html lang="en">

<head>
  <title>Browser</title>
</head>

<body>
  <h1>This is h1</h1>
  <p>This paragraph doesn't have CSS.</p>
  <p style="color:red">This paragraph is styled with inline CSS.</p>
</body>

</html>
```

Browser Output

This is h1

This paragraph doesn't have CSS.

This paragraph is styled with inline CSS.

Internal CSS

Internal CSS applies CSS styles to a specific HTML document. Internal CSS is defined inside an HTML document using `<style>` attribute within the `head` tag of an HTML. For example,

```
<head>
  <style>
    p {
      color: red;
    }
  </style>
</head>
```

Here,

- `<style>` - defines CSS
- `p` - selects `p` tag and applies CSS to it
- `color: red;` - changes the text color of the content of `<p>` elements to `red`

Now, let's look at a complete example of how we use internal CSS in an HTML.

```
<html lang="en">

<head>
  <style>
    p {
      color: red;
    }
  </style>

  <title>Browser</title>
</head>

<body>
  <h1>Internal CSS Example</h1>
  <p>This is Paragraph 1</P>
  <p>This is paragraph 2</p>
  <div>This is a content inside a div</div>
</body>

</html>
```

Browser Output



Browser

Internal CSS Example

This is Paragraph 1

This is paragraph 2

This is a content inside a div

Note: The styles defined in an internal stylesheet will only affect the elements on the current HTML page and will not be available on other HTML pages.

external-css External CSS

External CSS is an approach to applying CSS styles to HTML pages by defining the CSS in a separate file.

Let's see an example:

```
p {  
  color: blue;  
}
```

Here, we have CSS in a separate file named **style.css**. The external CSS file should have a **.css** extension.

The external CSS file can be linked to the HTML document by using the `link` element in the HTML. For example,

```
<head>  
  <link href="style.css" rel="stylesheet">  
</head>
```

We use the `<link>` tag to link the **style.css** file to an HTML document. In the above code,

- `href="style.css"` - URL or file path to the external CSS file.
- `rel="stylesheet"` - indicates the linked document is a CSS file

Example: External CSS

Let's see a complete example of using external CSS in an HTML document.


style.css


```
p {  
    color: blue;  
}
```

Now, let's connect it with html file

```
<html lang="en">  
  
<head>  
    <link href="style.css" rel="stylesheet">  
  
    <title>Browser</title>  
</head>  
  
<body>  
    <p>This is a sample text.</p>  
</body>  
  
</html>
```

Browser Output:

 Browser

This is a sample text.

Using Multiple Stylesheet

We can link multiple CSS file to an HTML file. Suppose we have the following two CSS files.

style.css

```
p {
```

```
    color: red;
}

div {
    color: yellow;
}
```

main.css

```
body {
    background: lightgreen;
}
```

Now, let's link these two CSS files to our HTML document.

```
<html lang="en">

<head>
  <link href="main.css" rel="stylesheet">
  <link href="style.css" rel="stylesheet">

  <title>Browser</title>
</head>

<body>
  <h1>Multiple Stylesheet Example</h1>
  <p>This is Paragraph 1</P>
  <p>This is paragraph 2</p>
  <div>This is a content inside a div</div>
</body>

</html>
```

Browser Output

Multiple Stylesheet Example

This is Paragraph 1

This is paragraph 2

This is a content inside a div

Here, we have linked **main.css** and **style.css** to our HTML file. Adding multiple CSS files helps us organize our CSS files into manageable parts.

Inline Style Override Internal Style

If an internal CSS and inline CSS are both applied to a tag, the styling from the inline tag is applied. Let's see an example.

```
<html lang="en">

<head>
  <style>
    h1 {
      color: blue;
    }

    p {
      background: red;
    }
  </style>

  <title>Browser</title>
```

```
</head>

<body>
  <h1 style="color: green">Priority Example</h1>
  <p>This is Paragraph 1</P>
  <p style="background: orange">This is paragraph 2</p>
</body>

</html>
```

Browser Output



Browser

Priority Example

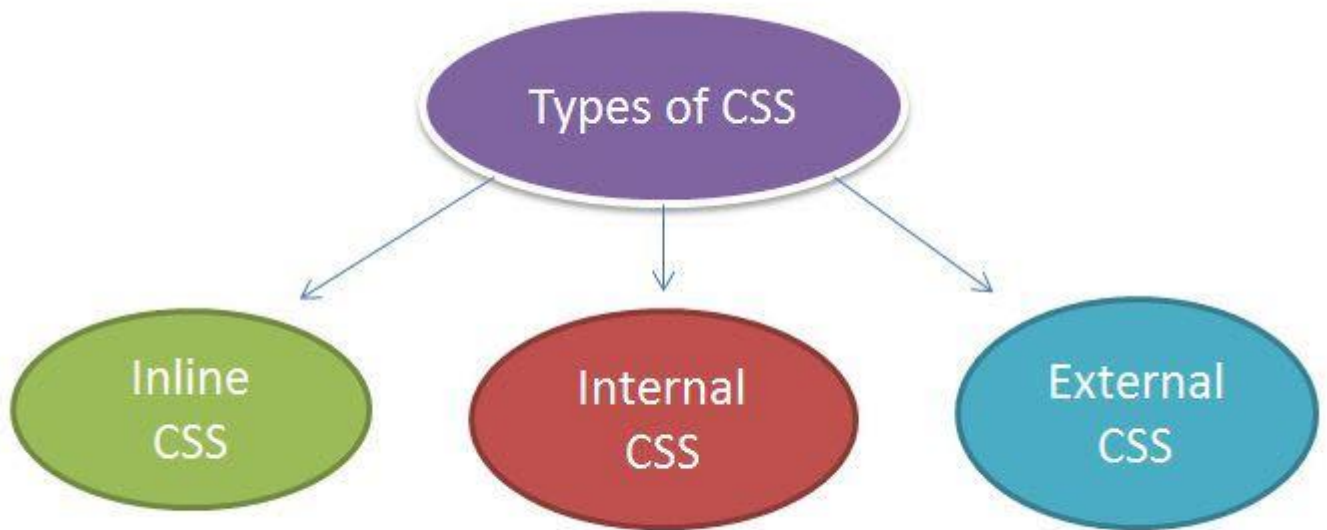
This is Paragraph 1

This is paragraph 2

As you can see the styling from inline CSS is applied to elements.

3 ways to add CSS to your HTML web page

[#html#css#style](#)



You all must be aware of the analogy about HTML, CSS, and Javascript that is being made with the human body. If HTML is the bones, then CSS is the muscle and javascript is the brain of the webpage.

There are millions of websites available on the internet, the only way for a website to stand out from others is its styling and interactivity. HTML pages without styles are just word documents getting displayed on the browser. So styles play a very crucial role in the development of any website.

Styles can be added to any HTML page in three ways: **Inline, Internal, and External.**

Let's get into the details for these three ways and how to apply the same on our web page.

Inline CSS

For inline styles, we use the style attribute of the HTML tags. CSS is passed as a string to the style attribute which adds the styles to the tags.

For example: If we want to make our web page background blue with inline CSS, we can write something like this:

```
<body style="background-color: blue"></body>
```

But it is a best practice to separate styles from the HTML tags that is the reason internal and external CSS comes into the picture.

Internal CSS

For Internal CSS, a style tag is used in which all the styles related to the webpage are added. This style tag is added in the head tag of the page so that the styles are added even before the HTML document is rendered.

For example: If we want to make our web page background as blue with internal css, we can right something like this:

```
<head>
  <style>
    body {
      background-color: blue;
    }
  </style>
</head>
```

Internal CSS does its job but imagine a scenario where we have to use the same styles for more than one page. In this case, if we use internal CSS it will lead to writing the same code twice, to overcome this external CSS is being used.

External CSS

In external CSS we use a separate file with a .css extension where we write all our styles. This CSS file can be used by multiple webpages by using a link tag which is added under the head tag.

For example: If we want to make our web page background as blue with external CSS, we have to make changes in two files.

First, we will add all our styles to our CSS file, for our example, it will be styles.css

```
body{
  background-color: blue;
}
```

Then we will link this css file to our html file as follows:

```
<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

Grouping Selectors in CSS

The CSS grouping selector is used to select multiple elements and style them together. This reduces the code and extra effort to declare common styles for each element. To group selectors, each selector is separated by a space.

Syntax

The syntax for CSS grouping selector is as follows –

```
element, element {
```

```
/*declarations*/  
}
```

The following examples illustrate CSS grouping selector –

Example

[Live Demo](#)

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
article, p, img {  
    display: block;  
    margin: auto;  
    text-align: center;  
    border-bottom: double orange;  
}  
</style>  
</head>  
<body>  
<article>Demo Text</article>  
<p>This is demo text.</p>  
<br/>  
  
</body>  
</html>
```

Output

This gives the following output –



Example

[Live Demo](#)

```
<!DOCTYPE html>
<html>
<head>
<style>
div::after,p::after{
    content: "Demo text";
    margin: 4px;
    box-shadow: inset 0 0 4px darkorange;
}
</style>
</head>
<body>
<div></div>
<p>This is example text.</p>
</body>
</html>
```

Output

This gives the following output –

Demo text

This is example text. Demo text

Introduction to XML

XML is a software- and hardware-independent tool for storing and transporting data.

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

XML Does Not DO Anything

Maybe it is a little hard to understand, but XML does not DO anything.

This note is a note to Tove from Jani, stored as XML:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML above is quite self-descriptive:

- It has sender information
- It has receiver information
- It has a heading
- It has a message body

But still, the XML above does not DO anything. XML is just information wrapped in tags.

Someone must write a piece of software to send, receive, store, or display it:

Note

To: Tove

From: Jani

Reminder

Don't forget me this weekend!

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

ADVERTISEMENT

XML Does Not Use Predefined Tags

The XML language has no predefined tags.

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

HTML works with predefined tags like <p>, <h1>, <table>, etc.

With XML, the author must define both the tags and the document structure.

XML is Extensible

Most XML applications will work as expected even if new data is added (or removed).

Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <body>).

Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.

The way XML is constructed, older version of the application can still work:

```
<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

Old Version

Note

To: Tove

From: Jani

Reminder

Don't forget me this weekend!

New Version

Note

To: Tove

From: Jani

Date: 2015-09-01 08:30

Don't forget me this weekend!

XML Simplifies Things

- XML simplifies data sharing
- XML simplifies data transport
- XML simplifies platform changes
- XML simplifies data availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

XML is a W3C Recommendation

XML became a W3C Recommendation as early as in February 1998.

DHTML Introduction

DHTML, or Dynamic HTML, is a technology that differs from traditional HTML. DHTML combines HTML, CSS, JavaScript, and the Document Object Model (DOM) to create dynamic content. It uses the Dynamic Object Model to modify settings, properties, and methods. Scripting is also an essential component of DHTML, which was part of earlier computing trends. It is supported by some versions of Netscape Navigator and Internet Explorer 4.0 and higher.

HTML: HTML stands for Hypertext Markup Language and it is a client-side markup language. It is used to build the block of web pages.

Javascript: It is a Client-side Scripting language. Javascript is supported by most of browsers, and also has cookie collection to determine the user's needs.

CSS: The abbreviation of CSS is Cascading Style Sheet. It helps in the styling of the web pages and helps in designing of the pages. The CSS rules for DHTML will be modified at different levels using JS with event handlers which adds a significant amount of dynamism with very little code.

DOM: It is known as a Document Object Model which acts as the weakest link in it. The only defect in it is that most of the browsers does not support DOM. It is a way to manipulate the static content.

DHTML is not a technology; rather, it is the combination of three different technologies, client-side scripting (JavaScript or VBScript), cascading style sheets and document object model.



Note: Many times DHTML is confused with being a language like HTML but it is not. It must be kept in mind that it is an interface or browsers enhancement feature which makes it possible to access the object model through Javascript language and hence make the webpage more interactive.

Key Features:

- Tags and their properties can be changed using DHTML.
- It is used for real-time positioning.
- Dynamic fonts can be generated using DHTML.
- It is also used for data binding.
- It makes a webpage dynamic and be used to create animations, games, applications along with providing new ways of navigating through websites.

- The functionality of a webpage is enhanced due to the usage of low-bandwidth effect by DHTML.
- DHTML also facilitates the use of methods, events, properties, and codes.

Why Use DHTML?

DHTML makes a webpage dynamic but Javascript also does, the question arises that what different does DHTML do? So the answer is that DHTML has the ability to change a webpages look, content and style once the document has loaded on our demand without changing or deleting everything already existing on the browser's webpage. DHTML can change the content of a webpage on demand without the browser having to erase everything else, i.e. being able to alter changes on a webpage even after the document has completely loaded.

Advantages:

- Size of the files are compact in compared to other interactional media like Flash or Shockwave, and it downloads faster.
- It is supported by big browser manufacturers like Microsoft and Netscape.
- Highly flexible and easy to make changes.
- Viewer requires no extra plug-ins for browsing through the webpage that uses DHTML, they do not need any extra requirements or special software to view it.
- User time is saved by sending less number of requests to the server. As it is possible to modify and replace elements even after a page is loaded, it is not required to create separate pages for changing styles which in turn saves time in building pages and also reduces the number of requests that are sent to the server.
- It has more advanced functionality than a static HTML. it is capable of holding more content on the web page at the same time.

Disadvantages:

- It is not supported by all the browsers. It is supported only by recent browsers such as Netscape 6, IE 5.5, and Opera 5 like browsers.
- Learning of DHTML requires a lot of pre-requisites languages such as HTML, CSS, JS, etc should be known to the designer before starting with DHTML which is a long and time-consuming in itself.
- Implementation of different browsers are different. So if it worked in one browser, it might not necessarily work the same way in another browser.
- Even after being great with functionality, DHTML requires a few tools and utilities that are some expensive. For example, the DHTML text editor, Dreamweaver. Along with it the improvement cost of transferring from HTML to DHTML makes cost rise much higher.

Difference between HTML and DHTML:

- HTML is a markup language while DHTML is a collection of technologies.
- HTML is used to create static webpages while DHTML is capable of creating dynamic webpages.
- DHTML is used to create animations and dynamic menus but HTML not used.

- HTML sites are slow upon client-side technologies whereas DHTML sites are comparatively faster.
- Web pages created using HTML are rather simple and have no styling as it uses only one language whereas DHTML uses HTML, CSS, and Javascript which results in a much better and way more presentable webpage.
- HTML cannot be used as server side code but DHTML used as server side code.
- DHTML needs database connectivity but not in case of HTML.
- Files in HTML are stored using .htm or .html extension while DHTML uses .dhtm extension.
- HTML requires no processing from the browser but DHTML does.

DCOM

1.What Is DCOM

DCOM enables components residing on different computers to interact as if they were local, allowing for the creation of distributed applications. It is built on top of the Remote Procedure Call (RPC) protocol, which facilitates communication between processes on different computers. DCOM uses the Object Remote Procedure Call (ORPC) layer to provide a higher level of abstraction for clients to interact with server objects.

The main features of DCOM include:

1. Location Transparency: Clients can access remote objects as if they were local, hiding the complexity of network communication.
2. Language Independence: DCOM supports various programming languages, allowing components written in different languages to interact seamlessly.
3. Security: DCOM provides multiple levels of security, including authentication, authorization, and encryption, to protect data during communication.
4. Scalability: DCOM enables the distribution of components across multiple machines, allowing applications to scale as needed.
5. Object Activation: DCOM supports object activation, which allows clients to create remote objects on demand.

The Purpose Of DCOM

The primary purpose of DCOM (Distributed Component Object Model) is to enable the development of distributed applications by allowing software components to communicate and interact with each other across a network. DCOM extends the capabilities of the Component Object Model (COM) to support communication between components on different computers as if they were on the same machine.

Benefits Of DCOM

DCOM provides several benefits for distributed application development. These benefits include:

1. **Location Transparency:** DCOM allows developers to create applications that can access remote objects as if they were local, simplifying the development process and providing a consistent programming model for both local and remote objects.
2. **Language Independence:** DCOM supports various programming languages, enabling components written in different languages to interact seamlessly..
3. **Reusability and Modular Design:** DCOM encourages the creation of reusable, modular software components that can be combined and reused in various configurations. This can lead to reduced development time, easier maintenance, and increased software quality.
4. **Scalability:** DCOM enables the distribution of components across multiple machines, allowing applications to scale as needed. This is particularly important for enterprise-level applications that need to handle large workloads or adapt to changing demands.
5. **Security:** DCOM provides multiple levels of security, including authentication, authorization, and encryption, to protect data during communication between components. This ensures that sensitive information is not exposed during the execution of distributed applications.
6. **Object Activation:** DCOM supports object activation, which allows clients to create remote objects on demand. This can help reduce resource usage and improve application performance by only activating objects when they are needed.
7. **Integration with Existing Technologies:** DCOM integrates with other Microsoft technologies, such as Active Directory for security and

administration, and OLE (Object Linking and Embedding) for embedding and linking objects in documents and applications.

Limitations Of DCOM

, it also has some limitations that have led to its replacement by more modern technologies like the .NET Framework, Web Services, and RESTful APIs. Some of these limitations include:

1. **Platform Dependency:** DCOM is primarily designed for Windows-based systems, making it less suitable for cross-platform development. Although there have been attempts to implement DCOM on other platforms, the support is limited and not as seamless as other technologies.
2. **Complexity:** DCOM can be challenging to configure, especially when dealing with security settings and network configurations. This complexity may lead to increased development time and difficulty in troubleshooting issues.
3. **Firewall Issues:** DCOM relies on the Remote Procedure Call (RPC) protocol, which uses dynamic port allocation. This can create issues when passing through firewalls, as they typically require specific ports to be opened, and DCOM's dynamic port allocation can make this difficult to manage.
4. **Limited Internet Support:** DCOM was designed for use within local area networks (LANs) and is not optimized for the Internet. While it can be used over the Internet, it can suffer from performance and security issues, making it less suitable for modern web-based applications.
5. **Interoperability:** Although DCOM supports various programming languages, it can struggle with interoperability between different platforms or when integrating with non-DCOM technologies.
6. **Maintenance and Debugging:** Debugging and maintaining DCOM-based applications can be challenging, especially when dealing with remote objects and distributed components. This can increase the time required to identify and resolve issues in the application.
7. **Obsolescence:** With the introduction of newer technologies like .NET Framework, Web Services, and RESTful APIs, DCOM has become less relevant in the world of distributed application development. These modern technologies offer better performance, security, and

ease of development, making them more suitable for current development practices.

Despite these limitations, DCOM is still used in some legacy systems, and understanding its drawbacks can help developers make informed decisions when choosing technologies for distributed application development.

How Does DCOM Work

DCOM is a Microsoft technology that extends the capabilities of the Component Object Model (COM) to support communication between software components across a network. Here is an overview of how DCOM works:

1. **Client-Server Model:** DCOM follows a client-server model, where clients request services from server objects hosted on remote machines. Clients and servers communicate through interfaces, which are a collection of methods exposed by the server objects.
2. **Interface and Object:** Clients interact with server objects using interfaces, which act as contracts that define methods and properties that the object supports. Objects are instances of classes, and they implement one or more interfaces to provide their services.
3. **Object Activation:** When a client wants to use a remote object, it requests the object's activation. DCOM activates the object on the server and returns a reference to the client. The client can then call methods on the object through the reference.
4. **RPC and ORPC:** DCOM is built on top of the Remote Procedure Call (RPC) protocol, which enables one process to call functions in another process running on a remote computer. DCOM uses the Object Remote Procedure Call (ORPC) layer to provide a higher level of abstraction, allowing clients to interact with server objects instead of dealing with low-level RPC details.
5. **Proxies and Stubs:** To communicate with remote objects, DCOM uses proxies and stubs. A proxy is a local object that represents a remote object in the client process, while a stub is a remote object that represents the local object in the server process. Proxies and stubs handle the marshaling and unmarshaling of data between the client and server during method calls.
6. **Security:** DCOM provides multiple levels of security, including authentication, authorization, and encryption. Authentication verifies the identity of the client and server, authorization determines what

actions the client is allowed to perform, and encryption protects the data during communication.

7. Distributed Garbage Collection: DCOM implements distributed garbage collection to manage the lifecycle of remote objects. When a client no longer needs a remote object, it releases the reference, and DCOM checks if there are any remaining references to the object. If there are no remaining references, DCOM deactivates the object and reclaims the resources.

DCOM enabled the development of distributed applications by providing a framework for communication between software components on different computers.

Event Bubbling

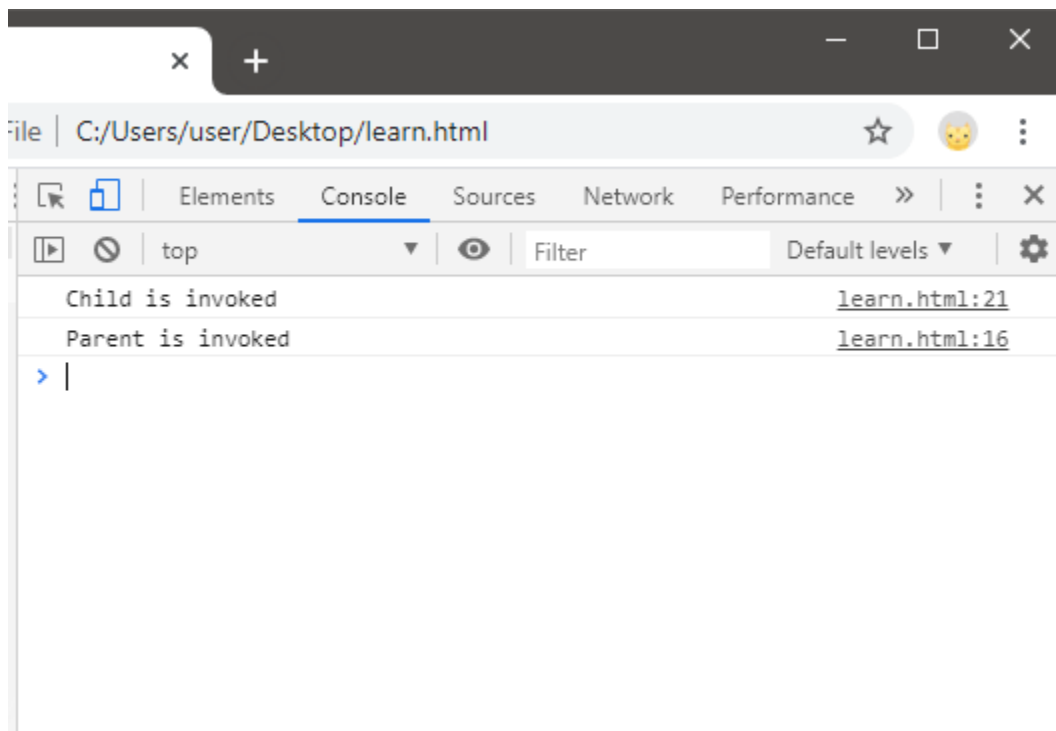
While developing a webpage or a website via [JavaScript](#), the concept of event bubbling is used where the event handlers are invoked when one element is nested on to the other element and are part of the same event. This technique or method is known as Event Bubbling. Thus, while performing event flow for a web page, event bubbling is used. We can understand event bubbling as a sequence of calling the event handlers when one element is nested in another element, and both the elements have registered listeners for the same event. So beginning from the deepest element to its parents covering all its ancestors on the way to top to bottom, calling is performed.

Example of Event Bubbling

Let's look at the below example to understand the working concept of Event Bubbling:

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <meta name="viewport" content="width=device-width">
6.   <title>Event Bubbling</title>
7. </head>
8. <body>
9.   <div id="p1">
10.    <button id="c1">I am child button</button>
11.  </div>
12.
13. <script>
14.   var parent = document.querySelector('#p1');
15.   parent.addEventListener('click', function(){
16.     console.log("Parent is invoked");
17.   });
18.
19.   var child = document.querySelector('#c1');
20.   child.addEventListener('click', function(){
21.     console.log("Child is invoked");
22.   });
23. </script>
24. </body>
25. </html>
```

Output:

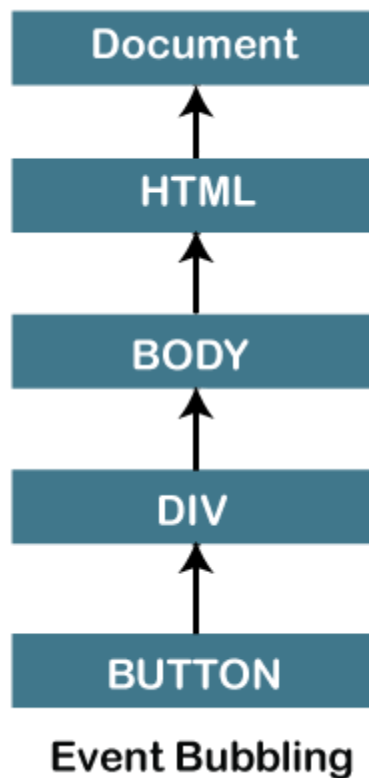


Explanation of Code:

- The above code is a HTML and JavaScript based code.
- We have used a div tag having div id = p1 and within div we have nested a button having button id = c1.
- Now, within the JavaScript section, we have assigned the html elements (p1 and c1) using the `querySelector ()` function to the variable parent and child.
- After that, we have created and included an event which is the click event to both div element and child button. Also created two functions that will help us to know the sequence order of the execution of the parent and child. It means if the child event is invoked first, "child is invoked" will be printed otherwise "parent is invoked" will get printed.
- Thus, when the button is clicked, it will first print "child is invoked" which means that the function within the child event handler executes first. Then it moves to the invocation of the div parent function.

The sequence has taken place due to the concept of event bubbling. Thus, in this way event bubbling takes place.

e can also understand the flow of event with the help of the below flow chart:



It means when the user click on the button, the click event flows in this order from bottom to top.

What is Event Bubbling?

Event bubbling, as I mentioned above, is the phase where the event bubbles up from the target element all the way to the global window object.

When an event "bubbles up" in the `DOM` event flow context, this refers to a phase in event flow where an event which traditionally occurs after the target phase continues to propagate from the target element to the root of the document through the `DOM` hierarchy.

How event bubbling works

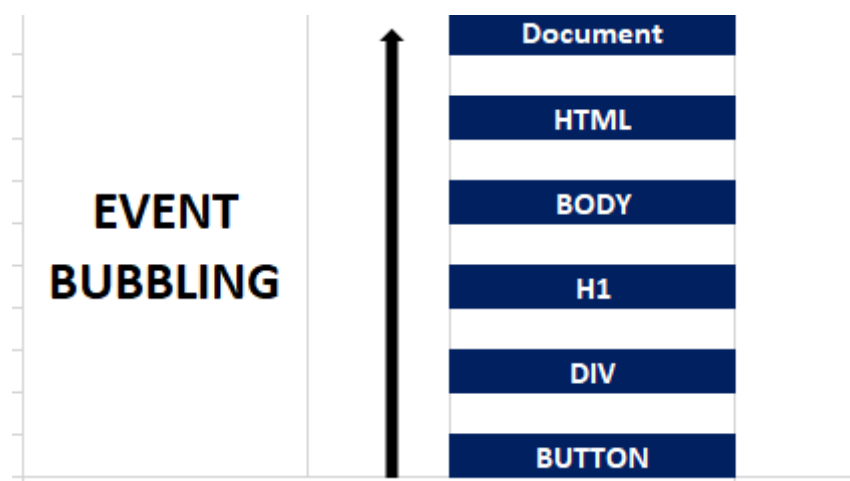
The event starts to bubble up through the `DOM` hierarchy after it has been processed at the target element. The event then travels from the target element to the root of the document. It passes through all ancestor elements according to how they are nested in the bubbling phase.

During an event handler's execution phase, any event handlers registered on ancestor elements get executed. This allows for easy capture of events at various stages of the DOM hierarchy.

This action can be stopped using the `stopPropagation()` method on the event object. It's useful when you aim to prevent event handlers from the ancestor elements getting triggered.

With bubbling, events can be handled in a hierarchical manner. By setting an event handler on the parent element or container, you can handle events from multiple child elements.

Event bubbling allows for an easier implementation of event delegation.



Here's an example of how it works, which I'll explain below:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Practice</title>
  </head>
  <body>
```



```
<h1>Bubbling and Capturing phase</h1>

<div>
  <button class="child">click me</button>
</div>

<script>
  const parent = document.querySelector("div");
  const child = document.querySelector(".child");

  parent.addEventListener("click", function () {
    console.log("clicked parent");
  });

  child.addEventListener("click", function () {
    console.log("clicked child");
  });
</script>
</body>
</html>
```

<!--

RESULTS OF THE ABOVE CODE

```
index.html:25 clicked child
index.html:21 clicked parent
```

-->

Code Explanation:

The above is a snippet of HTML and JavaScript code.

Inside the body element, we have the `h1`, `div`, and the `button` element. The `div` is the parent element of the `button` element. We gave a class name of `child` for the `button` element.

In the JavaScript section, we created variables for both the parent and child elements. Then we added event listeners to both the parent and child elements.

Thus, when the button gets clicked, the clicked child is first invoked. This means that the function within the parent element executes after the function within the child element.

What is data binding?

Data binding is the process that couples two data sources together and synchronizes them. With data binding, a change to an element in a data set automatically updates in the bound data set.

Data binding may be used for many reasons, such as to link an application's user interface ([UI](#)) and the data it displays, for data entry, reporting and in text box elements. It also lets internet users manipulate the representation of data in the elements of a web page without needing to use a complicated programming or [scripting](#) processes.

Data and data objects of different logic functions can be bound together in data binding. Data types with different languages can also be bound, such as data binding Extensible Markup Language ([XML](#)) and UI.

Each data change in one data set automatically reflects in the other bound data set. In binding syntax, the data source is the data provider, and the other data set is the data consumer. The binding forms the link between the data provider and the data consumer, enabling the connection between visual element data and a data source.

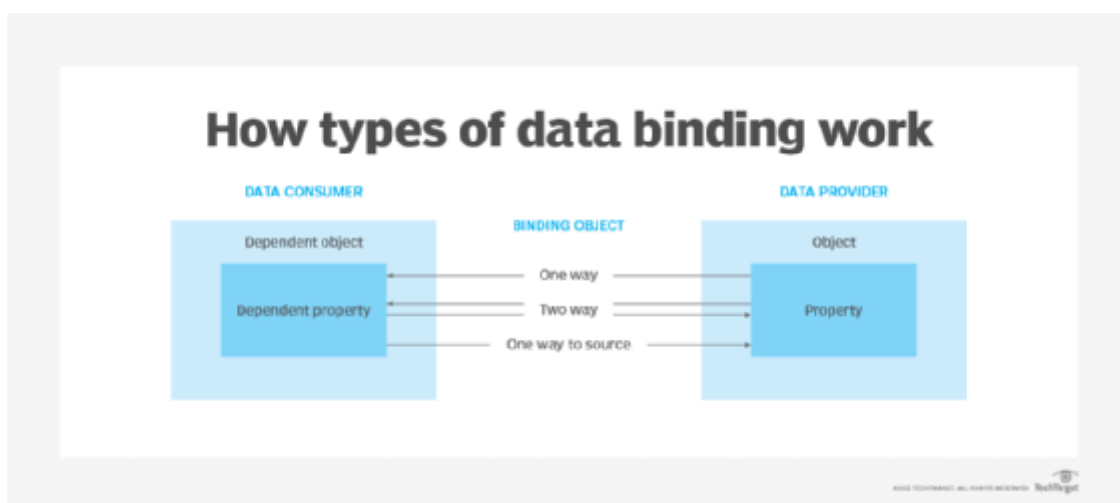
Data binding eliminates the need for Document Object Model (DOM) manipulation. DOM is an application programming interface, or API, for Hypertext Markup Language ([HTML](#)) and XML.

What are the types of data binding?

Types of data binding are typically defined by their data flow and include the following:

- **One-way binding** is a relatively simple type of data binding. Changes to the data provider update automatically in the data consumer data set, but not the other way around.
- **Two-way binding** is where changes to either the data provider or the data consumer automatically updates the other.
- **One-way-to-source binding** is the reverse of one-way binding. Changes to the data consumer automatically update the data provider but not the other way around.
- **One-time binding** is where changes to the data provider do not automatically update the data consumer. This approach is useful when only a snapshot of the data is needed, and the data is static.

Data binding can be simple or complex. Microsoft defines simple data binding as the ability to bind to a single data element. Complex data binding is when multiple elements are bound together.



Learn

how data binding works through this visualization.

How to use data binding

The data binding architecture consists of data source objects ([DSOs](#)) that supply information to viewed pages. DSOs also supply information to data consumers that display the DSO information and agents that ensure the data is synchronized between the DSOs and the consumers.

In a Windows [.NET framework](#), both simple and complex data binding options are available. Simple data binding is configured so that one control in the UI is bound to the data value from the data source.

Windows Presentation Foundation in .NET uses data binding by connecting the properties of target objects and data sources, including [Common Language Runtime](#), Language Integrated Query and XML objects. Data templates are also provided to control the presentation of data.

Data binding libraries enable users to bind UI components to data sources in a declarative format. These libraries also provide classes and methods to make changes in data observable. Consequently, collections, fields and objects are more visible.

Data binding examples

The following examples show how data binding can be used:

- **Reporting.** Binding is a common way to compile reports that display data from a data source to a screen or printer.
- **Data entry.** Data binding is also a common way to enter large amounts of data and keep it updated and synchronized to a data source.
- **Lookup tables.** Lookup tables are information tables that are typically a part of larger data displays. Data binding and controls are used to display and change data.
- **Master-detail formats.** This is a model for communication protocols where one device or process controls another. These formats may have two tables of data bound together.

Data binding tools

Data binding tools include the following:

- **Visual Studio** is a Microsoft product that provides design tools for working with custom objects as a data source in applications. Visual

Studio is also used to bind UI controls. Changes made to objects are automatically made in a database.

- **Data Binding Library** is a support library for Android developers that binds UI components to data sources.
- **Google Web Toolkit** is an open source tool from Google that enables web developers to create and maintain browser-based [Java](#) applications deployed as JavaScript. Google Web Toolkit has a feature called UiBinder that allows the creation of UIs.
- **AngularJS** is an open source JavaScript web framework that facilitates the development of single-page applications. The framework uses HTML and two-way data binding to synchronize data providers or data consumers automatically. As of January 1, 2022, Google is no longer releasing updates for AngularJS, and it has [discontinued long-term support](#).

UNIT 5

JavaScript Introduction

This page contains some examples of what JavaScript can do.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`.

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Example

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

[Try it Yourself »](#)

JavaScript accepts both double and single quotes:

Example

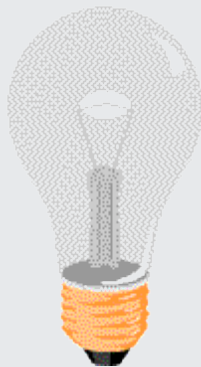
```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

[Try it Yourself »](#)

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the `src` (source) attribute of an `` tag:

The Light Bulb



Turn on the light

Turn off the light

[Try it Yourself »](#)

ADVERTISEMENT

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

```
document.getElementById("demo").style.fontSize = "35px";
```

[Try it Yourself »](#)

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "none";
```

[Try it Yourself »](#)

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "block";
```

[Try it Yourself »](#)

Did You Know?

JavaScript and [Java](#) are completely different languages, both in concept and design.

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.

[JavaScript Versions »](#)

Introduction

JavaScript is one of the most used languages when it comes to building web applications as it allows developers to wrap HTML and CSS code in it to make web apps interactive. It enables the interaction of users with the web application. It is also used for making animations on websites and has a large community on GitHub. JavaScript has tons of libraries, one of which is React which we will be covering in this article later.

Use cases:

- Building web server and its interactive functions
- Animations and graphics, adding special effects to web components
- Validating forms and exception errors
- Adding behavior and functionalities to web pages

In JavaScript, promises are used to handle asynchronous operations. When dealing with several asynchronous activities, where callbacks might cause callback hell and unmanageable code, they are simple to manage.

Important points to remember in JS:

- It is a scripting language
- It has unstructured code
- It is a programming language
- It is used both client-side and server-side to make web apps interactive
- It is built and maintained by Brendan Eich and the team
- Uses built-in browser DOM
- Extension of JavaScript file is .js

Client-side scripting

Client-side scripting is when the server sends the code along with the HTML web page to the client. The script is referred to by the code.

In other words, client-side scripting is a method for browsers to run scripts without having to connect to a server.

The code runs on the client's computer's browser either while the web page is loading or after it has finished loading.

Client-side scripting is mostly used for dynamic user interface components including pull-down menus, navigation tools, animation buttons, and data validation.

It is currently quickly expanding and evolving on a daily basis. As a result, creating client-side web programming has become easier and faster, lowering server demand.

By far the most popular client-side scripting languages or web scripting languages, JavaScript and jQuery are frequently utilized to construct dynamic and responsive webpages and websites.

The browser downloads the code to the local machine (temporarily) and begins processing it without the server. As a result, client-side scripting is browser-specific.

What is Client-Side Script, and how does it work?

A client-side script is a tiny program (or collection of instructions) that is put into a web page. It is handled by the client browser rather than the web server.

The client-side script, along with the HTML web page it is embedded in, is downloaded from the server at the client end. The code is interpreted and executed by the web browser, which then displays the results on the screen.

The client refers to the script that runs on the user's computer system. It can either be integrated (or injected) into the HTML content or stored in a separate file (known as an external script).

When the script files are requested, they are transmitted from the web server (or servers) to the client system. The script is run by the client's web browser, which subsequently displays the web page, including any visible script output.

To further understand, look at the diagram below.

Client-side scripts may also include instructions for the web browser to follow in response to user activities like clicking a page button. If a client wants to see the source code of a web page, they can typically look at them.

Client-side Scripting Languages are widely used.

Client-side scripting language or client-side programming refers to a language in which a client-side script or program is written utilizing syntax.

The following are the most popular client-side scripting languages:

1. **JavaScript** is the most commonly used client-side scripting or programming language. It is written in the ECMAScript programming language.

JavaScript is a dynamically typed (also known as weakly typed) object-oriented scripting language. It uses an integrated interpreter to run directly in the browser.

Weakly typed indicates that the variables can be implicitly transformed from one data type to another.

2. **VBScript**: This scripting language was developed by Microsoft, based on Visual Basic. It was mostly used to improve the functionality of web pages in Internet Explorer. The Internet Explorer web browser interprets VBScript. No one really uses it now.

3. **jQuery**: jQuery is a JavaScript library that is fast, tiny, and lightweight. It's used to turn a lot of JavaScript code into user-friendly functionality.

The jQuery language is used by the majority of the world's largest firms, including Google, Microsoft, IBM, Netflix, and others.

Scripting on the client-side

Client-side scripting is a technique for enhancing the interactivity of online pages or websites. It's mostly utilized on the front end, where the user can see what's going on through their browser. The following are some of the most common uses of client-side scripting:

- To get data from a user's screen or a web browser.
- In the realm of online games, this term is used.
- To make changes to a web page without having to reload it.

Validation is done using client-side scripting. If a user enters invalid credentials on the login page, the web page displays an error notice on the client machine instead of sending the information to the web server.

- Instead of simply displaying visuals, design ad banners that interact with the user.
- To make animated pictures that change as the mouse moves over them.
- A client-side script can be used to identify installed plug-ins and alert the user if one is needed.

The Benefits of Client-Side Scripting

The following are some of the many benefits of client-side scripting:

1. Client-side scripting is a simple language to learn and utilize. It only necessitates rudimentary programming knowledge or experience.
2. Client-side scripting has the advantage of being lightweight and reasonably simple to implement (syntax not too complex). The code modification and execution are both quick.

3. Data processing is done on the client side instead of the server, making large-scale applications easier to scale. As a result, the server's burden is reduced.
4. Data validation on the client side can be accomplished using a client-side scripting language such as JavaScript.
5. Client-side script execution is faster since the script is downloaded from the server and executed directly on the user's machine via the browser.
6. Client-side scripting can also be used for mathematical assessment.
7. Client-side programming facilitates the completion of complex activities in a limited number of steps.
8. Script code that is only performed by the browser and not by the server.
9. Executing script code takes far too little time.
10. When a user taps a key, moves the mouse, or clicks, the browser responds promptly.

Client-side Scripting's Drawbacks

The following are some of the disadvantages of client-side scripting:

1. Client-side scripting is insecure since the code is transmitted to the client as is, and therefore visible to it if the client looks at the source code of his web page. In a nutshell, code is almost always visible.
2. If we need to access databases or send sensitive data over the internet, client-side programming is not an option.
3. There is no guarantee that the user's browser has JavaScript enabled. As a result, notwithstanding the possibility of offloading, all essential functions must be loaded on the server.
4. The script's (or program's) smooth operation is entirely dependent on the client's browser, its settings, and its security level.
5. Debugging and maintaining a web application relying on excessive JavaScript might be difficult.
6. Client-side scripting languages are typically more constrained than server-side scripting languages in terms of capabilities.

Conclusion

This was all about client-side scripting in JavaScript. If you have any query related to React or JavaScript, do drop it down in the comment section also do check out [codedamn courses](#) if you want to learn more about JavaScript and React with its use cases and amazing projects. They also have an in-built playground for different programming languages and environment sets so do check that out and join [codedamn's community](#)!

What is JavaScript?

JavaScript is the **Programming Language** for the Web.

JavaScript can update and change both **HTML** and **CSS**.

JavaScript can **calculate**, **manipulate** and **validate** data.

JavaScript Quickstart Tutorial

This tutorial will take a quick look at the most important JavaScript data types.

JavaScript variables can be:

- Numbers
 - Strings
 - Objects
 - Arrays
-
- Functions

JavaScript Variables

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables:

Example

```
var x = 5;  
var y = 6;  
var z = x + y;
```

[Try it Yourself »](#)

From the example above, you can expect:

- x stores the value 5
- y stores the value 6
- z stores the value 11

JavaScript Numbers

JavaScript has only **one type** of number. Numbers can be written with or without decimals.

Example

```
var x = 3.14;    // A number with decimals
var y = 3;       // A number without decimals
```

[Try it yourself »](#)

All numbers are stored as double precision floating point numbers.

The maximum number of decimals is 17, but floating point is not always 100% accurate:

Example

```
var x = 0.2 + 0.1;    // x will be 0.30000000000000004
```

[Try it yourself »](#)

JavaScript Strings

Strings **store text**. Strings are written inside quotes. You can use **single** or double **quotes**:

Example

```
var carname = "Volvo XC60"; // Double quotes
var carname = 'Volvo XC60'; // Single quotes
```

[Try it Yourself »](#)

The length of a string is found in the built in property **length**:

Example

```
var txt = "ABCDEFGHJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

[Try it Yourself »](#)

ADVERTISEMENT

JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
var car = "Fiat";
```

[Try it Yourself »](#)

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

```
var car = {type:"Fiat", model:"500", color:"white"};
```

[Try it Yourself »](#)

JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable.

Example

```
var cars = ["Saab", "Volvo", "BMW"];
```

[Try it Yourself »](#)

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

Example

```
function myFunction(p1, p2) {  
    return p1 * p2;           // The function returns the product of  
    p1 and p2  
}
```

[Try it Yourself »](#)

What can JavaScript Do?

This section contains some examples of what JavaScript can do:

- JavaScript Can Change HTML Content
 - JavaScript Can Change HTML Attribute Values
 - JavaScript Can Change HTML Styles (CSS)
 - JavaScript Can Hide HTML Elements
-
- JavaScript Can Show HTML Elements

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is **getElementById()**.

This example uses the method to "find" an HTML element (with id="demo") and changes the element content (**innerHTML**) to "Hello JavaScript":

Example

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

[Try it Yourself »](#)

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the src (source) attribute of an tag:

The Light Bulb

Turn on the light Turn off the light

[Try it Yourself »](#)

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

```
document.getElementById("demo").style.fontSize = "35px";  
or  
document.getElementById('demo').style.fontSize = '35px';
```

[Try it Yourself »](#)

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the display style:

Example

```
document.getElementById("demo").style.display = "none";  
or  
document.getElementById('demo').style.display = 'none';
```


[Try it Yourself »](#)

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the display style:

Example

```
document.getElementById("demo").style.display = "block";  
or  
document.getElementById('demo').style.display = 'block';
```

[Try it Yourself »](#)

JavaScript Example

1. [JavaScript Example](#)
2. [Within body tag](#)
3. [Within head tag](#)

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Let's create the first JavaScript example.

1. `<script type="text/javascript">`
2. `document.write("JavaScript is a simple language for javatpoint learners");`
3. `</script>`

[Test it Now](#)

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

The **document.write()** function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

3 Places to put JavaScript code

1. Between the body tag of html
 2. Between the head tag of html
 3. In .js file (external JavaScript)
-

1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

1. `<script type="text/javascript">`
 2. `alert("Hello Javatpoint");`
 3. `</script>`
- [Test it Now](#)
-

2) JavaScript Example : code between the head tag

Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.

In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.

To call function, you need to work on event. Here we are using onclick event to call msg() function.

1. `<html>`
2. `<head>`
3. `<script type="text/javascript">`
4. `function msg(){`
5. `alert("Hello Javatpoint");`
6. `}`
7. `</script>`

8. `</head>`
9. `<body>`
10. `<p>Welcome to JavaScript</p>`
11. `<form>`
12. `<input type="button" value="click" onclick="msg()"/>`
13. `</form>`
14. `</body>`
15. `</html>`

JavaScript Introduction

[Previous](#) [Next](#)

This page contains some examples of what JavaScript can do.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`.

The example below "finds" an HTML element (with `id="demo"`), and changes the element content (innerHTML) to "Hello JavaScript":

Example

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

[Try it Yourself »](#)

JavaScript accepts both double and single quotes:

Example

```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

[Try it Yourself »](#)

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the `src` (source) attribute of an `` tag:

The Light Bulb

Turn on the light Turn off the light

[Try it Yourself »](#)

ADVERTISEMENT

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

```
document.getElementById("demo").style.fontSize = "35px";
```

[Try it Yourself »](#)

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "none";
```

[Try it Yourself »](#)

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "block";
```

JavaScript Variables

[Previous](#) [Next](#)

Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

In this first example, `x`, `y`, and `z` are undeclared variables.

They are automatically declared when first used:

Example

```
x = 5;  
y = 6;  
z = x + y;
```

[Try it Yourself »](#)

Note

It is considered good programming practice to always declare variables before use.

From the examples you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

Example using var

```
var x = 5;  
var y = 6;  
var z = x + y;
```

[Try it Yourself »](#)

Note

The `var` keyword was used in all JavaScript code from 1995 to 2015.

The `let` and `const` keywords were added to JavaScript in 2015.

The `var` keyword should only be used in code written for older browsers.

Example using let

```
let x = 5;  
let y = 6;  
let z = x + y;
```

[Try it Yourself »](#)

Example using const

```
const x = 5;  
const y = 6;  
const z = x + y;
```

[Try it Yourself »](#)

Mixed Example

```
const price1 = 5;  
const price2 = 6;  
let total = price1 + price2;
```

[Try it Yourself »](#)

The two variables `price1` and `price2` are declared with the `const` keyword.

These are constant values and cannot be changed.

The variable `total` is declared with the `let` keyword.

The value `total` can be changed.

When to Use var, let, or const?

1. Always declare variables
2. Always use `const` if the value should not be changed
3. Always use `const` if the type should not be changed (Arrays and Objects)
4. Only use `let` if you can't use `const`
5. Only use `var` if you MUST support old browsers.

Just Like Algebra

Just like in algebra, variables hold values:

```
let x = 5;  
let y = 6;
```

Just like in algebra, variables are used in expressions:

```
let z = x + y;
```

From the example above, you can guess that the total is calculated to be 11.

Note

Variables are containers for storing values.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with \$ and _ (but we will not use it in this tutorial).
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Note

JavaScript identifiers are case-sensitive.

The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

$x = x + 5$

In JavaScript, however, it makes perfect sense: it assigns the value of $x + 5$ to x.

(It calculates the value of $x + 5$ and puts the result into x. The value of x is incremented by 5.)

Note

The "equal to" operator is written like `==` in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

```
const pi = 3.14;  
let person = "John Doe";  
let answer = 'Yes I am!';
```

[Try it Yourself »](#)

Declaring a JavaScript Variable

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the `var` or the `let` keyword:

```
var carName;
```

or:

```
let carName;
```

After the declaration, the variable has no value (technically it is **undefined**).

To **assign** a value to the variable, use the equal sign:

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

You can also assign a value to the variable when you declare it:

```
let carName = "Volvo";
```

In the example below, we create a variable called **carName** and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

Example

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

```
<script>
```

```
let carName = "Volvo";
```

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

```
</script>
```

[Try it Yourself »](#)

Note

It's a good programming practice to declare all variables at the beginning of a script.

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with **let** and separate the variables by **comma**:

Example

```
let person = "John Doe", carName = "Volvo", price = 200;
```

[Try it Yourself »](#)

A declaration can span multiple lines:

Example

```
let person = "John Doe",  
    carName = "Volvo",  
    price = 200;
```

[Try it Yourself »](#)

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

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

The variable `carName` will have the value `undefined` after the execution of this statement:

Example

```
let carName;
```

[Try it Yourself »](#)

Re-Declaring JavaScript Variables

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

The variable `carName` will still have the value "Volvo" after the execution of these statements:

Example

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

[Try it Yourself »](#)

Note

You cannot re-declare a variable declared with `let` or `const`.

This will not work:

```
let carName = "Volvo";  
let carName;
```

JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like `=` and `+`:

Example

```
let x = 5 + 2 + 3;
```

[Try it Yourself »](#)

You can also add strings, but strings will be concatenated:

Example

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

[Try it Yourself »](#)

Also try this:

Example

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

[Try it Yourself »](#)

Note

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

Example

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

[Try it Yourself »](#)

JavaScript Dollar Sign \$

Since JavaScript treats a dollar sign as a letter, identifiers containing \$ are valid variable names:

Example

```
let $ = "Hello World";  
let $$$ = 2;  
let $myMoney = 5;
```

[Try it Yourself »](#)

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function `$` is used to select HTML elements. In jQuery `$("p");` means "select all p elements".

JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing `_` are valid variable names:

Example

```
let _lastName = "Johnson";  
let _x = 2;  
let _100 = 5;
```

[Try it Yourself »](#)

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

JavaScript Functions

[❏ Previous](#)[Next ❏](#)

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

Example

```
// Function to compute the product of p1 and p2  
function myFunction(p1, p2) {  
  return p1 * p2;  
}
```

[Try it Yourself »](#)

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses `()`.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **`(parameter1, parameter2, ...)`**

The code to be executed, by the function, is placed inside curly brackets: **`{}`**

```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

Function **parameters** are listed inside the parentheses `()` in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

Function Return

When JavaScript reaches a **return** statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example

Calculate the product of two numbers, and return the result:

```
// Function is called, the return value will end up in x
let x = myFunction(4, 3);

function myFunction(a, b) {
  // Function returns the product of a and b
  return a * b;
}
```

[Try it Yourself »](#)

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

You can use the same code with different arguments, to produce different results.

The () Operator

The () operator invokes (calls) the function:

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {  
  return (5/9) * (fahrenheit-32);  
}
```

```
let value = toCelsius(77);
```

[Try it Yourself »](#)

Accessing a function with incorrect parameters can return an incorrect answer:

Example

```
function toCelsius(fahrenheit) {  
  return (5/9) * (fahrenheit-32);  
}
```

```
let value = toCelsius();
```

[Try it Yourself »](#)

Accessing a function without () returns the function and not the function result:

Example

```
function toCelsius(fahrenheit) {  
  return (5/9) * (fahrenheit-32);  
}
```

```
let value = toCelsius;
```

[Try it Yourself »](#)

Note

As you see from the examples above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);  
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

[Try it Yourself »](#)

You will learn a lot more about functions later in this tutorial.

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName
```

```
function myFunction() {  
  let carName = "Volvo";  
  // code here CAN use carName  
}
```

```
// code here can NOT use carName
```

[Try it Yourself »](#)

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Exercise:

Execute the function named `myFunction`.

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

Submit Answer »

[Start the Exercise](#)

JavaScript For Loop

◀ PreviousNext ▶

Loops can execute a block of code a number of times.

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

[Try it Yourself »](#)

Different Kinds of Loops

JavaScript supports different kinds of loops:

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **for/of** - loops through the values of an iterable object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

The For Loop

The **for** statement creates a loop with 3 optional expressions:

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example

```
for (let i = 0; i < 5; i++) {  
  text += "The number is " + i + "<br>";  
}
```

[Try it Yourself »](#)

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

ADVERTISEMENT

Expression 1

Normally you will use expression 1 to initialize the variable used in the loop (let i = 0).

This is not always the case. JavaScript doesn't care. Expression 1 is optional.

You can initiate many values in expression 1 (separated by comma):

Example

```
for (let i = 0, len = cars.length, text = ""; i < len; i++) {  
  text += cars[i] + "<br>";  
}
```

[Try it Yourself »](#)

And you can omit expression 1 (like when your values are set before the loop starts):

Example

```
let i = 2;
let len = cars.length;
let text = "";
for (; i < len; i++) {
  text += cars[i] + "<br>";
}
```

[Try it Yourself »](#)

Expression 2

Often expression 2 is used to evaluate the condition of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 2 is also optional.

If expression 2 returns true, the loop will start over again. If it returns false, the loop will end.

If you omit expression 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Expression 3

Often expression 3 increments the value of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 3 is optional.

Expression 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Expression 3 can also be omitted (like when you increment your values inside the loop):

Example

```
let i = 0;
let len = cars.length;
let text = "";
for (; i < len; ) {
  text += cars[i] + "<br>";
  i++;
}
```

[Try it Yourself »](#)

Loop Scope

Using `var` in a loop:

Example

```
var i = 5;

for (var i = 0; i < 10; i++) {
  // some code
}

// Here i is 10
```

[Try it Yourself »](#)

Using `let` in a loop:

Example

```
let i = 5;

for (let i = 0; i < 10; i++) {
  // some code
}

// Here i is 5
```

[Try it Yourself »](#)

In the first example, using `var`, the variable declared in the loop redeclares the variable outside the loop.

In the second example, using `let`, the variable declared in the loop does not redeclare the variable outside the loop.

When `let` is used to declare the `i` variable in a loop, the `i` variable will only be visible within the loop.

For/Of and For/In Loops

The `for/in` loop and the `for/of` loop are explained in the next chapter.

While Loops

The `while` loop and the `do/while` are explained in the next chapters.

JavaScript For In

[Previous](#)[Next](#)

The For In Loop

The JavaScript `for in` statement loops through the properties of an Object:

Syntax

```
for (key in object) {  
    // code block to be executed  
}
```


Example

```
const person = {fname:"John", lname:"Doe", age:25};
```

```
let text = "";
for (let x in person) {
  text += person[x];
}
```

[Try it Yourself »](#)

Example Explained

- The **for in** loop iterates over a **person** object
- Each iteration returns a **key** (x)
- The key is used to access the **value** of the key
- The value of the key is **person[x]**

For In Over Arrays

The JavaScript **for in** statement can also loop over the properties of an Array:

Syntax

```
for (variable in array) {
  code
}
```

Example

```
const numbers = [45, 4, 9, 16, 25];
```

```
let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}
```

[Try it Yourself »](#)

Do not use **for in** over an Array if the index **order** is important.

The index order is implementation-dependent, and array values may not be accessed in the order you expect.

It is better to use a **for** loop, a **for of** loop, or **Array.forEach()** when the order is important.

ADVERTISEMENT

Array.forEach()

The `forEach()` method calls a function (a callback function) once for each array element.

Example

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt += value;
}
```

[Try it Yourself »](#)

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

The example above uses only the value parameter. It can be rewritten to:

Example

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
```

```
numbers.forEach(myFunction);
```

```
function myFunction(value) {  
  txt += value;  
}
```

[Try it Yourself »](#)

JavaScript For Of

[❏ Previous](#)[Next ❏](#)

The For Of Loop

The JavaScript **for of** statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

Syntax

```
for (variable of iterable) {  
  // code block to be executed  
}
```

variable - For every iteration the value of the next property is assigned to the variable. *Variable* can be declared with **const**, **let**, or **var**.

iterable - An object that has iterable properties.

Browser Support

For/of was added to JavaScript in 2015 ([ES6](#))

Safari 7 was the first browser to support for of:

Chrome 38	Edge 12	Firefox 51	Safari 7
Oct 2014	Jul 2015	Oct 2016	Oct 2013

For/of is not supported in Internet Explorer.

Looping over an Array

Example

```
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}
```

[Try it Yourself »](#)

Looping over a String

Example

```
let language = "JavaScript";

let text = "";
for (let x of language) {
  text += x;
}
```

[Try it Yourself »](#)

The While Loop

The `while` loop and the `do/while` loop are explained in the next chapter.

JavaScript While Loop

[Previous](#)[Next](#)

Loops can execute a block of code as long as a specified condition is true.

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

Syntax

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

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

Example

```
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}
```

[Try it Yourself »](#)

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

The Do While Loop

The `do while` loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

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

Example

The example below uses a `do while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);
```

[Try it Yourself »](#)

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Comparing For and While

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a **for** loop to collect the car names from the cars array:

Example

```
const cars = ["BMW", "Volvo", "Saab", "Ford"];
let i = 0;
let text = "";

for (;cars[i];) {
  text += cars[i];
  i++;
}
```

[Try it Yourself »](#)

The loop in this example uses a **while** loop to collect the car names from the cars array:

Example

```
const cars = ["BMW", "Volvo", "Saab", "Ford"];
let i = 0;
let text = "";

while (cars[i]) {
  text += cars[i];
  i++;
}
```

[Try it Yourself »](#)

Javascript Data Structure Tutorial - Javascript Repetition Statements

Javascript while statement

To execute a set of statements when a condition is true, use the while loop.

The following code demonstrates how to use the while loop.

```
var number = 1;
var sum = 0;

while (number < 11) {
    sum += number;
    ++number;
    console.log(number);
}
console.log(sum);
```

Javascript for statement

To execute a set of statements a specified number of times, use the for loop.

The following code shows how to sum integers using a for loop

```
var number = 1;
var sum = 0;

for (var number = 1; number < 11; number++) {
    sum += number;
    console.log(sum);
}
console.log(sum);
```


Note

for loops are used frequently to access the elements of an array, as shown in the following example.

```
var numbers = [31, 17, 12, 22, 100, 1, 2, 3, 4, 4, 5, 65, 6, 7, 78,999];
var sum = 0;
for (var i = 0; i < numbers.length; ++i) {
    sum += numbers[i];
    console.log(sum);
}
console.log(sum);
```

-

[Next »](#)

Javascript Data Structure Tutorial - Javascript Functions

-

[« Previous](#)

-

[Next »](#)

JavaScript can define both value-returning functions and void-returning functions.

Example

The following code demonstrates how to create a value-returning functions and call it in JavaScript.

```
function factorial(number) { //from w w w . j a v a 2 s . c o m
    var product = 1;
    for (var i = number; i >= 1; --i) {
        product *= i;
    }
    return product;
}
console.log(factorial(2));
console.log(factorial(3));
console.log(factorial(4));
console.log(factorial(5));
console.log(factorial(10));
console.log(factorial(20));
```

Example 2

The following code illustrates how to write a function to operate on values.

```
function curve(arr, amount) {
    for (var i = 0; i < arr.length; ++i) {
        arr[i] += amount;
    }
}

var grades = [77, 73, 74, 81, 90, 34, 21, 12, 34, 54,];
curve(grades, 5);
```

```
console.log(grades);
```

Note

All function parameters in JavaScript are passed by value.

The reference objects, such as arrays, are passed to functions by reference, as was demonstrated in the code above.

Recursion

Function calls can be made recursively in JavaScript.

The factorial() function can be written recursively, like this:

```
function factorial(number) {  
    if (number == 1) {  
        return number;  
    }  
    else {  
        return number * factorial(number-1);  
    }  
}  
  
console.log(factorial(5));  
console.log(factorial(15));  
console.log(factorial(25));  
console.log(factorial(50));
```

•

[Next »](#)

Javascript Data Structure Tutorial - Javascript Variable

- [« Previous](#)
- [Next »](#)

Declaring and Intializing

JavaScript variables are global by default.

JavaScript variables don't have to be declared before using.

When a JavaScript variable is initialized without declaration, it becomes a global variable.

Example

To declare a variable in JavaScript, use the keyword **var** followed by a variable name.

Here are some examples:

```
var number;  
var name;  
var myNumber  
var longValue = 11111;  
var rate = 1.2;  
var greeting = "Hello, world!";  
var flag = false;  
var myValue = 'a';
```

- [Next »](#)

Javascript Data Structure Tutorial - Javascript Decision Statement

- [« Previous](#)
- [Next »](#)

Decision statements such as if and switch allow the programs to make decisions on statements to execute based on a Boolean expression.

Javascript has the if statement. We can use them to execute statements by a boolean value.

The if statement comes in three forms:

- The simple if statement
- The if-else statement
- The if-else if statement

Example for a simple if statement

The following code shows how to write a simple if statement.

```
var mid = -1;
var high = 50;
var low = 1;
var current = 13;
var found = -1;
/*from w w w . j a v a 2 s . c o m*/
console.log(mid);
if (current < mid) {
```

```
    mid = (current-low) / 2;
    console.log(mid);
}
```

Example for if-else statement

The following code demonstrates the if-else statement.

```
var mid = 25;
var high = 50;
var low = 1;
var current = 13;
var found = -1;
/* www.java2s.c om*/
console.log(mid);
if (current < mid) {
    mid = (current-low) / 2;
    console.log(mid);
} else {
    mid = (current+high) / 2;
    console.log(mid);
}
```

Example for if-else if statement

The following code illustrates the if-else if statement.

```
var mid = 25;
var high = 50;
var low = 1;
```

```

var current = 13;
var found = -1;
// w w w . j a v a 2 s . c o m
console.log(mid);
if (current < mid) {
    mid = (current-low) / 2;
    console.log(mid);
}
else if (current > mid) {
    mid = (current+high) / 2;
    console.log(mid);
}
else {
    found = current;
    console.log(found);
}
console.log("over");

```

Switch Statement

The switch statement provides a cleaner, more structured construction when you have several simple decisions to make.

The following code demonstrates how the switch statement works.

```

var monthNum = 1;
var monthName;
switch (monthNum) { //from w w w . j a v a 2 s . c o m
    case "1": monthName = "January";
        break;
    case "2": monthName = "February";
        break;
    case "3": monthName = "March";
        break;
    case "4": monthName = "April";
        break;
    case "5": monthName = "May";

```

```
        break;
    case "6": monthName = "June";
        break;
    case "7": monthName = "July";
        break;
    case "8": monthName = "August";
        break;
    case "9": monthName = "September";
        break;
    case "10": monthName = "October";
        break;
    case "11": monthName = "November";
        break;
    case "12": monthName = "December";
        break;
    default: console.log("Bad input");
}

console.log(monthName);
```

•

[Next »](#)

UNIT 5