

What is HTML?

HTML is the standard markup language for creating Web pages.

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

Structure of HTML5

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
</body>
</html>
```

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

HTML Versions

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

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

There are many differences between HTML and HTML5 which are discussed below:

HTML	HTML5
It didn't support audio and video without the use of flash player support.	It supports audio and video controls with the use of <code><audio></code> and <code><video></code> tags.
It uses cookies to store temporary data.	It uses SQL databases and application cache to store offline data.
Does not allow JavaScript to run in browser.	Allows JavaScript to run in background. This is

	possible due to JS Web worker API in HTML5.
Vector graphics is possible in HTML with the help of various technologies such as VML, Silver-light, Flash, etc.	Vector graphics is additionally an integral a part of HTML5 like SVG and canvas.
It does not allow drag and drop effects.	It allows drag and drop effects.
Not possible to draw shapes like circle, rectangle, triangle etc.	HTML5 allows to draw shapes like circle, rectangle, triangle etc.
It works with all old browsers.	It supported by all new browser like Firefox, Mozilla, Chrome, Safari, etc.
Older version of HTML is less mobile-friendly.	HTML5 language is more mobile-friendly.
Doctype declaration is too long and complicated.	Doctype declaration is quite simple and easy.
Elements like nav, header were not present.	New element for web structure like nav, header, footer etc.
Character encoding is long and complicated.	Character encoding is simple and easy.
It is almost impossible to get true GeoLocation of user with the help of browser.	One can track the GeoLocation of a user easily by using JS GeoLocation API.
It can not handle inaccurate syntax.	It is capable of handling inaccurate syntax.
Attributes like charset, async and ping are absent in HTML.	Attributes of charset, async and ping are a part of HTML 5.

What is an HTML Element?

An HTML element is defined by a start tag, some content, and an end tag:

`<tagname>` Content goes here... `</tagname>`

The HTML **element** is everything from the start tag to the end tag:

`<h1>`My First Heading`</h1>`

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

Start tag	Element content	End tag
<h1>	My First Heading	</h1>
<p>	My first paragraph.	</p>
 	<i>None</i>	<i>none</i>

Note: Some HTML elements have no content (like the
 element). These elements are called empty elements. Empty elements do not have an end tag!

The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration represents the document type, and helps browsers to display web pages correctly.

It must only appear once, at the top of the page (before any HTML tags).

The <!DOCTYPE> declaration is not case sensitive.

The <!DOCTYPE> declaration for HTML5 is:

```
<!DOCTYPE html>
```

COMMENT IN HTML

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

HTML Ordered List | HTML Numbered List

HTML Ordered List or Numbered List displays elements in numbered format. The HTML ol tag is used for ordered list. We can use ordered list to represent items either in numerical order format or alphabetical order format, or any format where an order is emphasized. There can be different types of numbered list:

- Numeric Number (1, 2, 3)
- Capital Roman Number (I II III)
- Small Roman Number (i ii iii)

- Capital Alphabet (A B C)
- Small Alphabet (a b c)

To represent different ordered lists, there are 5 types of attributes in `` tag.

Type	Description
Type "1"	This is the default type. In this type, the list items are numbered with numbers.
Type "I"	In this type, the list items are numbered with upper case roman numbers.
Type "i"	In this type, the list items are numbered with lower case roman numbers.
Type "A"	In this type, the list items are numbered with upper case letters.
Type "a"	In this type, the list items are numbered with lower case letters.

HTML Unordered List | HTML Bulleted List

HTML Unordered List or Bulleted List displays elements in bulleted format . We can use unordered list where we do not need to display items in any particular order. The HTML `ul` tag is used for the unordered list. There can be 4 types of bulleted list:

- disc
- circle
- square
- none

To represent different ordered lists, there are 4 types of attributes in `` tag.

Type	Description
Type "disc"	This is the default style. In this style, the list items are marked with bullets.

Type "circle"	In this style, the list items are marked with circles.
Type "square"	In this style, the list items are marked with squares.
Type "none"	In this style, the list items are not marked .

HTML Description List or Definition List displays elements in definition form like in dictionary. The <dl>, <dt> and <dd> tags are used to define description list.

The 3 HTML description list tags are given below:

1. **<dl> tag** defines the description list.
2. **<dt> tag** defines data term.
3. **<dd> tag** defines data definition (description).

HTML Table Colspan & Rowspan

HTML tables can have cells that spans over multiple rows and/or columns

2022		
FIESTA		

HTML Table - Colspan

To make a cell span over multiple columns, use the **colspan** attribute:

```
<table border="1">
  <tr>
    <th colspan="2">Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>43</td>
  </tr>
```

```

<tr>
  <td>Eve</td>
  <td>Jackson</td>
  <td>57</td>
</tr>
</table>

```

Note: The value of the **colspan** attribute represents the number of columns to span.

Name		Age
Jill	Smith	43
Eve	Jackson	57

HTML Table - Rowspan

To make a cell span over multiple rows, use the **rowspan** attribute:

```

<table border="1">
  <tr>
    <th>Name</th>
    <td>Jill</td>
  </tr>
  <tr>
    <th rowspan="2">Phone</th>
    <td>555-1234</td>
  </tr>
  <tr>
    <td>555-8745</td>
  </tr>
</table>

```

Note: The value of the **rowspan** attribute represents the number of rows to span.

Name	Jill
Phone	555-1234
	555-8745

The <base> element

```

<!DOCTYPE html>

```

```

<html>

```

```

<head>

```

```
<base href="https://www.w3schools.com/" target="_blank">
```

```
</head>
```

```
<body>
```

```
<h1>The base element</h1>
```

<p><imgsrc="images/stickman.gif" width="24" height="39" alt="Stickman"> - Notice that we have only specified a relative address for the image. Since we have specified a base URL in the head section, the browser will look for the image at "https://www.w3schools.com/images/stickman.gif".</p>

<p>HTML base tag - Notice that the link opens in a new window, even if it has no target="_blank" attribute. This is because the target attribute of the base element is set to "_blank".</p>

```
</body>
```

```
</html>
```

Output:

The base element



- Notice that we have only specified a relative address for the image. Since we have specified a base URL in the head section, the browser will look for the image at "https://www.w3schools.com/images/stickman.gif".

[HTML base tag](#) - Notice that the link opens in a new window, even if it has no target="_blank" attribute. This is because the target attribute of the base element is set to "_blank".

Definition and Usage

The **<base>** tag specifies the base URL and/or target for all relative URLs in a document.

The **<base>** tag must have either an href or a target attribute present, or both.

There can only be one single **<base>** element in a document, and it must be inside the <head> element.

<link> tag

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

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

</head>

<body>

<h1>Hello World!</h1>

<h2>I am formatted with a linked style sheet.</h2>

<p>Me too!</p>

</body>

</html>
```



Definition and Usage

The `<link>` tag defines the relationship between the current document and an external resource.

The `<link>` tag is most often used to link to external style sheets or to add a [favicon](#) to your website.

The `<link>` element is an empty element, it contains attributes only.

Text Formatting TAGS

`<P>` Tags

The paragraph defines whole textual information and is also a part of HTML text formatting elements. Of course, you can have more than one paragraph on your website! Bear in mind that every new paragraph you write begins from a new line.

Bold Text

Anything that appears within `...` element, is displayed in bold as shown below –

Example

```
<!DOCTYPE html>

<html>

<head>

<title>Bold Text Example</title>

</head>

<body>

<p>The following word uses a <b>bold</b>typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses a **bold** typeface.

Italic Text

Anything that appears within `<i>...</i>` element is displayed in italicized as shown below –

Example

```
<!DOCTYPE html>

<html>

<head>

<title>Italic Text Example</title>

</head>

<body>

<p>The following word uses an <i>italicized</i>typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses *aitalized* typeface.

Underlined Text

Anything that appears within `<u>...</u>` element, is displayed with underline as shown below –

Example

```
<!DOCTYPE html>
```

```
<html>

<head>

<title>Underlined Text Example</title>

</head>

<body>

<p>The following word uses an <u>underlined</u>typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses aunderlined typeface.

Strike Text

Anything that appears within **<strike>...</strike>** element is displayed with strikethrough, which is a thin line through the text as shown below –

Example

```
<!DOCTYPE html>

<html>

<head>

<title>Strike Text Example</title>

</head>

<body>

<p>The following word uses a <strike>strikethrough</strike>typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses a ~~strikethrough~~ typeface.

Monospaced Font

The content of a **<tt>...</tt>** element is written in monospaced font. Most of the fonts are known as variable-width fonts because different letters are of different widths (for example, the letter 'm' is wider than the letter 'i'). In a monospaced font, however, each letter has the same width.

Example

```
<!DOCTYPE html>
```

```
<html>

<head>

<title>Monospaced Font Example</title>

</head>

<body>

<p>The following word uses a <tt>monospaced</tt>typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses a mo no spaced typeface.

Superscript Text

The content of a **^{...}** element is written in superscript; the font size used is the same size as the characters surrounding it but is displayed half a character's height above the other characters.

Example

```
<!DOCTYPE html>

<html>

<head>

<title>Superscript Text Example</title>

</head>

<body>

<p>The following word uses a <sup>superscript</sup>typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses a ^{superscript} typeface.

Subscript Text

The content of a **_{...}** element is written in subscript; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.

Example

```
<!DOCTYPE html>

<html>
```

```
<head>

<title>Subscript Text Example</title>

</head>

<body>

<p>The following word uses a <sub>subscript</sub>typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses a _{subscript} typeface.

Inserted Text

Anything that appears within **<ins>...</ins>** element is displayed as inserted text.

Deleted Text

Anything that appears within **...** element, is displayed as deleted text.

Example

```
<!DOCTYPE html>

<html>

<head>

<title>Inserted Text Example</title>

</head>

<body>

<p>I want to drink <del>cola</del><ins>wine</ins></p>

</body>

</html>
```

This will produce the following result –I want to drink ~~cola~~wine

Larger Text

The content of the **<big>...</big>** element is displayed one font size larger than the rest of the text surrounding it as shown below –

Example

```
<!DOCTYPE html>

<html>

<head>

<title>Larger Text Example</title>

</head>

<body>

<p>The following word uses a <b>big</b> typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses a **big** typeface.

Smaller Text

The content of the `<small>...</small>` element is displayed one font size smaller than the rest of the text surrounding it as shown below –

Example

```
<!DOCTYPE html>

<html>

<head>

<title>Smaller Text Example</title>

</head>

<body>

<p>The following word uses a <small>small</small> typeface.</p>

</body>

</html>
```

This will produce the following result –The following word uses a small typeface.

<q> tags

The `<q>` element inserts quotation marks around the text. The full name of this tag is “quote” so, it is used for inserting quotes, citing and so on. However, if you’re not quoting some other text,

you shouldn't use this tag just for adding quotation marks. You can insert some attributes to this element for suiting your needs.

```
<!DOCTYPE html>

<html>

<body>

<p>WWF's goal is to:

<q>Build a future where people live in harmony with nature.</q>

We hope they succeed.</p>

</body>

</html>
```

WWF's goal is to: “Build a future where people live in harmony with nature.” We hope they succeed.

<blockquote> tags

The `<blockquote>` element is quite similar to `<q>` tag, and it works almost in the same method. However, while using this HTML text formatting tag, bear in mind, that this quote will appear on a new line and have an indent for each line it contains. It makes it look like a block of text, and this is where its name came from.

```
<!DOCTYPE html>

<html>

<body>

<h1>About WWF</h1>

<p>Here is a quote from WWF's website:</p>

<blockquote cite="http://www.worldwildlife.org/who/index.html">

For 50 years, WWF has been protecting the future of nature. The world's leading conservation organization, WWF works in 100 countries and is supported by 1.2 million members in the United States and close to 5 million globally.

</blockquote>

</body>

</html>
```

<marquee> Tag

The HTML `<marquee>` tag is used for scrolling piece of text or image displayed either horizontally across or vertically down your web site page depending on the settings.

Example

```
<!DOCTYPE html>

<html>

<head>

<title>HTML marquee Tag</title>

</head>

<body>

<marquee>This is basic example of marquee</marquee>

<marqueedirection="up">The direction of text will be from bottom to top.</marquee>

</body>

</html>
```

Specific Attributes

The HTML <marquee> tag also supports the following additional attributes –

Attribute	Value	Description
behavior	scroll slide alternate	Defines the type of scrolling.
Bgcolor	rgb(x,x,x) #xxxxxx colorname	<i>Deprecated</i> – Defines the direction of scrolling the content.
direction	up down left right	Defines the direction of scrolling the content.
Height	pixels or %	Defines the height of marquee.
Hspace	Pixels	Specifies horizontal space around the marquee.
Loop	Number	Specifies how many times to loop. The default value is INFINITE, which means that the marquee loops endlessly.
scrolldelay	Seconds	Defines how long to delay between each jump.
scrollamount	Number	Defines how how far to jump.
Width	pixels or %	Defines the width of marquee.
Vspace	Pixels	Specifies vertical space around the marquee.

HTML Block and Inline Elements

<header> Tag

The <header> element represents a container for introductory content or a set of navigational links.

A <header> element typically contains:

- one or more heading elements (<h1> - <h6>)
- logo or icon
- authorship information

You can have several <header> elements in one document.

```
<!DOCTYPE html>

<html>

<body>

<article>

<header>

<h1>Most important heading here</h1>

<h3>Less important heading here</h3>

<p>Some additional information here.</p>

</header>

<p>LoremIpsumdolor set amet...</p>

</article>

</body>

</html>
```

Most important heading here

Less important heading here

Some additional information here.

LoremIpsumdolor set amet....

<footer> Tag

The <footer> tag defines a footer for a document or section.

A <footer> element should contain information about its containing element.

A <footer> element typically contains:

- authorship information
- copyright information
- contact information
- sitemap
- back to top links
- related documents

You can have several <footer> elements in one document.

```
<!DOCTYPE html>

<html>

<body>

<footer>

<p>Posted by: HegeRefsnes</p>

<p>Contact information: <a href="mailto:someone@example.com">someone@example.com</a>.</p>

</footer>

<p><strong>Note:</strong> The footer tag is not supported in Internet Explorer 8 and earlier versions.</p>

</body>

</html>
```

Posted by: HegeRefsnes

Contact information: someone@example.com.

Note: The footer tag is not supported in Internet Explorer 8 and earlier versions.

<h1> to <h6> Tags

The <h1> to <h6> tags are used to define HTML headings.

<h1> defines the most important heading. <h6> defines the least important heading.

Attributes

Attribute	Value	Description
align	left center right justify	Not supported in HTML5. Specifies the alignment of a heading

```
<!DOCTYPE html>

<html>

<body>

<h1>This is heading 1</h1>

<h2>This is heading 2</h2>

<h3>This is heading 3</h3>

<h4>This is heading 4</h4>

<h5>This is heading 5</h5>

<h6>This is heading 6</h6>

</body>

</html>
```

This is heading 1

This is heading 2

This is heading 3

This is heading 4

This is heading 5

This is heading 6

<hr> Tag

The <hr> tag defines a thematic break in an HTML page. The <hr> element is used to separate content (or define a change) in an HTML page.

Attributes

Attribute	Value	Description
align	left center right	Not supported in HTML5. Specifies the alignment of a <hr> element

noshade	Noshade	Not supported in HTML5.	Specifies that a <hr> element should render in one solid color (noshaded), instead of a shaded color
size	<i>Pixels</i>	Not supported in HTML5.	Specifies the height of a <hr> element
width	<i>pixels</i> %	Not supported in HTML5.	Specifies the width of a <hr> element

```

<!DOCTYPE html>

<html>

<body>

<h1>HTML</h1>

<p>HTML is a language for describing web pages.</p>

<hr>

<h1>CSS</h1>

<p>CSS defines how to display HTML elements.</p>

</body>

</html>

```

HTML

HTML is a language for describing web pages.

CSS

CSS defines how to display HTML elements.

<pre> Tag

The <pre> tag defines preformatted text.

Text in a <pre> element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks.

Attributes

Attribute	Value	Description
width	<i>number</i>	Not supported in HTML5.

Specifies the maximum number of characters per line

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<pre>
```

Text in a pre element

is displayed in a fixed-width

font, and it preserves

both spaces and

line breaks

```
</pre>
```

```
</body>
```

```
</html>
```

Text in a pre element
is displayed in a fixed-width
font, and it preserves
both spaces and
line breaks

<section> Tag

The <section> tag defines sections in a document, such as chapters, headers, footers, or any other sections of the document.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<section>
```

```
<h1>WWF</h1>
```

```
<p>The World Wide Fund for Nature (WWF) is an international organization working on issues regarding the conservation, research and restoration of the environment, formerly named the World Wildlife Fund. WWF was founded in 1961.</p>
```

```
</section>
```

```
<section>
```

```
<h1>WWF's Panda symbol</h1>
```

```
<p>The Panda has become the symbol of WWF. The well-known panda logo of WWF originated from a panda named Chi Chi that was transferred from the Beijing Zoo to the London Zoo in the same year of the establishment of WWF.</p>
```

```
</section>
```

```
</body>
```

```
</html>
```

WWF

The World Wide Fund for Nature (WWF) is an international organization working on issues regarding the conservation, research and restoration of the environment, formerly named the World Wildlife Fund. WWF was founded in 1961.

WWF's Panda symbol

The Panda has become the symbol of WWF. The well-known panda logo of WWF originated from a panda named Chi Chi that was transferred from the Beijing Zoo to the London Zoo in the same year of the establishment of WWF.

<abbr> Tag

The <abbr> tag defines an abbreviation or an acronym, like "Mr.", "Dec.", "ASAP", "ATM".

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>The <abbr title="World Health Organization">WHO</abbr> was founded in 1948.</p>
```

```
</body>
```

```
</html>
```

The WHO was founded in 1948.

HTML lists

Lists are used to group together related pieces of information so they are clearly associated with each other and easy to read. In modern web development, lists are workhorse elements, frequently used for navigation as well as general content.

Lists are good from a structural point of view as they help create a well-structured, more accessible, easy-to-maintain document. They are also useful because they provide specialized elements to which you can attach CSS styles. Finally, semantically correct lists help visitors read your web site, and they simplify maintenance when your pages need to be updated.

The three list types

There are three list types in HTML:

unordered list — used to group a set of related items in no particular order

ordered list — used to group a set of related items in a specific order

description list — used to display name/value pairs such as terms and definitions

Each list type has a specific purpose and meaning in a web page.

Unordered lists

Unordered (bulleted) lists are used when a set of items can be placed in any order. An example is a shopping list:

- milk
- bread
- butter
- coffee beans

Unordered lists use one set of tags wrapped around one or more sets of tags:

```
<ul>
<li>bread</li>
<li>coffee beans</li>
<li>milk</li>
<li>butter</li>
</ul>
```

Attribute	Value	Description
compact	Compact	Not supported in HTML5. Specifies that the list should render smaller than normal
type	disc square circle	Not supported in HTML5. Specifies the kind of marker to use in the list

Ordered lists

Ordered (numbered) lists are used to display a list of items that should be in a specific order. An example would be cooking instructions:

1. Gather ingredients
2. Mix ingredients together
3. Place ingredients in a baking dish
4. Bake in oven for an hour
5. Remove from oven
6. Allow to stand for ten minutes
7. Serve

```
<ol>
<li>Gather ingredients</li>
<li>Mix ingredients together</li>
<li>Place ingredients in a baking dish</li>
<li>Bake in oven for an hour</li>
<li>Remove from oven</li>
<li>Allow to stand for ten minutes</li>
<li>Serve</li>
</ol>
```

Attribute	Value	Description
compact	Compact	Not supported in HTML5. Specifies that the list should render smaller than normal

start	<i>Number</i>	Specifies the start value of an ordered list
type	1 A a I i	Specifies the kind of marker to use in the list

Ordered lists can be displayed with several sequencing options. The default in most browsers is decimal numbers, but there are others available:

- Letters
 - Lowercase ascii letters (a, b, c...)
 - Uppercase ascii letters (A, B, C...).
 - Lowercase classical Greek: (έ, ή, ί...)
- Numbers
 - Decimal numbers (1, 2, 3...)
 - Decimal numbers with leading zeros (01, 02, 03...)
 - Lowercase Roman numerals (i, ii, iii...)
 - Uppercase Roman numerals (I, II, III...)
 - Traditional Georgian numbering (an, ban, gan...)
 - Traditional Armenian numbering (mek, yerku, yerek...)

Description lists

Description lists (previously called *definition lists*, but renamed in HTML5) associate specific names and values within a list. Examples might be items in an ingredient list and their descriptions, article authors and brief bios, or competition winners and the years in which they won. You can have as many name-value groups as you like, but there must be at least one name and at least one value in each pair.

Description lists use one set of `<dl></dl>` tags wrapped around one or more groups of `<dt></dt>` (name) and `<dd></dd>` (value) tags. You must pair at least one `<dt></dt>` with at least one `<dd></dd>`, and the `<dt></dt>` should always come first in the source order.

```
<dl>
<dt>Name1</dt>
<dd>Value that applies to Name1</dd>
<dt>Name2</dt>
<dt>Name3</dt>
<dd>Value that applies to both Name2 and Name3</dd>
<dt>Name4</dt>
<dd>One value that applies to Name4</dd>
<dd>Another value that applies to Name4</dd>
</dl>
```

Result:

Name1

Value that applies to Name1

Name2

Name3

Value that applies to both Name2 and Name3

Name4

One value that applies to Name4

Another value that applies to Name4

HTML list advantages

- Flexibility: If you have to change the order of the list items in an ordered list, you simply move around the list item elements; when the browser renders the list, it will be properly ordered.
- Styling: Using an HTML list allows you to style the list properly using CSS. The list item tags `` are different from the other tags in your document, so you can specifically target CSS rules to them.
- Semantics: HTML lists give the content the proper semantic structure. This has important benefits, such as allowing screen readers to tell users with visual impairments that they are reading a list, rather than just reading out a confusing jumble of text and numbers.

To put it another way, don't code list items using regular text tags. Using text instead of a list makes more work for you and can create problems for your document's readers. So if your document needs a list, you should use the correct HTML list format.

HTML Links - Hyperlinks

HTML links are hyperlinks. You can click on a link and jump to another document. When you move the mouse over a link, the mouse arrow will turn into a little hand.

Creating a links:

- Links are created in a web page by using the `<A>` and `` tag.
- Anything returns between the `<a>` compared tag becomes hyperlink or hotspot.
- Documents to be navigated needs to be specified.
- By using the HREF attribute of the `<A>`, the next navigation web page or image can be specified.
``

Types of links: There are three types of links:

- 1) Links to an external document: ``
- 2) Links to a specified place within the same document: ``
- 3) Link to a particular file on a particular position: ``

Attribute	Value	Description
crossorigin	anonymous use-credentials	Specifies how the element handles cross-origin requests
href	<i>URL</i>	Specifies the location of the linked document
hreflang	<i>language_code</i>	Specifies the language of the text in the linked document
media	<i>media_query</i>	Specifies on what device the linked document will be displayed
rel	alternate author dns-prefetch help icon license next pingback preconnect prefetch preload prerender prev search stylesheet	Required. Specifies the relationship between the current document and the linked document

<u>sizes</u>	<i>HeightxWidth</i> any	Specifies the size of the linked resource. Only for rel="icon"
<u>target</u>	<i>_blank</i> <i>_self</i> <i>_top</i> <i>_parent</i> <i>frame_name</i>	Not supported in HTML5. Specifies where the linked document is to be loaded
<u>type</u>	<i>media_type</i>	Specifies the media type of the linked document

```

<!DOCTYPE html>

<html>

<body>

<h2>HTML Links</h2>

<a href="Html1.html">Visit our HTML tutorial</a>

</body>

</html>

```

HTML Links

[Visit our HTML tutorial](#)

Using the #id selector / Bookmark in HTML

In CSS, "#id" is a selector that may be used to designate an area that a link should point to, similar to anchor in HTML. In the following example, you'll see how to apply #id to an HTML tag, and then how to link to it. This example will link to the first paragraph at the top of this page.

```

<!DOCTYPE html>

<html>

<body>

<p><a href="#C4">Jump to Chapter 4</a></p>

<h2>Chapter 1</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 2</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 3</h2>

<p>This chapter explains bablabla</p>

<h2 id="C4">Chapter 4</h2>

```

```
<p>This chapter explains bablabla</p>

<h2>Chapter 5</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 6</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 7</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 8</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 9</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 10</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 11</h2>

<p>This chapter explains bablabla</p>

<h2>Chapter 12</h2>

<p>This chapter explains bablabla</p>

</body>

</html>
```

[Jump to Chapter 4](#)

Chapter 1

This chapter explains bablabla

Chapter 2

This chapter explains bablabla

Chapter 3

This chapter explains bablabla

Chapter 4

This chapter explains bablabla

Chapter 5

This chapter explains bablabla

Chapter 6

This chapter explains bablabla

Chapter 7

This chapter explains bablabla

Chapter 8

This chapter explains bablabla

Chapter 9

This chapter explains bablabla

Chapter 10

This chapter explains bablabla

Chapter 11

This chapter explains bablabla

Chapter 12

This chapter explains bablabla

Defining an HTML Table

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

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

The `<tr>` element defines a table row, the `<th>` element defines a table header, and the `<td>` element defines a table cell.

A more complex HTML table may also include `<caption>`, `<col>`, `<colgroup>`, `<thead>`, `<tfoot>`, and `<tbody>` elements.

Attributes

Attribute	Value	Description
align	left center right	Not supported in HTML5. Specifies the alignment of a table according to surrounding text
Valign	Top Middle Bottom	Controls the vertical alignment of cell contain it acceptor the value
bgcolor	<i>rgb(x,x,x)</i> <i>#xxxxxx</i> <i>colorname</i>	Not supported in HTML5. Specifies the background color for a table
border	1 0	Not supported in HTML5. Specifies whether or not the table is being used for layout purposes
cellpadding	<i>Pixels</i>	Not supported in HTML5. Specifies the space between the cell wall and the cell content
cellspacing	<i>Pixels</i>	Not supported in HTML5. Specifies the space between cells
size	Numeric value	Specifies the Horizontal size of the textbox.
maxlength	Numeric value	Specifies the Maximum Number of Character that user can enter.
Cheked	-	Not supported in HTML5. This attribute should be used only for check box and Radio Button from control. It indicates checked status.
width	<i>pixels</i> <i>%</i>	Not supported in HTML5. Specifies the width of a table
DISABLE	-	Turn off a from control
READONLY	-	Prevent the form controls value from being chang.

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<table>
```

```
<tr>

<th>Month</th>

<th>Savings</th>

</tr>

<tr>

<td>January</td>

<td>$100</td>

</tr>

<tr>

<td>February</td>

<td>$80</td>

</tr>

</table>

</body>

</html>
```

HTML Forms

An HTML form contains **form elements**.——

Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more.

The <input> Element

The `<input>` element is the most important form element.

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

Here are some examples:

Type	Description
<code><input type="text"></code>	Defines a one-line text input field
<code><input type="radio"></code>	Defines a radio button (for selecting one of many choices)
<code><input type="submit"></code>	Defines a submit button (for submitting the form)

Attribute Values

Value	Description
button	Defines a clickable button (mostly used with a JavaScript to activate a script)
checkbox	Defines a checkbox
color	Defines a color picker
date	Defines a date control (year, month, day (no time))
datetime-local	Defines a date and time control (year, month, day, time (no timezone))
email	Defines a field for an e-mail address
file	Defines a file-select field and a "Browse" button (for file uploads)
hidden	Defines a hidden input field
image	Defines an image as the submit button
month	Defines a month and year control (no timezone)
number	Defines a field for entering a number
password	Defines a password field
radio	Defines a radio button
range	Defines a range control (like a slider control)
reset	Defines a reset button
search	Defines a text field for entering a search string
submit	Defines a submit button
tel	Defines a field for entering a telephone number
text	Default. Defines a single-line text field
time	Defines a control for entering a time (no timezone)
url	Defines a field for entering a URL
week	Defines a week and year control (no timezone)

The Action Attribute

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

Normally, the form data is sent to a web page on the server when the user clicks on the submit button.

In the example above, the form data is sent to a page on the server called `"/action_page.php"`. This page contains a server-side script that handles the form data:

```
<form action="/Page1.html">
```

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

The Target Attribute

The `target` attribute specifies if the submitted result will open in a new browser tab, a frame, or in the current window.

The default value is "`_self`" which means the form will be submitted in the current window.

To make the form result open in a new browser tab, use the value "`_blank`":

```
<!DOCTYPE html>

<html>

<body>

<h2>The target Attribute</h2>

<p>When submitting this form, the result will be opened in a new browser tab:</p>

<form action="/Page1.html" target="_blank">

  First name:<br>

  <input type="text" name="firstname" value="Mickey">

  <br>

  Last name:<br>

  <input type="text" name="lastname" value="Mouse">

  <br><br>

  <input type="submit" value="Submit">

</form>

</body>

</html>
```

The target Attribute

When submitting this form, the result will be opened in a new browser tab:

First name:

Last name:

The Method Attribute

The **method** attribute specifies the HTTP method (**GET** or **POST**) to be used when submitting the form data:

```
<!DOCTYPE html>

<html>

<body>

<h2>The method Attribute</h2>

<p>This form will be submitted using the GET method:</p>

<form action="/Page1.html" target="_blank" method="GET">

  First name:<br>

  <input type="text" name="firstname" value="Mickey">

  <br>

  Last name:<br>

  <input type="text" name="lastname" value="Mouse">

  <br><br>

  <input type="submit" value="Submit">

</form>

<p>After you submit, notice that the form values is visible in the address bar of the new browser tab.</p>

</body>

</html>
```

When to Use GET?

The default method when submitting form data is GET.

However, when GET is used, the submitted form data will be **visible in the page address field**:

/action_page.php?firstname=Mickey&lastname=Mouse

When to Use POST?

Always use POST if the form data contains sensitive or personal information. The POST method does not display the submitted form data in the page address field.

The Name Attribute

Each input field must have a **name** attribute to be submitted.

If the **name** attribute is omitted, the data of that input field will not be sent at all.

Additional attributes Section

In addition to the attributes that operate on all `<input>` elements regardless of their type, email inputs support the following attributes:

Attribute	Description
maxlength	The maximum number of characters the input should accept
minlength	The minimum number of characters long the input can be and still be considered valid
multiple	Whether or not to allow multiple, comma-separated, e-mail addresses to be entered
pattern	A regular expression the input's contents must match in order to be valid
placeholder	An exemplar value to display in the input field whenever it is empty
readonly	A Boolean attribute indicating whether or not the contents of the input should be read-only
size	A number indicating how many characters wide the input field should be
spellcheck	Controls whether or not to enable spell checking for the input field, or if the default spell checking configuration should be used

HTML `<textarea>`

```
<!DOCTYPE html>

<html>

<body>

<form action="/Page1.html" id="usrform">

  Name: <input type="text" name="username">

  <input type="submit">

</form>

<br>

<textarea rows="4" cols="50" name="comment" form="usrform">

Enter text here...</textarea>

<p>The text area above is outside the form element, but should still be a part of the form.</p>

<p><b>Note:</b> The form attribute is not supported in IE.</p>

</body>

</html>
```

Name:

Enter text here...

The text area above is outside the form element, but should still be a part of the form.

Note: The form attribute is not supported in IE.

HTML <option> selected Attribute

The selected attribute is a boolean attribute.

When present, it specifies that an option should be pre-selected when the page loads.

The pre-selected option will be displayed first in the drop-down list.

```
<!DOCTYPE html>

<html>

<body>

<select>

<option value="volvo">Volvo</option>

<option value="saab">Saab</option>

<option value="vw">VW</option>

<option value="audi" selected>Audi</option>

</select>

</body>

</html>
```

HTML <button> form action Attribute

The form action attribute specifies where to send the form-data when a form is submitted. This attribute overrides the form's [action](#) attribute.

The form action attribute is only used for buttons with type="submit".

```
<!DOCTYPE html>

<html>

<body>
```

```

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

  First name: <input type="text" name="fname"><br>

  Last name: <input type="text" name="lname"><br>

  <button type="submit">Submit</button><br>

  <button type="submit" formaction="/action_page2.php">Submit to another page</button>

</form>

</body>

</html>

```

HTML <link> target Attribute

Syntax

```
<link target="_blank|_self|_parent|_top|framename">
```

Attribute Values

Value	Description
<code>_blank</code>	Load in a new window
<code>_self</code>	Load in the same frame as it was clicked
<code>_parent</code>	Load in the parent frameset
<code>_top</code>	Load in the full body of the window
<code>framename</code>	Load in a named frame

```

<head>
  <link rel="parent" href="wildcats.htm" target="_blank">
</head>

```

HTML Images

In HTML, images are defined with the `` tag.

The `` tag is empty, it contains attributes only, and does not have a closing tag.

The `src` attribute specifies the URL (web address) of the image:

``

Attribute	Value	Description
align	top bottom middle left right	Not supported in HTML5. Specifies the alignment of an image according to surrounding elements
alt	<i>Text</i>	Specifies an alternate text for an image
border	<i>Pixels</i>	Not supported in HTML5. Specifies the width of the border around an image
crossorigin	anonymous use-credentials	Allow images from third-party sites that allow cross-origin access to be used with canvas
height	<i>Pixels</i>	Specifies the height of an image
hspace	<i>Pixels</i>	Not supported in HTML5. Specifies the whitespace on left and right side of an image
ismap	Ismap	Specifies an image as a server-side image-map
longdesc	<i>URL</i>	Specifies a URL to a detailed description of an image
sizes		Specifies image sizes for different page layouts
src	<i>URL</i>	Specifies the URL of an image
srcset	<i>URL</i>	Specifies the URL of the image to use in different situations
usemap	<i>#mapname</i>	Specifies an image as a client-side image-map
vspace	<i>Pixels</i>	Not supported in HTML5. Specifies the whitespace on top and bottom of an image
width	<i>Pixels</i>	Specifies the width of an image

HTML5 - AUDIO & VIDEO

HTML5 features include native audio and video support without the need for Flash.

The HTML5 `<audio>` and `<video>` tags make it simple to add media to a website. You need to set **src** attribute to identify the media source and include a controls attribute so the user can play and pause the media.

Embedding Video

Here is the simplest form of embedding a video file in your webpage –

```
<video src = "foo.mp4" width = "300" height = "200" controls>
  Your browser does not support the <video> element.
</video>
```

The current HTML5 draft specification does not specify which video formats browsers should support in the video tag. But most commonly used video formats are –

- **Ogg** – Ogg files with Theora video codec and Vorbis audio codec.
- **mpeg4** – MPEG4 files with H.264 video codec and AAC audio codec.

You can use <source> tag to specify media along with media type and many other attributes. A video element allows multiple source elements and browser will use the first recognized format –

```
<!DOCTYPE HTML>
<html>
<body>
<videowidth="300"height="200"controlsautoplay>
<sourcesrc="/html5/foo.ogg"type="video/ogg"/>
<sourcesrc="/html5/foo.mp4"type="video/mp4"/>
  Your browser does not support the <video> element.
</video>
</body>
</html>
```

This will produce the following result –

Video Attribute Specification

The HTML5 video tag can have a number of attributes to control the look and feel and various functionalities of the control –

Sr.No.	Attribute & Description
1	Autoplay This Boolean attribute if specified, the video will automatically begin to play back as soon as it can do so without stopping to finish loading the data.
2	Autobuffer This Boolean attribute if specified, the video will automatically begin buffering even if it's not set to automatically play.
3	Controls If this attribute is present, it will allow the user to control video playback, including volume, seeking, and pause/resume playback.

4	Height This attribute specifies the height of the video's display area, in CSS pixels.
5	Loop This Boolean attribute if specified, will allow video automatically seek back to the start after reaching at the end.
6	Preload This attribute specifies that the video will be loaded at page load, and ready to run. Ignored if autoplay is present.
7	Poster This is a URL of an image to show until the user plays or seeks.
8	Src The URL of the video to embed. This is optional; you may instead use the <source> element within the video block to specify the video to embed.
9	Width This attribute specifies the width of the video's display area, in CSS pixels.

Embedding Audio

HTML5 supports <audio> tag which is used to embed sound content in an HTML or XHTML document as follows.

```
<audio src = "foo.wav" controls autoplay>
  Your browser does not support the <audio> element.
</audio>
```

The current HTML5 draft specification does not specify which audio formats browsers should support in the audio tag. But most commonly used audio formats are **ogg**, **mp3** and **wav**.

You can use tag to specify media along with media type and many other attributes. An audio element allows multiple source elements and browser will use the first recognized format –

```
<!DOCTYPE HTML>

<html>
<body>
<audiocontrolsautoplay>
<sourcesrc="/html5/audio.ogg"type="audio/ogg"/>
<sourcesrc="/html5/audio.wav"type="audio/wav"/>
  Your browser does not support the <audio> element.
</audio>
</body>
</html>
```

This will produce the following result –

Audio Attribute Specification

The HTML5 audio tag can have a number of attributes to control the look and feel and various functionalities of the control –

Sr.No.	Attribute & Description
1	Autoplay This Boolean attribute if specified, the audio will automatically begin to play back as soon as it can do so without stopping to finish loading the data.
2	Autobuffer This Boolean attribute if specified, the audio will automatically begin buffering even if it's not set to automatically play.
3	Controls If this attribute is present, it will allow the user to control audio playback, including volume, seeking, and pause/resume playback.
4	Loop This Boolean attribute if specified, will allow audio automatically seek back to the start after reaching at the end.
5	Preload This attribute specifies that the audio will be loaded at page load, and ready to run. Ignored if autoplay is present.
6	Src The URL of the audio to embed. This is optional; you may instead use the <source> element within the video block to specify the video to embed.

Handling Media Events

The HTML5 audio and video tag can have a number of attributes to control various functionalities of the control using JavaScript –

S.No.	Event & Description
1	Abort This event is generated when playback is aborted.
2	Canplay This event is generated when enough data is available that the media can be played.
3	Ended This event is generated when playback completes.
4	Error This event is generated when an error occurs.
5	Loadeddata This event is generated when the first frame of the media has finished loading.
6	Loadstart This event is generated when loading of the media begins.
7	Pause

	This event is generated when playback is paused.
8	Play This event is generated when playback starts or resumes.
9	Progress This event is generated periodically to inform the progress of the downloading the media.
10	Ratechange This event is generated when the playback speed changes.
11	Seeked This event is generated when a seek operation completes.
12	Seeking This event is generated when a seek operation begins.
13	Suspend This event is generated when loading of the media is suspended.
14	Volumechange This event is generated when the audio volume changes.
15	Waiting This event is generated when the requested operation such as playback is delayed pending the completion of another operation.

Following is the example which allows to play the given video –

```

<!DOCTYPE HTML>

<html>
<head>
</head>
<body>
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>

</body>
</html>

```

This will produce the following result –

Configuring Servers for Media Type

Most servers don't by default serve Ogg or mp4 media with the correct MIME types, so you'll likely need to add the appropriate configuration for this.

```

AddType audio/ogg .oga
AddType audio/wav .wav
AddType video/ogg .ogv .ogg

```


AddType video/mp4 .mp4

CSS

CSS is the language we use to style a Web page.

What is CSS?

CSS stands for Cascading Style Sheets

CSS describes how HTML elements are to be displayed on screen, paper, or in other media

CSS saves a lot of work. It can control the layout of multiple web pages all at once

External stylesheets are stored in CSS files

CSS is the language we use to style a Web page.

What is CSS

CSS stands for Cascading Style Sheets. It is a style sheet language which is used to describe the look and formatting of a document written in markup language. It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces. It can also be used with any kind of XML documents including plain XML, SVG and XUL.

CSS is used along with HTML and JavaScript in most websites to create user interfaces for web applications and user interfaces for many mobile applications.

What does CSS do

You can add new looks to your old HTML documents.

You can completely change the look of your website with only a few changes in CSS code.

Why use CSS

These are the three major benefits of CSS:

1) Solves a big problem

Before CSS, tags like font, color, background style, element alignments, border and size had to be repeated on every web page. This was a very long process. For example: If you are developing a large website where fonts and color information are added on every single page, it will become a long and expensive process. CSS was created to solve this problem. It was a W3C recommendation.

2) Saves a lot of time

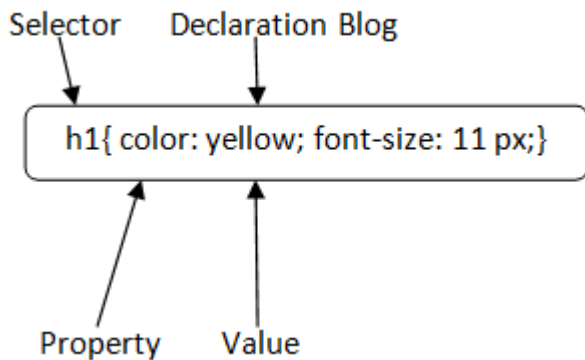
CSS style definitions are saved in external CSS files so it is possible to change the entire website by changing just one file.

3) Provide more attributes

CSS provides more detailed attributes than plain HTML to define the look and feel of the website.

CSS Syntax

A CSS rule set contains a selector and a declaration block.



Selector: Selector indicates the HTML element you want to style. It could be any tag like `<h1>`, `<title>` etc.

Declaration Block: The declaration block can contain one or more declarations separated by a semicolon. For the above example, there are two declarations:

1. `color: yellow;`
2. `font-size: 11 px;`

Each declaration contains a property name and value, separated by a colon.

Property: A Property is a type of attribute of HTML element. It could be color, border etc.

Value: Values are assigned to CSS properties. In the above example, value "yellow" is assigned to color property.

`Selector{Property1: value1; Property2: value2;;}`

CSS Selector

CSS selectors are used *to select the content you want to style*. Selectors are the part of CSS rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.

There are several different types of selectors in CSS.

1. CSS Element Selector
2. CSS Id Selector
3. CSS Class Selector
4. CSS Universal Selector
5. CSS Group Selector

1. CSS Element Selector

The element selector selects the HTML element by name.

```
<!DOCTYPE html>
<html>
<head>
<style>
p{
    text-align: center;
    color: blue;
}
</style>
</head>
<body>
<p>This style will be applied on every paragraph.</p>

<p>And me!</p>
</body>
</html>
```

Output:

This style will be applied on every paragraph.

Me too!

And me!

2) CSS Id Selector

The id selector selects the id attribute of an HTML element to select a specific element. An id is always unique within the page so it is chosen to select a single, unique element.

It is written with the hash character (#), followed by the id of the element.

Let's take an example with the id "para1".

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
    text-align: center;
    color: blue;
}
</style>
```

```
</head>
<body>
<p id="para1">Hello Javatpoint.com</p>
<p>This paragraph will not be affected.</p>
</body>
</html>
```

Output:

Hello Javatpoint.com

This paragraph will not be affected.

3) CSS Class Selector

The class selector selects HTML elements with a specific class attribute. It is used with a period character . (full stop symbol) followed by the class name.

Note: A class name should not be started with a number.

Let's take an example with a class "center".

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
    text-align: center;
    color: blue;
}
</style>
</head>
<body>
<h1 class="center">This heading is blue and center-aligned.</h1>
<p class="center">This paragraph is blue and center-aligned.</p>
</body>
</html>
```

Output:

This heading is blue and center-aligned.

This paragraph is blue and center-aligned.

CSS Class Selector for specific element

If you want to specify that only one specific HTML element should be affected then you should use the element name with class selector.

Let's see an example.

```
<!DOCTYPE html>
<html>
<head>
<style>
p.center {
  text-align: center;
  color: blue;
}
</style>
</head>
<body>
<h1 class="center">This heading is not affected</h1>
<p class="center">This paragraph is blue and center-aligned.</p>
</body>
</html>
```

OUTPUT

This heading is not affected

This paragraph is blue and center-aligned.

4) CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  color: green;
  font-size: 20px;
}
</style>
</head>
<body>
<h2>This is heading</h2>
<p>This style will be applied on every paragraph.</p>
```

```
<p id="para1">Me too!</p>
<p>And me!</p>
</body>
</html>
```

Output:

This is heading

This style will be applied on every paragraph.

Me too!

And me!

5) CSS Group Selector

The grouping selector is used to select all the elements with the same style definitions.

Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.

Let's see the CSS code without group selector.

```
h1 {
    text-align: center;
    color: blue;
}
h2 {
    text-align: center;
    color: blue;
}
p {
    text-align: center;
    color: blue;
}
```

As you can see, you need to define CSS properties for all the elements. It can be grouped in following ways:

```
h1,h2,p {
    text-align: center;
    color: blue;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
```

```
h1, h2, p {  
    text-align: center;  
    color: blue;  
}  
</style>  
</head>  
<body>  
<h1>Hello Javatpoint.com</h1>  
<h2>Hello Javatpoint.com (In smaller font)</h2>  
<p>This is a paragraph.</p>  
</body>  
</html>
```

Output:

Hello Javatpoint.com

Hello Javatpoint.com (In smaller font)

This is a paragraph.

How to add CSS

CSS is added to HTML pages to format the document according to information in the style sheet. There are three ways to insert CSS in HTML documents.

Inline CSS

Internal CSS

External CSS

1) Inline CSS

Inline CSS is used to apply CSS on a single line or element.

For example:

```
<p style="color:blue">Hello CSS</p>
```

For more visit here: [Inline CSS](#)

2) Internal CSS

Internal CSS is used to apply CSS on a single document or page. It can affect all the elements of the page. It is written inside the style tag within head section of html.

For example:

`<style>`

`p{color:blue}`

`</style>`

3) External CSS

External CSS is used to apply CSS on multiple pages or all pages. Here, we write all the CSS code in a css file. Its extension must be .css for example style.css.

For example:

`p{color:blue}`

You need to link this style.css file to your html pages like this:

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

The link tag must be used inside head section of html.

1. Inline CSS

We can apply CSS in a single element by inline CSS technique.

The inline CSS is also a method to insert style sheets in HTML document. This method mitigates some advantages of style sheets so it is advised to use this method sparingly.

If you want to use inline CSS, you should use the style attribute to the relevant tag.

Syntax:

`<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>`

Example:

`<h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>`

`<p>This paragraph is not affected.</p>`

Output:

Inline CSS is applied on this heading.

This paragraph is not affected.

Disadvantages of Inline CSS

You cannot use quotations within inline CSS. If you use quotations the browser will interpret this as an end of your style value.

These styles cannot be reused anywhere else.

These styles are tough to be edited because they are not stored at a single place.

It is not possible to style pseudo-codes and pseudo-classes with inline CSS.

Inline CSS does not provide browser cache advantages.

2. Internal CSS

The internal style sheet is used to add a unique style for a single document. It is defined in <head> section of the HTML page inside the <style> tag.

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: linen;
}
h1 {
    color: red;
    margin-left: 80px;
}
</style>
</head>
<body>
<h1>The internal style sheet is applied on this heading.</h1>
<p>This paragraph will not be affected.</p>
</body>
</html>
```

The internal style sheet is applied on this heading.

This paragraph will not be affected.

3. External CSS

The external style sheet is generally used when you want to make changes on multiple pages. It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.

It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

Example:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

The external style sheet may be written in any text editor but must be saved with a .css extension. This file should not contain HTML elements.

Let's take an example of a style sheet file named "mystyle.css".

File: mystyle.css

```
body {
    background-color: lightblue;
}
h1 {
    color: navy;
    margin-left: 20px;
}
```

Note: You should not use a space between the property value and the unit. For example: It should be margin-left:20px not margin-left:20 px.

CSS Comments

CSS comments are generally written to explain your code. It is very helpful for the users who reads your code so that they can easily understand the code.

Comments are ignored by browsers.

Comments are single or multiple lines statement and written within /*.....*/ .

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
    color: blue;
    /* This is a single-line comment */
```

```
text-align: center;
}
/* This is
a multi-line
comment */
</style>
</head>
<body>
<p>Hello Javatpoint.com</p>
<p>This statement is styled with CSS.</p>
<p>CSS comments are ignored by the browsers and not shown in the output.</p>
</body>
</html>
```

Output:

Hello Javatpoint.com

This statement is styled with CSS.

CSS comments are ignored by the browsers and not shown in the output.

NOTE:

Difference between id and class attribute: The only difference between them is that “id” is unique in a page and can only apply to at most one element, while “class” selector can apply to multiple elements.

HTML Links - Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

Note: A link does not have to be text. A link can be an image or any other HTML element!

HTML Links - Syntax

The HTML `<a>` tag defines a hyperlink. It has the following syntax:

```
<a href="url">Link text</a>
```

The most important attribute of the `<a>` element is the `href` attribute, which indicates the link's destination.

The *link text* is the part that will be visible to the reader.

Clicking on the link text, will send the reader to the specified URL address.

```
<a href="https://www.w3schools.com/">Visit W3Schools.com!</a>
```

By default, links will appear as follows in all browsers:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

Tip: Links can of course be styled with CSS, to get another look!

HTML Links - The target Attribute

By default, the linked page will be displayed in the current browser window. To change this, you must specify another target for the link.

The `target` attribute specifies where to open the linked document.

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

- `_self` - Default. Opens the document in the same window/tab as it was clicked
- `_blank` - Opens the document in a new window or tab
- `_parent` - Opens the document in the parent frame
- `_top` - Opens the document in the full body of the window

Example

Use `target="_blank"` to open the linked document in a new browser window or tab:

```
<a href="https://www.w3schools.com/" target="_blank">Visit W3Schools!</a>
```

Absolute URLs vs. Relative URLs

Both examples above are using an **absolute URL** (a full web address) in the `href` attribute.

A local link (a link to a page within the same website) is specified with a **relative URL** (without the "https://www" part):

Example

```
<h2>Absolute URLs</h2>
<p><a href="https://www.w3.org/">W3C</a></p>
<p><a href="https://www.google.com/">Google</a></p>
```

```
<h2>Relative URLs</h2>
<p><a href="html_images.asp">HTML Images</a></p>
<p><a href="/css/default.asp">CSS Tutorial</a></p>
```

CSS Background

CSS background property is used to define the background effects on element. There are 5 CSS background properties that affects the HTML elements:

background-color
background-image
background-repeat
background-attachment
background-position

Value	Description
Repeat	The background image is repeated both vertically and horizontally. The last image will be clipped if it does not fit. This is default
repeat-x	The background image is repeated only horizontally
repeat-y	The background image is repeated only vertically
no-repeat	The background-image is not repeated. The image will only be shown once

Property Values

scroll: It is the default value that prevents the element from scrolling with the contents, but scrolls with the page.

fixed: Using this value, the background image doesn't move with the element, even the element has a scrolling mechanism. It causes the image to be locked in one place, even the rest of the document scrolls.

local: Using this value, if the element has a scrolling mechanism, the background image scrolls with the content of the element.

initial: It sets the property to its default value.

CSS Font

CSS Font property is used to control the look of texts. By the use of CSS font property you can change the text size, color, style and more. You have already studied how to make text bold or underlined. Here, you will also know how to resize your font using percentage.

These are some important font attributes:

1. **CSS Font color:** This property is used to change the color of the text. (standalone attribute)
2. **CSS Font family:** This property is used to change the face of the font.
3. **CSS Font size:** This property is used to increase or decrease the size of the font.
4. **CSS Font style:** This property is used to make the font bold, italic or oblique.
5. **CSS Font variant:** This property creates a small-caps effect.
6. **CSS Font weight:** This property is used to increase or decrease the boldness and lightness of the font.

CSS font family can be divided in two types:

- Generic family: It includes Serif, Sans-serif, and Monospace.
- Font family: It specifies the font family name like Arial, New Times Roman etc.

Serif: Serif fonts include small lines at the end of characters. Example of serif: Times new roman, Georgia etc.

Sans-serif: A sans-serif font doesn't include the small lines at the end of characters. Example of Sans-serif: Arial, Verdana etc.

monospaced font, also called a **fixed-pitch**, **fixed-width**, or **non-proportional font**, is a [font](#) whose letters and characters each occupy the same amount of horizontal space. This contrasts with [variable-width fonts](#), where the letters and spacings have different widths.

Oblique type is **a form of type that slants slightly to the right, used for the same purposes as italic type.**

Font Size Value	Description
xx-small	used to display the extremely small text size.
x-small	used to display the extra small text size.
small	used to display small text size.
medium	used to display medium text size.
large	used to display large text size.
x-large	used to display extra large text size.
xx-large	used to display extremely large text size.
smaller	used to display comparatively smaller text size.
larger	used to display comparatively larger text size.
size in pixels or %	used to set value in percentage or in pixels.

CSS Border

The CSS border is a shorthand property used to set the border on an element.

The [CSS](#) border properties are use to specify the style, color and size of the border of an element. The CSS border properties are given below

- border-style
- border-color
- border-width

CSS Margins

The CSS [margin](#) properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

All the margin properties can have the following values:

- auto - the browser calculates the margin
- *length* - specifies a margin in px, pt, cm, etc.
- % - specifies a margin in % of the width of the containing element

CSS Padding

Padding is used to create space around an element's content, inside of any defined borders.

Different List Item Markers

The `list-style-type` property specifies the type of list item marker.

```
ul.a {  
  list-style-type: circle;  
}  
  
ul.b {  
  list-style-type: square;  
}  
  
ol.c {  
  list-style-type: upper-roman;  
}  
  
ol.d {  
  list-style-type: lower-alpha;  
}
```

An Image as The List Item Marker

The `list-style-image` property specifies an image as the list item marker:

```
ul {  
  list-style-image: url('sqpurple.gif');  
}
```

position The List Item Markers

The `list-style-position` property specifies the position of the list-item markers (bullet points).

"list-style-position: outside;" means that the bullet points will be outside the list item. The start of each line of a list item will be aligned vertically. This is default:

- | |
|---|
| • Coffee - A brewed drink prepared from roasted coffee beans... |
| • Tea |
| • Coca-cola |

"list-style-position: inside;" means that the bullet points will be inside the list item. As it is part of the list item, it will be part of the text and push the text at the start:

- | |
|---|
| • Coffee - A brewed drink prepared from roasted coffee beans... |
| • Tea |
| • Coca-cola |

```
ul.a {  
  list-style-position: outside;  
}
```

```
ul.b {  
  list-style-position: inside;  
}
```

CSS Text

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

```
h1 {  
  color: green;  
}
```

```
h1 {  
  text-align: center;  
}
```

```
h2 {  
  text-align: left;  
}
```

```
h3 {  
  text-align: right;  
}
```

```
h1 {  
  text-decoration: overline;  
}
```

```
h2 {
```

```
text-decoration: line-through;
}
h3 {
text-decoration: underline;
}
p.ex {
text-decoration: overline underline;
}
p{
text-transform: uppercase;
text-transform: lowercase;
text-transform: capitalize;}
```

CSS Text Spacing

Property	Description
letter-spacing	Specifies the space between characters in a text
line-height	Specifies the line height
text-indent	Specifies the indentation of the first line in a text-block
white-space	Specifies how to handle white-space inside an element
word-spacing	Specifies the space between words in a text

The `text-indent` property is used to specify the indentation of the first line of a text:

```
p {
text-indent: 50px;
}
```

Letter Spacing

The `letter-spacing` property is used to specify the space between the characters in a text.

```
h1 {  
  letter-spacing: 5px;  
}  
h2 {  
  letter-spacing: -2px;  
}
```

Line Height

The `line-height` property is used to specify the space between lines:

```
p.small {  
  line-height: 0.8;  
}  
  
p.big {  
  line-height: 1.8;  
}
```

Word Spacing

The `word-spacing` property is used to specify the space between the words in a text.

```
p.one {  
  word-spacing: 10px;  
}  
  
p.two {  
  word-spacing: -2px;  
}
```

White Space

The `white-space` property specifies how white-space inside an element is handled.

```
p {  
  white-space: nowrap;  
}
```

Differences Between `<datalist>` and `<select>` tag

`<select>` tag

`<datalist>` tag

The user can choose only one option

The user can choose any option from the given list but can

from the given list.

also use its own input.

This tag is a form input type.

This tag is not a form input type.

The user has to scan a long list so as to select an option.

The user can easily input the option and get the hints and then can be chosen by the user.

The user can be restricted to a list of options.

The user is not restricted by the list of options.

It doesn't provide the auto-complete feature.

It provides the auto-complete feature.

HTML Div Tag

The **HTML <div> tag** is used to group the large section of HTML elements together.

We know that every tag has a specific purpose e.g. p tag is used to specify paragraph, <h1> to <h6> tag are used to specify headings but the <div> tag is just like a container unit which is used to encapsulate other page elements and divides the HTML documents into sections.

The div tag is generally used by web developers to group HTML elements together and apply CSS styles to many elements at once. For example: If you wrap a set of paragraph elements into a div element so you can take the advantage of CSS styles and apply font style to all paragraphs at once instead of coding the same style for each paragraph element.

```
<div style="border:1px solid pink;padding:20px;font-size:20px">
```

```
<p>Welcome to Javatpoint.com, Here you get tutorials on latest technologies.</p>
```

```
<p>This is second paragraph</p>
```

```
</div>
```

Welcome to Javatpoint.com, Here you get tutorials on latest technologies.

This is second paragraph.

Difference between HTML div tag and span tag

div tag

span tag

HTML div is a block element.	HTML span is an inline element
HTML div element is used to wrap large sections of elements .	HTML span element is used to wrap small portion of texts, image etc.

HTML tag

HTML tag is used as a generic container of inline elements. It is used for styling purpose to the grouped inline elements (using class and id attribute or inline style).

The tag does not have any default meaning or rendering.

The tag can be useful for the following task:

- To change the language of a part of the text.
- To change the color, font, background of a part of text using CSS
- To apply the scripts to the particular part of the text.

Note: HTML is much similar as <div> tag, but <div> is used for block-level elements and tag is used for inline elements.

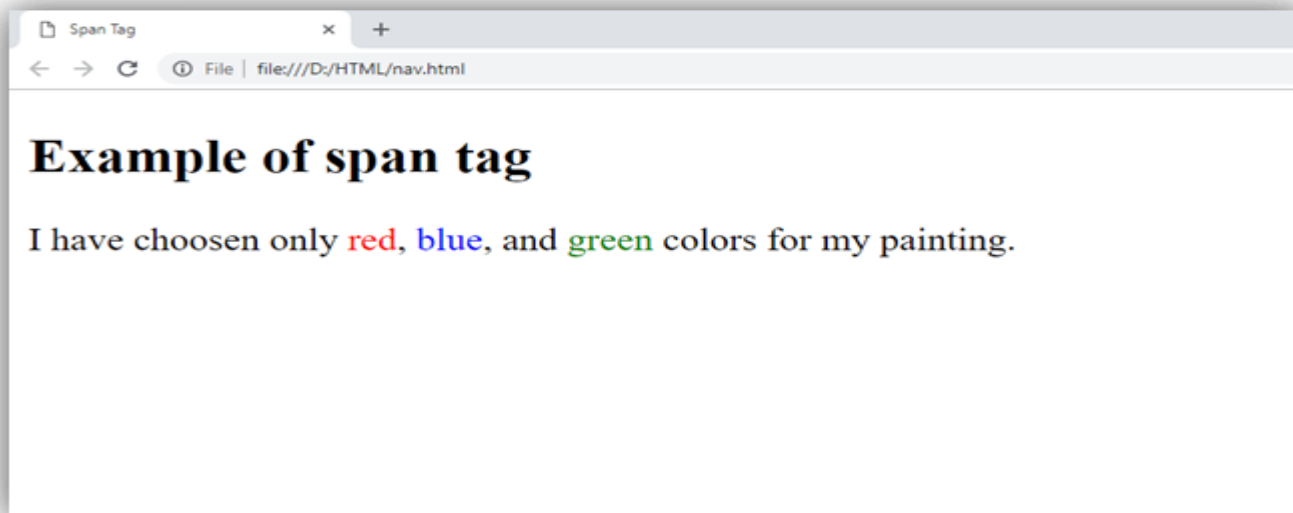
Syntax

****Write your content here.....****

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Span Tag</title>
</head>
<body>
<h2>Example of span tag</h2>
<p>I have choosen only
  <span style="color: red;">red</span>,
  <span style="color: blue;">blue</span>, and
  <span style="color: green;">green</span> colors for my painting.
</p>
</body>
</html>
```

Output:



HTML Section Tag

The HTML <section> tag is used to define sections in a document. When you put your content on a web page, it may contains many chapters, headers, footers, or other sections on a web page that is why HTML <section> tag is used.

HTML <section> is a new tag introduced in HTML5.

HTML section tag example

CSS code:

```
section{
border:1px solid pink;
padding:15px;
margin:10px;
}
```

HTML code:

```
<h2> Indian Leader</h2>
<section>
<h3> Jawaharlal Nehru </h3>
<p> Jawaharlal Nehru was the first Prime Minister of India and a central figure in
Indian politics for much of the 20th century. He emerged as the paramount leader of the Indian
independence movement under the tutelage of Mahatma Gandhi. -Source Wikipedia </p>
</section>
<section>
```

<h3>Subhas Chandra Bose </h3>

<p>Subhas Chandra Bose was an Indian nationalist whose attempt during World War II to rid India of British rule with the help of Nazi Germany and Japan left a troubled legacy.

The honorific Netaji (Hindustani language: "Respected Leader"), first applied to Bose in Germany, by the Indian soldiers of the Indische Legion and by the German and Indian officials in the Special Bureau for India in Berlin, in early 1942, is now used widely throughout India. -

source Wikipedia</p>

</section>

Indian Leader

Jawaharlal Nehru

Jawaharlal Nehru was the first Prime Minister of India and a central figure in Indian politics for much of the 20th century. He emerged as the paramount leader of the Indian independence movement under the tutelage of Mahatma Gandhi. -Source Wikipedia

Subhas Chandra Bose

Subhas Chandra Bose was an Indian nationalist whose attempt during World War II to rid India of British rule with the help of Nazi Germany and Japan left a troubled legacy. The honorific Netaji (Hindustani language: "Respected Leader"), first applied to Bose in Germany, by the Indian soldiers of the Indische Legion and by the German and Indian officials in the Special Bureau for India in Berlin, in early 1942, is now used widely throughout India. -source Wikipedia

HTML Header Tag

HTML <header> tag is used as a container of introductory content or navigation links. Generally a <header> element contains one or more heading elements, logo or icons or author's information.

You can use several <header> elements in one document, but a <header> element cannot be placed within a <footer>, <address> or another <header> element.

HTML Header Tag Example

<header>

<h2>ABCOOnline.com</h2>

<p> World's no.1 shopping website</p>

</header>

ABCOnline.com

World's no.1 shopping website

CSS Code:

```
header{  
border: 1px solid pink;  
background-color: pink;  
padding: 10px;  
border-radius: 5px;  
}
```

HTML Code:

```
<header>  
<h2>ABCOnline.com</h2>  
<p>World's no.1 shopping website</p>  
</header>
```

ABCOnline.com

World's no.1 shopping website

HTML Footer Tag

HTML <footer> tag is used to define a footer for a document or a section. It is generally used in the last of the section (bottom of the page).

The footer tag is included in HTML5.

HTML <footer> tag contains information about its containing elements for example:

- author information
- contact information
- copyright information
- sitemap
- back to top links
- related documents etc.

Note: You can have one or many footer elements in one document.

If you want to put information like address, e-mail etc. about the author on your web page, all the relevant elements should be included into the footer element.

```
<!DOCTYPE>
<html>
<body>
<footer>
<p>Posted by: Sonoo Jaiswal</p>
<p>
<address> JavaTpoint, plot no. 6, near MMX Mall,Mohan Nagar, Ghaziabad Pin no. 201007
</address>
</p>
<p>Contact information:
<a href="mailto:sonoojaiswal1987@gmail.com">sonoojaiswal1987@gmail.com</a>.
</p>
</footer>
</body>
</html>
```

Posted by: Sonoo Jaiswal

JavaTpoint, plot no. 6, near MMX Mall,Mohan Nagar, Ghaziabad Pin no. 201007

Contact information: sonoojaiswal1987@gmail.com.






HTML Audio Tag

HTML audio tag is used to define sounds such as music and other audio clips. Currently there are three supported file format for HTML 5 audio tag.

1. mp3
2. wav
3. ogg

HTML5 supports <video> and <audio> controls. The Flash, Silverlight and similar technologies are used to play the multimedia items.

This table defines that which web browser supports which audio file format.

Browser	mp3	wav	ogg
 Internet Explorer	yes	no	no
 Google Chrome	yes	yes	yes
 Mozilla Firefox	yes*	yes	yes
 Opera	no	yes	yes
 Apple Safari	yes	yes	no

HTML Audio Tag Example

Let's see the code to play mp3 file using HTML audio tag.

```
<!DOCTYPE>
```

```
<html>
```

```
<body>
```

```
<audio controls>
```

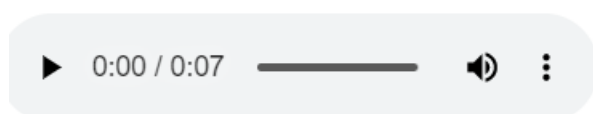
```
  <source src="koyal.mp3" type="audio/mpeg">
```

Your browser does not support the html audio tag.

```
</audio>
```

```
</body>
```

```
</html>
```



Attributes of HTML Audio Tag

There is given a list of HTML audio tag.

Attribute	Description
controls	It defines the audio controls which is displayed with play/pause buttons.
autoplay	It specifies that the audio will start playing as soon as it is ready.

loop	It specifies that the audio file will start over again, every time when it is completed.
muted	It is used to mute the audio output.
preload	It specifies the author view to upload audio file when the page loads.
src	It specifies the source URL of the audio file.

HTML Audio Tag Attribute Example

Here we are going to use controls, autoplay, loop and src attributes of HTML audio tag.

```
<audio controls autoplay loop>
  <source src="koyal.mp3" type="audio/mpeg"></audio>
```

MIME(Multipurpose Internet Mail Extensions) Types for HTML Audio format

The available MIME type HTML audio tag is given below.

Audio Format	MIME Type
mp3	audio/mpeg
Ogg	audio/ogg
Wav	audio/wav






HTML Video Tag

HTML 5 supports <video> tag also. The HTML video tag is used for streaming video files such as a movie clip, song clip on the web page.

Currently, there are three video formats supported for HTML video tag:

1. mp4
2. webM
3. ogg

Let's see the table that defines which web browser supports video file format.

Browser	mp4	webM	ogg
 Internet Explorer	yes	No	no
 Google Chrome	yes	Yes	yes
 Mozilla Firefox	yes	Yes	yes
 Opera	no	Yes	yes
 Apple Safari	yes	No	no

Android also supports mp4 format.

HTML Video Tag Example

Let's see the code to play mp4 file using HTML video tag.

<video controls>

<source src="movie.mp4" type="video/mp4">

Your browser does not support the html video tag.

</video>

Attributes of HTML Video Tag

Let's see the list of HTML 5 video tag attributes.

Attribute	Description
controls	It defines the video controls which is displayed with play/pause buttons.
height	It is used to set the height of the video player.
width	It is used to set the width of the video player.
poster	It specifies the image which is displayed on the screen when the video is not played.
autoplay	It specifies that the video will start playing as soon as it is ready.
loop	It specifies that the video file will start over again, every time when it is completed.

muted	It is used to mute the video output.
preload	It specifies the author view to upload video file when the page loads.
src	It specifies the source URL of the video file.

HTML Video Tag Attribute Example

Let's see the example of video tag in HTML where are using height, width, autoplay, controls and loop attributes.

1. `<video width="320" height="240" controls autoplay loop>`
2. `<source src="movie.mp4" type="video/mp4">`
3. Your browser does not support the html video tag.
4. `</video>`

MIME Types for HTML Video format

The available MIME type HTML video tag is given below.

Video Format	MIME Type
mp4	video/mp4
Ogg	video/ogg
webM	video/webM

HTML Form Attributes

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:

```
<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>
```

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 window
<code>iframe</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.

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

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.

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

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

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.

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

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.

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

List of All <form> Attributes

Attribute	Description
<u>accept-charset</u>	Specifies the character encodings used for form submission
<u>action</u>	Specifies where to send the form-data when a form is submitted
<u>autocomplete</u>	Specifies whether a form should have autocomplete on or off
<u>enctype</u>	Specifies how the form-data should be encoded when submitting it to the server (only for method="post")
<u>method</u>	Specifies the HTTP method to use when sending form-data
<u>name</u>	Specifies the name of the form
<u>novalidate</u>	Specifies that the form should not be validated when submitted
<u>rel</u>	Specifies the relationship between a linked resource and the current document
<u>target</u>	Specifies where to display the response that is received after submitting the form

What is Bootstrap?

- Bootstrap is a free front-end framework for faster and easier web development
- Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins
- Bootstrap also gives you the ability to easily create responsive designs

What is Responsive Web Design?

Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.

Bootstrap Versions

This tutorial follows **Bootstrap 4**, which was released in 2018, as an upgrade to [Bootstrap 3](#), with new components, faster stylesheets, more responsiveness, etc.

[Bootstrap 5](#) (released 2021) is the newest version of [Bootstrap](#); It supports the latest, stable releases of all major browsers and platforms. However, Internet Explorer 11 and down is not supported.

The main differences between Bootstrap 5 and Bootstrap 3 & 4, is that Bootstrap 5 has switched to [JavaScript](#) instead of [jQuery](#).

Note: [Bootstrap 3](#) and Bootstrap 4 is still supported by the team for critical bugfixes and documentation changes, and it is perfectly safe to continue to use them. However, new features will NOT be added to them.

Why Use Bootstrap?

Advantages of Bootstrap:

- **Easy to use:** Anybody with just basic knowledge of HTML and CSS can start using Bootstrap
- **Responsive features:** Bootstrap's responsive CSS adjusts to phones, tablets, and desktops
- **Mobile-first approach:** In Bootstrap, mobile-first styles are part of the core framework
- **Browser compatibility:** Bootstrap 4 is compatible with all modern browsers (Chrome, Firefox, Internet Explorer 10+, Edge, Safari, and Opera)

Where to Get Bootstrap 4?

There are two ways to start using Bootstrap 4 on your own web site.

You can:

- Include Bootstrap 4 from a CDN
- Download Bootstrap 4 from getbootstrap.com

Bootstrap 4 CDN

If you don't want to download and host Bootstrap 4 yourself, you can include it from a CDN (Content Delivery Network).

jsDelivr provides CDN support for Bootstrap's CSS and JavaScript. You must also include jQuery:

jsDelivr:

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">

<!-- jQuery library -->
<script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></script>

<!-- Popper JS -->
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>

<!-- Latest compiled JavaScript -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>
```

Downloading Bootstrap 4

If you want to download and host Bootstrap 4 yourself, go to <https://getbootstrap.com/>, and follow the instructions there.

Create First Web Page With Bootstrap 4

1. Add the HTML5 doctype

Bootstrap 4 uses HTML elements and CSS properties that require the HTML5 doctype.

Always include the HTML5 doctype at the beginning of the page, along with the lang attribute and the correct character set:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
  </head>
</html>
```

2. Bootstrap 4 is mobile-first

Bootstrap 4 is designed to be responsive to mobile devices. Mobile-first styles are part of the core framework.

To ensure proper rendering and touch zooming, add the following `<meta>` tag inside the `<head>` element:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

The `width=device-width` part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The `initial-scale=1` part sets the initial zoom level when the page is first loaded by the browser.

3. Containers

Bootstrap 4 also requires a containing element to wrap site contents.

There are two container classes to choose from:

1. The `.container` class provides a responsive **fixed width container**

2. The `.container-fluid` class provides a **full width container**, spanning the entire width of the viewport

`.container`

`.container-fluid`

Two Basic Bootstrap 4 Pages

The following example shows the code for a basic Bootstrap 4 page (with a responsive fixed width container):

Container Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap 4 Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">
  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>
</head>
<body>

<div class="container">
  <h1>My First Bootstrap Page</h1>
  <p>This is some text.</p>
</div>

</body>
</html>
```

The following example shows the code for a basic Bootstrap 4 page (with a full width container):

Container Fluid Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap 4 Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">
  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>
</head>
<body>

<div class="container-fluid">
  <h1>My First Bootstrap Page</h1>
  <p>This is some text.</p>
</div>

</body>
</html>
```

Bootstrap 4 Containers

Containers

Bootstrap requires a containing element to wrap site contents.

Containers are used to pad the content inside of them, and there are two container classes available:

1. The `.container` class provides a responsive **fixed width container**
2. The `.container-fluid` class provides a **full width container**, spanning the entire width of the viewport

`.container`

`.container-fluid`

Fixed Container

Use the `.container` class to create a responsive, fixed-width container.

Note that its width (`max-width`) will change on different screen sizes:

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
max-width	100%	540px	720px	960px	1140px

Open the example below and resize the browser window to see that the container width will change at different breakpoints:

Example

```
<div class="container">
  <h1>My First Bootstrap Page</h1>
  <p>This is some text.</p>
</div>
```

Fluid Container

Use the `.container-fluid` class to create a full width container, that will always span the entire width of the screen (`width` is always `100%`):

Example

```
<div class="container-fluid">
  <h1>My First Bootstrap Page</h1>
  <p>This is some text.</p>
</div>
```

Container Padding

By default, containers have 15px left and right padding, with no top or bottom padding. Therefore, we often use **spacing utilities**, such as extra padding and margins to make them look even better. For example, `.pt-3` means "add a top padding of 16px":

Example

```
<div class="container pt-3"></div>
```

Container Border and Color

Other utilities, such as borders and colors, are also often used together with containers:

Example

My First Bootstrap Page

This container has a border and some extra padding and margins.

My First Bootstrap Page

This container has a dark background color and a white text, and some extra padding and margins.

My First Bootstrap Page

This container has a blue background color and a white text, and some extra padding and margins.

```
<div class="container p-3 my-3 border"></div>
```

```
<div class="container p-3 my-3 bg-dark text-white"></div>
```

```
<div class="container p-3 my-3 bg-primary text-white"></div>
```

Class	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
<code>.container-sm</code>	100%	540px	720px	960px	1140px
<code>.container-md</code>	100%	100%	720px	960px	1140px

Class	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
<code>.container-lg</code>	100%	100%	100%	960px	1140px
<code>.container-xl</code>	100%	100%	100%	100%	1140px

Responsive Containers

You can also use the `.container-sm|md|lg|xl` classes to create responsive containers.

The `max-width` of the container will change on different screen sizes/viewports:

Example

```
<div class="container-sm">.container-sm</div>
<div class="container-md">.container-md</div>
<div class="container-lg">.container-lg</div>
<div class="container-xl">.container-xl</div>
```

Bootstrap 4 Grids

Bootstrap 4 Grid System

Bootstrap's grid system is built with flexbox and allows up to 12 columns across the page.

If you do not want to use all 12 columns individually, you can group the columns together to create wider columns:

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											

The grid system is responsive, and the columns will re-arrange automatically depending on the screen size.

Make sure that the sum adds up to 12 or fewer (it is not required that you use all 12 available columns).

Grid Classes

The Bootstrap 4 grid system has five classes:

- `.col-` (extra small devices - screen width less than 576px)
- `.col-sm-` (small devices - screen width equal to or greater than 576px)
- `.col-md-` (medium devices - screen width equal to or greater than 768px)
- `.col-lg-` (large devices - screen width equal to or greater than 992px)
- `.col-xl-` (xlarge devices - screen width equal to or greater than 1200px)

The classes above can be combined to create more dynamic and flexible layouts.

Tip: Each class scales up, so if you wish to set the same widths for `sm` and `md`, you only need to specify `sm`.

Basic Structure of a Bootstrap 4 Grid

The following is a basic structure of a Bootstrap 4 grid:

```
<!-- Control the column width, and how they should appear on different devices -->
<div class="row">
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
</div>
<div class="row">
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
</div>

<!-- Or let Bootstrap automatically handle the layout -->
<div class="row">
  <div class="col"></div>
  <div class="col"></div>
  <div class="col"></div>
</div>
```

First example: create a row (`<div class="row">`). Then, add the desired number of columns (tags with appropriate `.col-*-*` classes). The first star (*) represents the responsiveness: `sm`, `md`, `lg` or `xl`, while the second star represents a number, which should add up to 12 for each row.

Second example: instead of adding a number to each `col`, let bootstrap handle the layout, to create equal width columns: two `"col"` elements = 50% width to each col. three cols = 33.33% width to each col. four cols = 25% width, etc. You can also use `.col-sm|md|lg|xl` to make the columns responsive.

Below we have collected some examples of basic Bootstrap 4 grid layouts.

Three Equal Columns

```
.col
.col
.col
```

The following example shows how to create three equal-width columns, on all devices and screen widths:

Example

```
<div class="row">
  <div class="col">.col</div>
  <div class="col">.col</div>
  <div class="col">.col</div>
</div>
```

Responsive Columns

```
.col-sm-3
.col-sm-3
.col-sm-3
.col-sm-3
```

The following example shows how to create four equal-width columns starting at tablets and scaling to extra large desktops. **On mobile phones or screens that are less than 576px wide, the columns will automatically stack on top of each other:**

Example

```
<div class="row">
  <div class="col-sm-3">.col-sm-3</div>
  <div class="col-sm-3">.col-sm-3</div>
  <div class="col-sm-3">.col-sm-3</div>
  <div class="col-sm-3">.col-sm-3</div>
</div>
```

Unequal Responsive Columns

```
.col-sm-4
.col-sm-8
```

The following example shows how to get two various-width columns starting at tablets and scaling to large extra desktops:

Example

```
<div class="row">
  <div class="col-sm-4">.col-sm-4</div>
  <div class="col-sm-8">.col-sm-8</div>
</div>
```

Three equal width columns

Note: Try to add a new div with class="col" inside the row class - this will create four equal-width columns.

```
.col.col.col
```

```
<body>
```

```
<div class="container-fluid">
```

```
  <h1>Three equal width columns</h1>
```

<p>Note: Try to add a new div with class="col" inside the row class - this will create four equal-width columns.</p>

```
<div class="row">

  <div class="col" style="background-color:lavender;">.col</div>

  <div class="col" style="background-color:orange;">.col</div>

  <div class="col" style="background-color:lavender;">.col</div>

</div>

</div>

</body>
```

Responsive Columns

Resize the browser window to see the effect.

The columns will automatically stack on top of each other when the screen is less than 576px wide.

.col-sm-3	.col-sm-3	.col-sm-3	.col-sm-3
-----------	-----------	-----------	-----------

```
<div class="container-fluid">

  <h1>Responsive Columns</h1>

  <p>Resize the browser window to see the effect.</p>

  <p>The columns will automatically stack on top of each other when the screen is less than 576px wide.</p>

  <div class="row">

    <div class="col-sm-3" style="background-color:lavender;">.col-sm-3</div>

    <div class="col-sm-3" style="background-color:lavenderblush;">.col-sm-3</div>

    <div class="col-sm-3" style="background-color:lavender;">.col-sm-3</div>

    <div class="col-sm-3" style="background-color:lavenderblush;">.col-sm-3</div>

  </div>

</div>
```

Two Unequal Responsive Columns

Resize the browser window to see the effect.

The columns will automatically stack on top of each other when the screen is less than 576px wide.

.col-sm-4	.col-sm-8
-----------	-----------

```
<div class="container-fluid">

  <h1>Two Unequal Responsive Columns</h1>

  <p>Resize the browser window to see the effect.</p>

  <p>The columns will automatically stack on top of each other when the screen is less than 576px wide.</p>
```

```
<div class="row">
  <div class="col-sm-4" style="background-color:lavender;">.col-sm-4</div>
  <div class="col-sm-8" style="background-color:lavenderblush;">.col-sm-8</div>
</div>
</div>
```

Bootstrap 4 Alerts

Bootstrap 4 provides an easy way to create predefined alert messages:

- × **Success!** This alert box indicates a successful or positive action.
- × **Info!** This alert box indicates a neutral informative change or action.
- × **Warning!** This alert box indicates a warning that might need attention.
- × **Danger!** This alert box indicates a dangerous or potentially negative action.
- × **Primary!** This alert box indicates an important action.
- × **Secondary!** This alert box indicates a less important action.
- × **Dark!** Dark grey alert box.
- × **Light!** Light grey alert box.

Alerts are created with the `.alert` class, followed by one of the contextual classes `.alert-success`, `.alert-info`, `.alert-warning`, `.alert-danger`, `.alert-primary`, `.alert-secondary`, `.alert-light` or `.alert-dark`:

Example

```
<div class="alert alert-success">
  <strong>Success!</strong> Indicates a successful or positive action.
</div>
```

Alert Links

Add the `alert-link` class to any links inside the alert box to create "matching colored links":

- Success!** You should [read this message](#).
- Info!** You should [read this message](#).
- Warning!** You should [read this message](#).
- Danger!** You should [read this message](#).
- Primary!** You should [read this message](#).
- Secondary!** You should [read this message](#).
- Dark!** You should [read this message](#).
- Light!** You should [read this message](#).

Example

```
<div class="alert alert-success">
  <strong>Success!</strong> You should <a href="#" class="alert-link">read this message</a>.
</div>
```

Closing Alerts

[✕](#)Click on the "x" symbol to the right to close me.

To close the alert message, add a `.alert-dismissible` class to the alert container. Then add `class="close"` and `data-dismiss="alert"` to a link or a button element (when you click on this the alert box will disappear).

Example

```
<div class="alert alert-success alert-dismissible">
  <button type="button" class="close" data-dismiss="alert">&times;</button>
  <strong>Success!</strong> Indicates a successful or positive action.
</div>
```

Tip: `×` (×) is an HTML entity that is the preferred icon for close buttons, rather than the letter "x".

For a list of all HTML Entities, [visit our HTML Entities Reference](#).

Animated Alerts

[✕](#)Click on the "x" symbol to the right to close me. I will "fade" out.

The `.fade` and `.show` classes adds a fading effect when closing the alert message:

Example

```
<div class="alert alert-danger alert-dismissible fade show">
```

Bootstrap 4 Table

Bootstrap 4 Basic Table

A basic Bootstrap 4 table has a light padding and horizontal dividers.

The `.table` class adds basic styling to a table:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Striped Rows

The `.table-striped` class adds zebra-stripes to a table:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Bordered Table

The `.table-bordered` class adds borders on all sides of the table and cells:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Hover Rows

The `.table-hover` class adds a hover effect (grey background color) on table rows:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Black/Dark Table

The `.table-dark` class adds a black background to the table:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Dark Striped Table

Combine `.table-dark` and `.table-striped` to create a dark, striped table:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Hoverable Dark Table

The `.table-hover` class adds a hover effect (grey background color) on table rows:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Borderless Table

The `.table-borderless` class removes borders from the table:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Contextual Classes

Contextual classes can be used to color the whole table (`<table>`), the table rows (`<tr>`) or table cells (`<td>`).

Example

Firstname	Lastname	Email
Default	Defaultson	def@somemail.com
Primary	Joe	joe@example.com
Success	Doe	john@example.com
Danger	Moe	mary@example.com
Info	Dooley	july@example.com
Warning	Refs	bo@example.com
Active	Activeson	act@example.com
Secondary	Secondson	sec@example.com
Light	Angie	angie@example.com
Dark	Bo	bo@example.com

The contextual classes that can be used are:

Class	Description
<code>.table-primary</code>	Blue: Indicates an important action
<code>.table-success</code>	Green: Indicates a successful or positive action
<code>.table-danger</code>	Red: Indicates a dangerous or potentially negative action
<code>.table-info</code>	Light blue: Indicates a neutral informative change or action
<code>.table-warning</code>	Orange: Indicates a warning that might need attention
<code>.table-active</code>	Grey: Applies the hover color to the table row or table cell
<code>.table-secondary</code>	Grey: Indicates a slightly less important action
<code>.table-light</code>	Light grey table or table row background
<code>.table-dark</code>	Dark grey table or table row background

Table Head Colors

The `.thead-dark` class adds a black background to table headers, and the `.thead-light` class adds a grey background to table headers:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Small table

The `.table-sm` class makes the table smaller by cutting cell padding in half:

Example

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Responsive Tables

The `.table-responsive` class adds a scrollbar to the table when needed (when it is too big horizontally):

Example

```
<div class="table-responsive">
  <table class="table">
    ...
  </table>
</div>
```

You can also decide when the table should get a scrollbar, depending on screen width:

Class	Screen width
<code>.table-responsive-sm</code>	< 576px
<code>.table-responsive-md</code>	< 768px
<code>.table-responsive-lg</code>	< 992px
<code>.table-responsive-xl</code>	< 1200px

Example

```
<div class="table-responsive-sm">
  <table class="table">
    ...
  </table>
</div>
```

Bootstrap 4 Colors

Text Colors

Bootstrap 4 has some contextual classes that can be used to provide "meaning through colors".

The classes for text colors are: `.text-muted`, `.text-primary`, `.text-success`, `.text-info`, `.text-warning`, `.text-danger`, `.text-secondary`, `.text-white`, `.text-dark`, `.text-body` (default body color/often black) and `.text-light`:

Example

This text is muted.

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

Secondary text.

Dark grey text.

Body text.

Contextual text classes can also be used on links, which will add a darker hover color:

Example

Muted link. [Primary link](#). [Success link](#). [Info link](#). [Warning link](#). [Danger link](#). [Secondary link](#). [Dark grey link](#). [Body/black link](#). [Light grey link](#).

You can also add 50% opacity for black or white text with the `.text-black-50` or `.text-white-50` classes:

Example

Black text with 50% opacity on white background

White text with 50% opacity on black background

Background Colors

The classes for background colors are: `.bg-primary`, `.bg-success`, `.bg-info`, `.bg-warning`, `.bg-danger`, `.bg-secondary`, `.bg-dark` and `.bg-light`.

Note that background colors do not set the text color, so in some cases you'll want to use them together with a `.text-*` class.

Example

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

Secondary background color.

Dark grey background color.

Light grey background color.

Bootstrap 4 Forms

Bootstrap 4's Default Settings

Form controls automatically receive some global styling with Bootstrap:

All textual `<input>`, `<textarea>`, and `<select>` elements with class `.form-control` have a width of 100%.

Bootstrap 4 Form Layouts

Bootstrap provides two types of form layouts:

- Stacked (full-width) form
- Inline form

Bootstrap 4 Stacked Form

Email:

Password:

☐ Remember me

The following example creates a stacked form with two input fields, one checkbox, and a submit button.

Add a wrapper element with `.form-group`, around each form control, to ensure proper margins:

Example

```
<form action="/action_page.php">
  <div class="form-group">
    <label for="email">Email address:</label>
    <input type="email" class="form-control" placeholder="Enter email" id="email">
  </div>
  <div class="form-group">
    <label for="pwd">Password:</label>
    <input type="password" class="form-control" placeholder="Enter
password" id="pwd">
  </div>
  <div class="form-group form-check">
    <label class="form-check-label">
      <input class="form-check-input" type="checkbox"> Remember me
    </label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Bootstrap Inline Form

Email: Password: ☐ Remember me

In an inline form, all of the elements are inline and left-aligned.

Note: This only applies to forms within viewports that are at least 576px wide. On screens smaller than 576px, it will stack horizontally.

Additional rule for an inline form:

- Add class `.form-inline` to the `<form>` element

The following example creates an inline form with two input fields, one checkbox, and one submit button:

Example

```
<form class="form-inline" action="/action_page.php">
  <label for="email">Email address:</label>
  <input type="email" class="form-control" placeholder="Enter email" id="email">
  <label for="pwd">Password:</label>
  <input type="password" class="form-control" placeholder="Enter password" id="pwd">
  <div class="form-check">
    <label class="form-check-label">
      <input class="form-check-input" type="checkbox"> Remember me
    </label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Inline Form with Utilities

The inline form above feels "compressed", and will look much better with Bootstrap's spacing utilities. The following example adds a right margin (`.mr-sm-2`) to each input on all devices (small and up). And a margin bottom class (`.mb-2`) is used to style the input field when it breaks (goes from horizontal to vertical due to not enough space/width):

Email: Password: ☐ Remember me

Example

```
<form class="form-inline" action="/action_page.php">
  <label for="email" class="mr-sm-2">Email address:</label>
  <input type="email" class="form-control mb-2 mr-sm-2" placeholder="Enter
email" id="email">
  <label for="pwd" class="mr-sm-2">Password:</label>
  <input type="password" class="form-control mb-2 mr-sm-2" placeholder="Enter
```

```
password" id="pwd">
  <div class="form-check mb-2 mr-sm-2">
    <label class="form-check-label">
      <input class="form-check-input" type="checkbox"> Remember me
    </label>
  </div>
  <button type="submit" class="btn btn-primary mb-2">Submit</button>
</form>
```

Form Row/Grid

You can also use columns (`.col`) to control the width and alignment of form inputs without using spacing utilities. Just remember to put them inside a `.row` container.

In the example below, we use two columns that will appear side by side.

Example

```
<form>
  <div class="row">
    <div class="col">
      <input type="text" class="form-control" id="email" placeholder="Enter
email" name="email">
    </div>
    <div class="col">
      <input type="password" class="form-control" placeholder="Enter
password" name="pswd">
    </div>
  </div>
</form>
```

If you want less grid margins (override default column gutters), use `.form-row` instead of `.row`:

Example

```
<form>
  <div class="form-row">
    <div class="col">
      <input type="text" class="form-control" id="email" placeholder="Enter
email" name="email">
    </div>
    <div class="col">
      <input type="password" class="form-control" placeholder="Enter
password" name="pswd">
    </div>
  </div>
```

```
</div>
</form>
```

Form Validation

Username:

Enter username ×

Please fill out this field.

Password:

Enter password ×

Please fill out this field.

☐ I agree on blabla.

Check this checkbox to continue.

You can use different validation classes to provide valuable feedback to users. Add either `.was-validated` or `.needs-validation` to the `<form>` element, depending on whether you want to provide validation feedback before or after submitting the form. The input fields will have a green (valid) or red (invalid) border to indicate what's missing in the form. You can also add a `.valid-feedback` or `.invalid-feedback` message to tell the user explicitly what's missing, or needs to be done before submitting the form

Bootstrap 4 Pagination

Basic Pagination

If you have a web site with lots of pages, you may wish to add some sort of pagination to each page.



To create a basic pagination, add the `.pagination` class to an `` element. Then add the `.page-item` to each `` element and a `.page-link` class to each link inside ``:

Example

```
<ul class="pagination">
  <li class="page-item"><a class="page-link" href="#">Previous</a></li>
  <li class="page-item"><a class="page-link" href="#">1</a></li>
  <li class="page-item"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item"><a class="page-link" href="#">Next</a></li>
</ul>
```

Active State

The `.active` class is used to "highlight" the current page:

Example

```
<ul class="pagination">
  <li class="page-item"><a class="page-link" href="#">Previous</a></li>
  <li class="page-item"><a class="page-link" href="#">1</a></li>
  <li class="page-item active"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item"><a class="page-link" href="#">Next</a></li>
</ul>
```

Disabled State

The `.disabled` class is used for un-clickable links:

Example

```
<ul class="pagination">
  <li class="page-item disabled"><a class="page-link" href="#">Previous</a></li>
  <li class="page-item"><a class="page-link" href="#">1</a></li>
  <li class="page-item"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item"><a class="page-link" href="#">Next</a></li>
</ul>
```

Pagination Sizing

Pagination blocks can also be sized to a larger or a smaller size:

Add class `.pagination-lg` for larger blocks or `.pagination-sm` for smaller blocks:

Example

```
<ul class="pagination pagination-lg">
  <li class="page-item"><a class="page-link" href="#">Previous</a></li>
  <li class="page-item"><a class="page-link" href="#">1</a></li>
  <li class="page-item"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item"><a class="page-link" href="#">Next</a></li>
</ul>

<ul class="pagination pagination-sm">
  <li class="page-item"><a class="page-link" href="#">Previous</a></li>
  <li class="page-item"><a class="page-link" href="#">1</a></li>
  <li class="page-item"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item"><a class="page-link" href="#">Next</a></li>
</ul>
```

Pagination Alignment

Use utility classes to change the alignment of the pagination:

Previous 1 2 3 Next

Previous 1 2 3 Next

Previous 1 2 3 Next

Example

```
<!-- Default (left-aligned) -->
<ul class="pagination" style="margin:20px 0">
  <li class="page-item">...</li>
</ul>

<!-- Center-aligned -->
<ul class="pagination justify-content-center" style="margin:20px 0">
  <li class="page-item">...</li>
</ul>

<!-- Right-aligned -->
<ul class="pagination justify-content-end" style="margin:20px 0">
  <li class="page-item">...</li>
</ul>
```

Breadcrumbs

Another form for pagination, is breadcrumbs:

Photos / Summer 2017 / Italy / Rome

The `.breadcrumb` and `.breadcrumb-item` classes indicate the current page's location within a navigational hierarchy:

Example

```
<ul class="breadcrumb">
  <li class="breadcrumb-item"><a href="#">Photos</a></li>
  <li class="breadcrumb-item"><a href="#">Summer 2017</a></li>
  <li class="breadcrumb-item"><a href="#">Italy</a></li>
  <li class="breadcrumb-item active">Rome</li>
</ul>
```

Bootstrap 4 Buttons

Button Styles

Bootstrap 4 provides different styles of buttons:

Basic Primary Secondary Success Info Warning Danger Dark Light Link

Basic Primary Secondary Success Info Warning Danger Dark Light Link

Example

```
<button type="button" class="btn">Basic</button>
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-dark">Dark</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-link">Link</button>
```

The button classes can be used on `<a>`, `<button>`, or `<input>` elements:

Example

```
<a href="#" class="btn btn-info" role="button">Link Button</a>
<button type="button" class="btn btn-info">Button</button>
<input type="button" class="btn btn-info" value="Input Button">
<input type="submit" class="btn btn-info" value="Submit Button">
```

Why do we put a # in the href attribute of the link?

Since we do not have any page to link it to, and we do not want to get a "404" message, we put # as the link. In real life it should of course been a real URL to the "Search" page.

Button Outline

Bootstrap 4 provides eight outline/bordered buttons:

Primary Secondary Success Info Warning Danger Dark Light

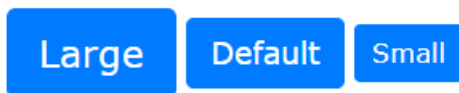


Example

```
<button type="button" class="btn btn-outline-primary">Primary</button>
<button type="button" class="btn btn-outline-secondary">Secondary</button>
<button type="button" class="btn btn-outline-success">Success</button>
<button type="button" class="btn btn-outline-info">Info</button>
<button type="button" class="btn btn-outline-warning">Warning</button>
<button type="button" class="btn btn-outline-danger">Danger</button>
<button type="button" class="btn btn-outline-dark">Dark</button>
<button type="button" class="btn btn-outline-light text-dark">Light</button>
```

Button Sizes

Use the `.btn-lg` class for large buttons or `.btn-sm` class for small buttons:



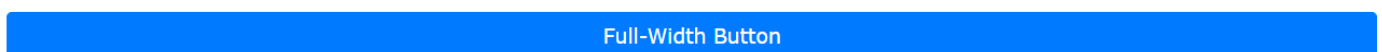
Large Default Small

Example

```
<button type="button" class="btn btn-primary btn-lg">Large</button>
<button type="button" class="btn btn-primary">Default</button>
<button type="button" class="btn btn-primary btn-sm">Small</button>
```

Block Level Buttons

Add class `.btn-block` to create a block level button that spans the entire width of the parent element.



Full-Width Button

Example

```
<button type="button" class="btn btn-primary btn-block">Full-Width Button</button>
```

Active/Disabled Buttons

A button can be set to an active (appear pressed) or a disabled (unclickable) state:

... ..



Active Primary Disabled Primary

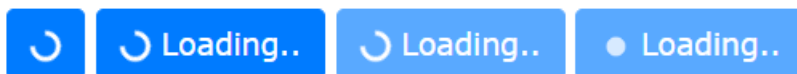
The class `.active` makes a button appear pressed, and the `disabled` attribute makes a button unclickable. Note that `<a>` elements do not support the disabled attribute and must therefore use the `.disabled` class to make it visually appear disabled.

Example

```
<button type="button" class="btn btn-primary active">Active Primary</button>
<button type="button" class="btn btn-primary" disabled>Disabled Primary</button>
<a href="#" class="btn btn-primary disabled">Disabled Link</a>
```

Spinner Buttons

... ..



Example

```
<button class="btn btn-primary">
  <span class="spinner-border spinner-border-sm"></span>
</button>
```

```
<button class="btn btn-primary">
  <span class="spinner-border spinner-border-sm"></span>
  Loading..
</button>
```

```
<button class="btn btn-primary" disabled>
  <span class="spinner-border spinner-border-sm"></span>
  Loading..
</button>
```

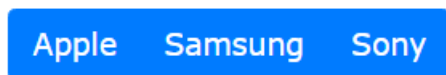
```
<button class="btn btn-primary" disabled>
```

```
<span class="spinner-grow spinner-grow-sm"></span>  
Loading..  
</button>
```

Bootstrap 4 Button Groups

Button Groups

Bootstrap 4 allows you to group a series of buttons together (on a single line) in a button group:



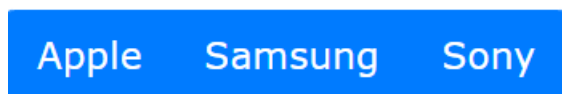
Use a `<div>` element with class `.btn-group` to create a button group:

Example

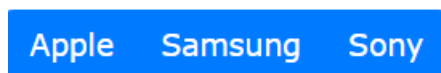
```
<div class="btn-group">  
  <button type="button" class="btn btn-primary">Apple</button>  
  <button type="button" class="btn btn-primary">Samsung</button>  
  <button type="button" class="btn btn-primary">Sony</button>  
</div>
```

Tip: Instead of applying button sizes to every button in a group, use class `.btn-group-lg` for a large button group or the `.btn-group-sm` for a small button group:

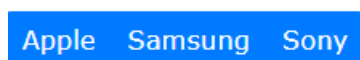
Large Buttons:



Default Buttons:



Small Buttons:



Example

```
<div class="btn-group btn-group-lg">  
  <button type="button" class="btn btn-primary">Apple</button>  
  <button type="button" class="btn btn-primary">Samsung</button>  
  <button type="button" class="btn btn-primary">Sony</button>  
</div>
```

Vertical Button Groups

Bootstrap 4 also supports vertical button groups:



Use the class `.btn-group-vertical` to create a vertical button group:

Example

```
<div class="btn-group-vertical">  
  <button type="button" class="btn btn-primary">Apple</button>  
  <button type="button" class="btn btn-primary">Samsung</button>  
  <button type="button" class="btn btn-primary">Sony</button>  
</div>
```

Bootstrap 4 Images

Bootstrap 4 Image Shapes

Rounded Corners:



Circle:



Thumbnail:



Rounded Corners

The `.rounded` class adds rounded corners to an image:

Example

```

```

Circle

The `.rounded-circle` class shapes the image to a circle:

Example

```

```

Thumbnail

The `.img-thumbnail` class shapes the image to a thumbnail (bordered):

Example

```

```

Aligning Images

Float an image to the right with the `.float-right` class or to the left with `.float-left`:

Example

```
  
  
Aligning images
```

Float an image to the right with the `.float-right` class or to the left with `.float-left`:



Centered Image

Center an image by adding the utility classes `.mx-auto` (margin:auto) and `.d-block` (display:block) to the image:

Center an image by adding the utility classes `.mx-auto` (`margin:auto`) and `.d-block` (`display:block`) to the image:



Example

```

```

Responsive Images

Images come in all sizes. So do screens. Responsive images automatically adjust to fit the size of the screen.

Create responsive images by adding an `.img-fluid` class to the `` tag. The image will then scale nicely to the parent element.

The `.img-fluid` class applies `max-width: 100%;` and `height: auto;` to the image:

Example

```

```

Bootstrap 4 Media Objects

Media Objects

Bootstrap provides an easy way to align media objects (like images or videos) together with content. Media objects are often used to display blog comments, tweets and so on:

Display blog comments, tweets and so on:



John Doe *Posted on February 19, 2016*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



John Doe *Posted on February 20, 2016*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Basic Media Object



John Doe *Posted on February 19, 2016*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

To create a media object, add the `.media` class to a container element, and place media content inside a child container with the `.media-body` class. Add padding and margins as needed, with the spacing utilities:

Example

```
<div class="media border p-3">
  
  <div class="media-body">
    <h4>John Doe <small><i>Posted on February 19, 2016</i></small></h4>
    <p>Lorem ipsum...</p>
  </div>
</div>
```

Nested Media Objects

Media objects can also be nested (a media object inside a media object):



John Doe *Posted on February 19, 2016*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



Jane Doe *Posted on February 20, 2016*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

To nest media objects, place a new `.media` container inside the `.media-body` container:

Example

```
<div class="media border p-3">
  
  <div class="media-body">
    <h4>John Doe <small><i>Posted on February 19, 2016</i></small></h4>
    <p>Lorem ipsum...</p>
    <div class="media p-3">
      <img src="img_avatar2.png" alt="Jane Doe" class="mr-3 mt-3 rounded-
```

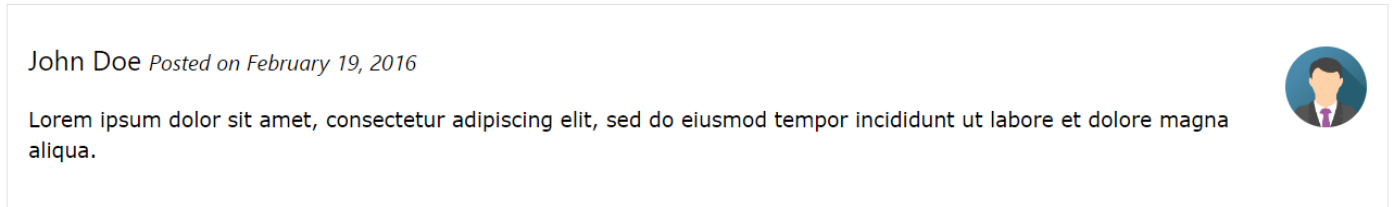


```

circle" style="width:45px;">
  <div class="media-body">
    <h4>Jane Doe <small><i>Posted on February 20 2016</i></small></h4>
    <p>Lorem ipsum...</p>
  </div>
</div>
</div>
</div>

```

Right-Aligned Media Image



To right-align the media image, add the image after the `.media-body` container:

Example

```

<div class="media border p-3">
  <div class="media-body">
    <h4>John Doe <small><i>Posted on February 19, 2016</i></small></h4>
    <p>Lorem ipsum...</p>
  </div>
  
</div>

```

Top, Middle or Bottom Alignment

Use the flex utilities, `align-self-*` classes to place the media object on the top, middle or at the bottom:



Media Top

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



Media Middle

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



Media Bottom

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Example

```
<!-- Media top -->
<div class="media">
  
  <div class="media-body">
    <h4>Media Top</h4>
    <p>Lorem ipsum...</p>
  </div>
</div>

<!-- Media middle -->
<div class="media">
  
  <div class="media-body">
    <h4>Media Middle</h4>
    <p>Lorem ipsum...</p>
  </div>
</div>

<!-- Media bottom -->
<div class="media">
  
  <div class="media-body">
    <h4>Media Bottom</h4>
    <p>Lorem ipsum...</p>
  </div>
</div>
```

Bootstrap 4 Grid System

Bootstrap's grid system allows up to 12 columns across the page.

If you do not want to use all 12 column individually, you can group the columns together to create wider columns:

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											

Bootstrap's grid system is responsive, and the columns will re-arrange depending on the screen size: On a big screen it might look better with the content organized in three columns, but on a small screen it would be better if the content items were stacked on top of each other.

Grid Classes

The Bootstrap 4 grid system has five classes:

- `.col-` (extra small devices - screen width less than 576px)
- `.col-sm-` (small devices - screen width equal to or greater than 576px)
- `.col-md-` (medium devices - screen width equal to or greater than 768px)
- `.col-lg-` (large devices - screen width equal to or greater than 992px)
- `.col-xl-` (xlarge devices - screen width equal to or greater than 1200px)

The classes above can be combined to create more dynamic and flexible layouts.

Tip: Each class scales up, so if you wish to set the same widths for `sm` and `md`, you only need to specify `sm`.

Grid System Rules

Some Bootstrap 4 grid system rules:

- Rows must be placed within a `.container` (fixed-width) or `.container-fluid` (full-width) for proper alignment and padding
- Use rows to create horizontal groups of columns
- Content should be placed within columns, and only columns may be immediate children of rows
- Predefined classes like `.row` and `.col-sm-4` are available for quickly making grid layouts
- Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on `.rows`

- Grid columns are created by specifying the number of 12 available columns you wish to span. For example, three equal columns would use three `.col-sm-4`
- Column widths are in percentage, so they are always fluid and sized relative to their parent element
- The biggest **difference between Bootstrap 3 and Bootstrap 4** is that Bootstrap 4 now uses flexbox, instead of floats. One big advantage with flexbox is that grid columns without a specified width will automatically layout as "equal width columns" (and equal height). Example: Three elements with `.col-sm` will each automatically be 33.33% wide from the small breakpoint and up

Basic Structure of a Bootstrap 4 Grid

The following is a basic structure of a Bootstrap 4 grid:

```
<!-- Control the column width, and how they should appear on different devices -->
<div class="row">
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
</div>

<!-- Or let Bootstrap automatically handle the layout -->
<div class="row">
  <div class="col"></div>
  <div class="col"></div>
  <div class="col"></div>
  <div class="col"></div>
</div>
```

First example: create a row (`<div class="row">`). Then, add the desired number of columns (tags with appropriate `.col-*-*` classes). The first star (*) represents the responsiveness: sm, md, lg or xl, while the second star represents a number, which should always add up to 12 for each row.

Second example: instead of adding a number to each `col`, let bootstrap handle the layout, to create equal width columns: two `"col"` elements = 50% width to each col. three cols = 33.33% width to each col. four cols = 25% width, etc. You can also use `.col-sm|md|lg|xl` to make the columns responsive.

Grid Options

The following table summarizes how the Bootstrap 4 grid system works across different screen sizes:

	Extra small (<576px)	Small (>=576px)	Medium (>=768px)	Large (>=992px)	Extra Large (>=1200px)

Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
Grid behaviour	Horizontal at all times	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints
Container width	None (auto)	540px	720px	960px	1140px
Suitable for	Portrait phones	Landscape phones	Tablets	Laptops	Laptops and Desktops
# of columns	12	12	12	12	12
Gutter width	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)
Nestable	Yes	Yes	Yes	Yes	Yes
Offsets	Yes	Yes	Yes	Yes	Yes
Column ordering	Yes	Yes	Yes	Yes	Yes

Bootstrap 4 Grid Example: Stacked-to-horizontal

We will create a basic grid system that starts out stacked on extra small devices, before becoming horizontal on larger devices.

The following example shows a simple "stacked-to-horizontal" two-column layout, meaning it will result in a 50%/50% split on all screens, except for extra small screens, which it will automatically stack (100%):

Example: Stacked-to-horizontal

```
<div class="container">
  <div class="row">
    <div class="col-sm-6 bg-success">
      <p>Lorem ipsum...</p>
    </div>
    <div class="col-sm-6 bg-warning">
      <p>Sed ut perspiciatis...</p>
    </div>
  </div>
</div>
```

Tip: You can turn any fixed-width layout into a **full-width** layout by changing the `.container` class to `.container-fluid`:

Example: Fluid container

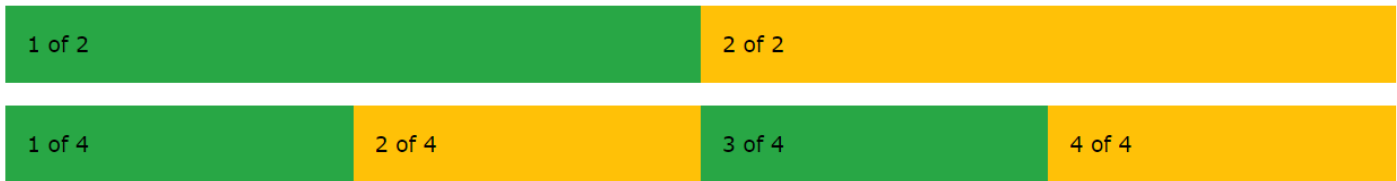
```
<div class="container-fluid">
  <div class="row">
    <div class="col-sm-6">
      <p>Lorem ipsum...</p>
    </div>
    <div class="col-sm-6">
      <p>Sed ut perspiciatis...</p>
    </div>
  </div>
</div>
```

Auto Layout Columns

In Bootstrap 4, there is an easy way to create equal width columns for all devices: just remove the number from `.col-size-*` and only use the `.col-size` class on a specified number of **col elements**. Bootstrap will recognize how many columns there are, and each column will get the same width. The size classes determines when the columns should be responsive:

```
<!-- Two columns: 50% width on all screens, except for extra small (100% width) -->
<div class="row">
  <div class="col-sm">1 of 2</div>
  <div class="col-sm">2 of 2</div>
</div>
```

```
<!-- Four columns: 25% width on all screens, except for extra small (100% width)-->
<div class="row">
  <div class="col-sm">1 of 4</div>
  <div class="col-sm">2 of 4</div>
  <div class="col-sm">3 of 4</div>
  <div class="col-sm">4 of 4</div>
</div>
```



Extra Small Grid Example

	Extra small	Small	Medium	Large	Extra Large
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
Screen width	<576px	>=576px	>=768px	>=992px	>=1200px

Assume we have a simple layout with two columns. We want the columns to split 25%/75% for **ALL** devices.

We will add the following classes to our two columns:

```
<div class="col-3">...</div>
<div class="col-9">...</div>
```

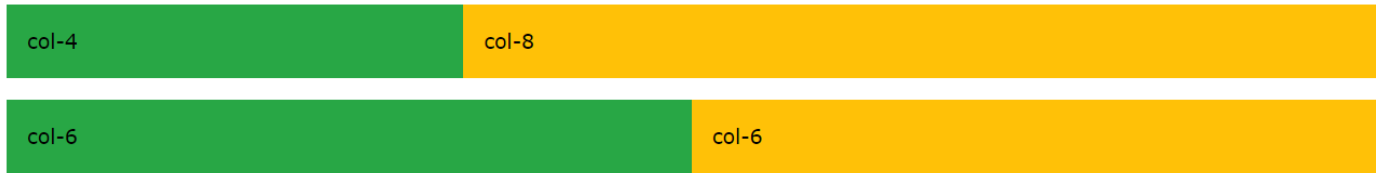
The following example will result in a 25%/75% split on all devices (extra small, small, medium, large and xlarge).



Example

```
<div class="container-fluid">
  <div class="row">
    <div class="col-3 bg-success">
      <p>Lorem ipsum...</p>
    </div>
    <div class="col-9 bg-warning">
      <p>Sed ut perspiciatis...</p>
    </div>
  </div>
</div>
```

For a 33.3%/66.6% split, you would use .col-4 and .col-8 (and for a 50%/50% split, you would use .col-6 and .col-6):



```
<!-- 33.3%/66.6% split -->
<div class="container-fluid">
  <div class="row">
    <div class="col-4 bg-success">
      <p>Lorem ipsum...</p>
    </div>
    <div class="col-8 bg-warning">
      <p>Sed ut perspiciatis...</p>
    </div>
  </div>
</div>
```

```

    </div>
  </div>
</div>

<!-- 50%/50% split -->
<div class="container-fluid">
  <div class="row">
    <div class="col-6 bg-success">
      <p>Lorem ipsum...</p>
    </div>
    <div class="col-6 bg-warning">
      <p>Sed ut perspiciatis...</p>
    </div>
  </div>
</div>

```

Auto Layout Columns

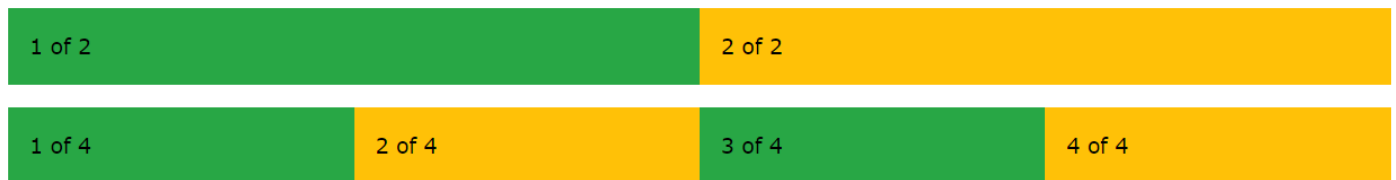
In Bootstrap 4, there is an easy way to create equal width columns for all devices: just remove the number from `.col-*` and only use the `.col` class on a specified number of **col elements**. Bootstrap will recognize how many columns there are, and each column will get the same width:

```

<!-- Two columns: 50% width-->
<div class="row">
  <div class="col">1 of 2</div>
  <div class="col">2 of 2</div>
</div>

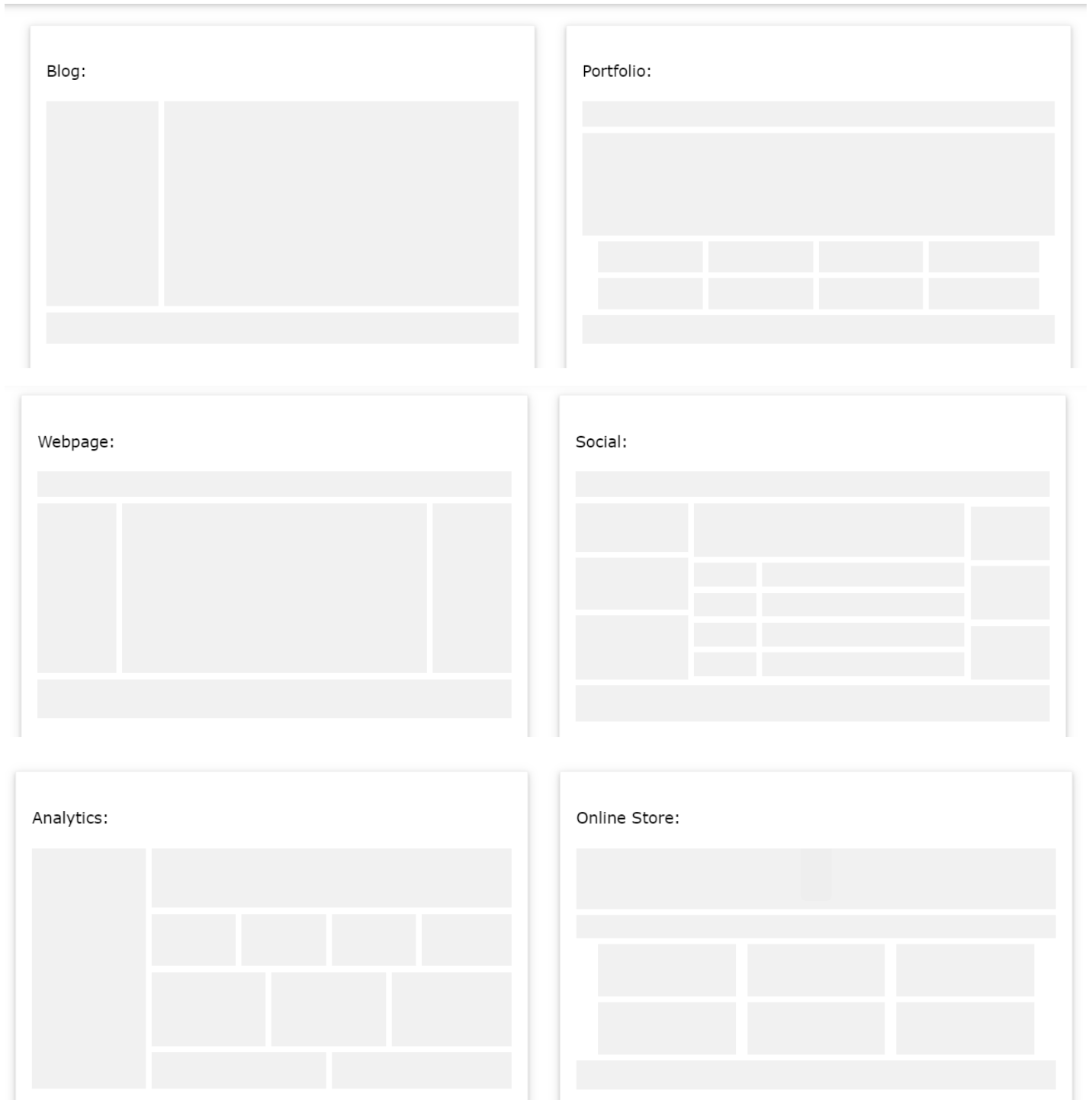
<!-- Four columns: 25% width-->
<div class="row">
  <div class="col">1 of 4</div>
  <div class="col">2 of 4</div>
  <div class="col">3 of 4</div>
  <div class="col">4 of 4</div>
</div>

```



NOTE: you can check all this examplees with all the other grid type options like `md,lg,xl`

Bootstrap Templates



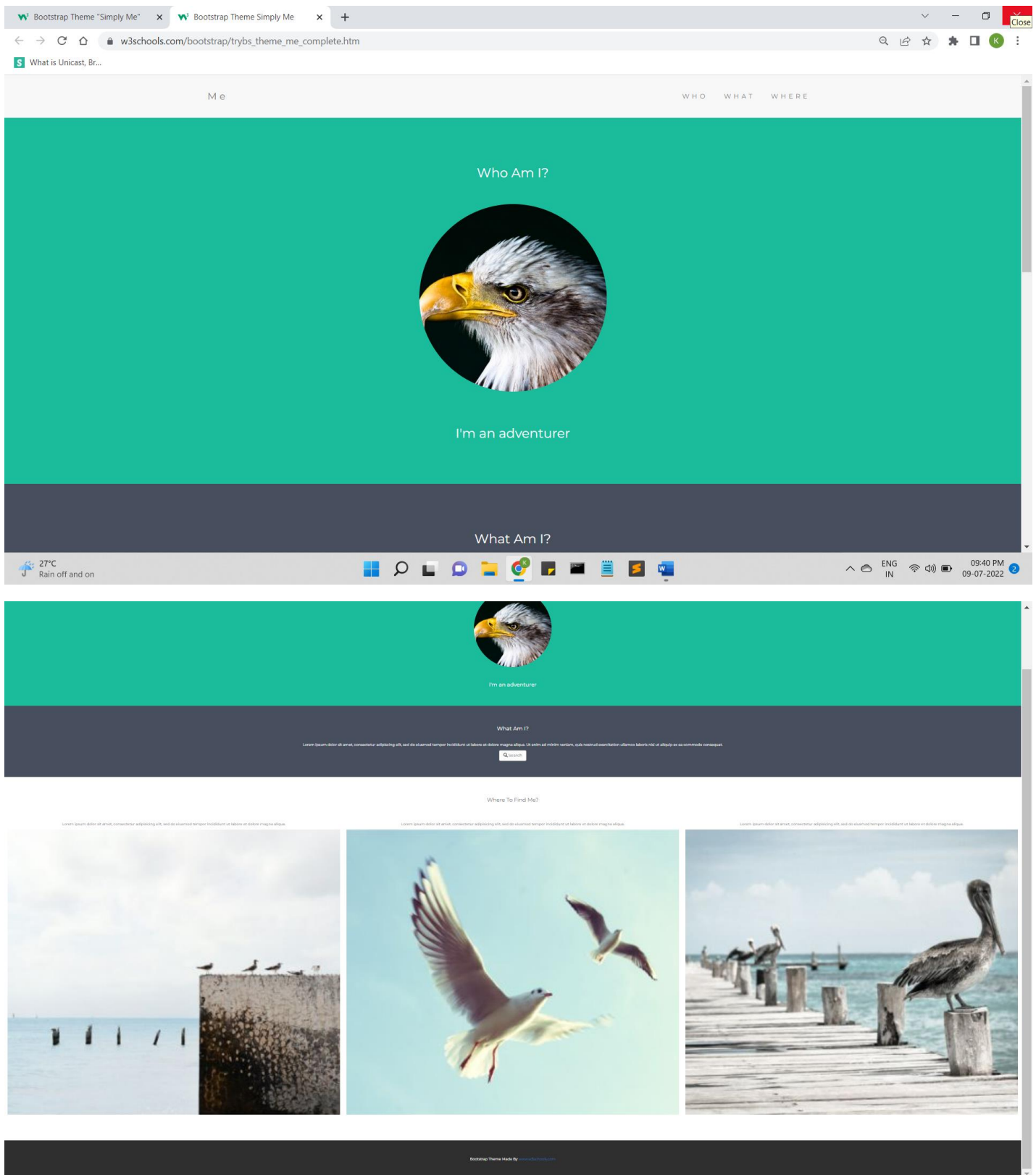
Bootstrap Theme "Simply Me"

Create a Theme: "Simply Me"

This page will show you how to build a Bootstrap theme from scratch.

We will start with a simple HTML page, and then add more and more components, until we have a fully functional, personal and responsive website.

The result will look like this, and you are free to modify, save, share, use or do whatever you want with it:



```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<!-- Theme Made By www.w3schools.com - No Copyright -->
```

```
<title>Bootstrap Theme Simply Me</title>
```

```
<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

<link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet">

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>

<style>
body {
    font: 20px Montserrat, sans-serif;
    line-height: 1.8;
    color: #f5f6f7;
}
p {font-size: 16px;}
.margin {margin-bottom: 45px;}
.bg-1 {
    background-color: #1abc9c; /* Green */
    color: #ffffff;
}
.bg-2 {
    background-color: #474e5d; /* Dark Blue */
    color: #ffffff;
}
.bg-3 {
    background-color: #ffffff; /* White */
    color: #555555;
}
.bg-4 {
    background-color: #2f2f2f; /* Black Gray */
    color: #fff;
}
.container-fluid {
    padding-top: 70px;
    padding-bottom: 70px;
```

```

}
.navbar {
  padding-top: 15px;
  padding-bottom: 15px;
  border: 0;
  border-radius: 0;
  margin-bottom: 0;
  font-size: 12px;
  letter-spacing: 5px;
}
.navbar-nav li a:hover {
  color: #1abc9c !important;
}
</style>
</head>
<body>

<!-- Navbar -->
<nav class="navbar navbar-default">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target="#myNavbar">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">Me</a>
    </div>
    <div class="collapse navbar-collapse" id="myNavbar">
      <ul class="nav navbar-nav navbar-right">
        <li><a href="#">WHO</a></li>
        <li><a href="#">WHAT</a></li>
        <li><a href="#">WHERE</a></li>
      </ul>
    </div>
  </div>

```

```

    </div>
</nav>

<!-- First Container -->
<div class="container-fluid bg-1 text-center">
    <h3 class="margin">Who Am I?</h3>

    <h3>I'm an adventurer</h3>
</div>

<!-- Second Container -->
<div class="container-fluid bg-2 text-center">
    <h3 class="margin">What Am I?</h3>

    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. </p>

    <a href="#" class="btn btn-default btn-lg">
        <span class="glyphicon glyphicon-search"></span> Search
    </a>
</div>

<!-- Third Container (Grid) -->
<div class="container-fluid bg-3 text-center">
    <h3 class="margin">Where To Find Me?</h3><br>
    <div class="row">
        <div class="col-sm-4">
            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.</p>

            
        </div>
        <div class="col-sm-4">
            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.</p>

            
        </div>
    </div>
</div>

```

```
<div class="col-sm-4">

    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.</p>

</div>

</div>

</div>

<!-- Footer -->

<footer class="container-fluid bg-4 text-center">

    <p>Bootstrap Theme Made By <a
href="https://www.w3schools.com">www.w3schools.com</a></p>

</footer>

</body>

</html>
```

What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

History of JavaScript

In 1993, **Mosaic**, the first popular web browser, came into existence. In the **year 1994**, **Netscape** was founded by **Marc Andreessen**. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTML to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited **Brendan Eich** intending to implement and embed Scheme programming language to the browser. But, before Brendan could start, the company merged with **Sun Microsystems** for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'Javascript'. Finally, in May 1995, Marc Andreessen coined the first code of Javascript named '**Mocha**'. Later, the marketing team replaced the name with '**LiveScript**'. But, due to

trademark reasons and certain other reasons, in December 1995, the language was finally renamed to 'JavaScript'. From then, JavaScript came into existence.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

JavaScript Example

```
<script>
document.write("Hello JavaScript by JavaScript");
</script>
```

External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external [JavaScript](#) file that prints Hello Javatpoint in a alert dialog box.

message.js

```
function msg(){
    alert("Hello Javatpoint");
}
```

Let's include the JavaScript file into [html](#) page. It calls the [JavaScript function](#) on button click.

index.html

```
<html>
<head>
<script type="text/javascript" src="message.js"></script>
</head>
<body>
<p>Welcome to JavaScript</p>
```



```
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

1. It helps in the reusability of code in more than one HTML file.
2. It allows easy code readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
4. It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

1. The stealer may download the coder's code using the url of the js file.
2. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
3. The web browser needs to make an additional http request to get the js code.
4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
5. We need to check each file that depends on the commonly created external javascript file.
6. If it is a few lines of code, then better to implement the internal javascript code.

JavaScript Comment

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment
2. Multi-line Comment

JavaScript Single line Comment

It is represented by double forward slashes (//). It can be used before and after the statement.

Let's see the example of single-line comment i.e. added before the statement.

```
<script>
// It is single line comment
document.write("hello javascript");
</script>
```

Let's see the example of single-line comment i.e. added after the statement.

```
<script>
var a=10;
var b=20;
var c=a+b;//It adds values of a and b variable
document.write(c);//prints sum of 10 and 20
</script>
```

JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.

It is represented by forward slash with asterisk then asterisk with forward slash. For example:

```
/* your code here */
```

It can be used before, after and middle of the statement.

```
<script>
/* It is multi line comment.
It will not be displayed */
```

```
document.write("example of javascript multiline comment");
```

```
</script>
```

JavaScript Variable

JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar(\$) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

```
var x = 10;  
var _value="sonoo";
```

Incorrect JavaScript variables

```
var 123=30;  
var *aa=320;
```

Example of JavaScript variable

Let's see a simple example of JavaScript variable.

```
<script>  
var x = 10;  
var y = 20;  
var z=x+y;  
document.write(z);  
</script>
```

Output of the above example

```
30
```

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>  
function abc(){  
var x=10;//local variable  
}
```

```
</script>
```

Or,

```
<script>
```

```
If(10<13){  
var y=20;//JavaScript local variable  
}
```

```
</script>
```

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<script>
```

```
var data=200;//global variable  
function a(){  
document.writeln(data);  
}  
function b(){  
document.writeln(data);  
}  
a();//calling JavaScript function  
b();
```

```
</script>
```

JavaScript Global Variable

A **JavaScript global variable** is declared outside the function or declared with window object. It can be accessed from any function.

Let's see the simple example of global variable in JavaScript.

```
<script>
```

```
var value=50;//global variable  
function a(){  
alert(value);  
}  
function b(){  
alert(value);  
}
```

```
</script>
```

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**. For example:

```
window.value=90;
```

Now it can be declared inside any function and can be accessed from any function. For example:

```
function m(){  
  window.value=100;//declaring global variable by window object  
}  
function n(){  
  alert(window.value);//accessing global variable from other function  
}
```

Internals of global variable in JavaScript

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also. For example:

```
var value=50;  
function a(){  
  alert(window.value);//accessing global variable  
}
```

Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

1. var a=40;//holding number
2. var b="Rahul";//holding string

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

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

JavaScript non-primitive data types

The non-primitive data types are as follows:

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

We will have great discussion on each data type later.

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

```
var sum=10+20;
```

Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Logical Operators
4. Assignment Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

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

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

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

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x = 5**, the table below explains the comparison operators:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	True
>	greater than	x > 8	False
<	less than	x < 8	True

>=	greater than or equal to	x >= 8	False
<=	less than or equal to	x <= 8	True

NOTE:The main difference between the == and === operator in javascript is that the == operator does the type conversion of the operands before comparison, whereas the === operator compares the values as well as the data types of the operands.

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
	Logical OR	(10==20 20==33) = false
!	Logical Not	!(10==20) = true
OR Example	let x = 6; let y = 3; (x == 5 y == 5) (x == 6 y == 0) (x == 0 y == 3) (x == 6 y == 3)	false true true true
And Example	let x = 6; let y = 3; (x < 10 && y > 1) (x < 10 && y < 1)	true false
Not Example	let x = 6; let y = 3; !(x === y) !(x > y)	true false

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

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

JavaScript New Operators

new	creates an instance (object)
-----	------------------------------

JavaScript If-else

The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.

1. If Statement
2. If else statement
3. if else if statement

JavaScript If statement

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

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

Example:

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

```
}  
</script>
```

Output of the above example

```
value of a is greater than 10
```

JavaScript If...else Statement

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

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

Let's see the example of if-else statement in JavaScript to find out the even or odd number.

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

Output of the above example

```
a is even number
```

JavaScript If...else if statement

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

```
if(expression1){  
  //content to be evaluated if expression1 is true  
}  
else if(expression2){  
  //content to be evaluated if expression2 is true  
}  
else if(expression3){  
  //content to be evaluated if expression3 is true  
}
```

```
else{
  //content to be evaluated if no expression is true
}
```

Let's see the simple example of if else if statement in javascript.

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

Output of the above example

a is equal to 20

JavaScript Switch

The **JavaScript switch statement** is used to *execute one code from multiple expressions*. It is just like else if statement that we have learned in previous page. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

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

default:

code to be executed if above values are not matched;

}

Let's see the simple example of switch statement in javascript.

```
<script>
```

```
var grade='B';
```

```
var result;
```

```
switch(grade){
```

```
case 'A':
```

```
result="A Grade";
```

```
break;
```

```
case 'B':
```

```
result="B Grade";
```

```
break;
```

```
case 'C':
```

```
result="C Grade";
```

```
break;
```

```
default:
```

```
result="No Grade";
```

```
}
```

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

```
</script>
```

Output of the above example

B Grade

The switch statement is fall-through i.e. all the cases will be evaluated if you don't use break statement.

Let's understand the behaviour of switch statement in JavaScript.

```
<script>
```

```
var grade='B';
```

```
var result;
```

```
switch(grade){
```

```
case 'A':
```

```
result+=" A Grade";
```

```
case 'B':
```

```
result+=" B Grade";
```

```
case 'C':
```

```
result+=" C Grade";
```

```
default:
```

```
result+=" No Grade";
}
document.write(result);
</script>
```

Output of the above example

```
undefined B Grade C Grade No Grade
```

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop

1) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

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

Let's see the simple example of for loop in javascript.

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

Output:

```
1
2
3
4
5
```

2) JavaScript while loop

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

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

Let's see the simple example of while loop in javascript.

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

Output:

```
11
12
13
14
15
```

3) JavaScript do while loop

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

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

Let's see the simple example of do while loop in javascript.

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

```
}while (i<=25);  
</script>
```

Output:

```
21  
22  
23  
24  
25
```

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

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

1) JavaScript array literal

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

```
var arrayname=[value1,value2.....valueN];
```

As you can see, values are contained inside [] and separated by , (comma).

Let's see the simple example of creating and using array in JavaScript.

```
<script>  
var emp=["Sonoo","Vimal","Ratan"];  
for (i=0;i<emp.length;i++){  
  document.write(emp[i] + "<br/>");  
}  
</script>
```

The .length property returns the length of an array.

Output of the above example

```
Sonoo  
Vimal  
Ratan
```

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

```
var arrayname=new Array();
```

Here, **new keyword** is used to create instance of array.

Let's see the example of creating array directly.

```
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

Output of the above example

```
Arun
Varun
John
```

3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

```
<script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

Output of the above example

```
Jai
Vijay
Smith
```

JavaScript String

The **JavaScript string** is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

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

1) By string literal

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

```
var stringname="string value";  
var name="khyati"
```

Let's see the simple example of creating string literal.

```
<script>  
var str="This is string literal";  
document.write(str);  
</script>
```

Output:

```
This is string literal
```

2) By string object (using new keyword)

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

```
var stringname=new String("string literal");  
var name=new String("Khyati");
```

Here, **new keyword** is used to create instance of string.

Let's see the example of creating string in JavaScript by new keyword.

```
<script>  
var stringname=new String("hello javascript string");  
document.write(stringname);  
</script>
```

Output:

```
hello javascript string
```

JavaScript String Methods

Let's see the list of JavaScript string methods with examples.

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

<u>valueOf()</u>	It provides the primitive value of string object.
------------------	---

JavaScript String charAt(index) Method

The JavaScript String charAt() method returns the character at the given index.

```
<script>
var str="javascript";
document.write(str.charAt(2));
</script>
```

Output:

```
v
```

2) JavaScript String concat(str) Method

The JavaScript String concat(str) method concatenates or joins two strings.

```
<script>
var s1="javascript ";
var s2="concat example";
var s3=s1.concat(s2);
document.write(s3);
</script>
```

Output:

```
javascript concat example
```

3) JavaScript String indexOf(str) Method

The JavaScript String indexOf(str) method returns the index position of the given string.

```
<script>
var s1="javascript from javatpoint indexof";
var n=s1.indexOf("from");
document.write(n);
</script>
```

Output:

```
11
```

4) JavaScript String lastIndexOf(str) Method

The JavaScript String lastIndexOf(str) method returns the last index position of the given string.

```
<script>
var s1="javascript from javatpoint indexof";
var n=s1.lastIndexOf("java");
document.write(n);
</script>
```

Output:

```
16
```

5) JavaScript String toLowerCase() Method

The JavaScript String toLowerCase() method returns the given string in lowercase letters.

```
<script>
var s1="JavaScript toLowerCase Example";
var s2=s1.toLowerCase();
document.write(s2);
</script>
```

Output:

```
javascript tolowercase example
```

6) JavaScript String toUpperCase() Method

The JavaScript String toUpperCase() method returns the given string in uppercase letters.

```
<script>
var s1="JavaScript toUpperCase Example";
var s2=s1.toUpperCase();
document.write(s2);
</script>
```

Output:

```
JAVASCRIPT TOUPPERCASE EXAMPLE
```

7) JavaScript String slice(beginIndex, endIndex) Method

The JavaScript String slice(beginIndex, endIndex) method returns the parts of string from given beginIndex to endIndex. In slice() method, beginIndex is inclusive and endIndex is exclusive.

```
<script>
var s1="abcdefgh";
var s2=s1.slice(2,5);
document.write(s2);
```

```
</script>
```

Output:

```
cde
```

8) JavaScript String trim() Method

The JavaScript String trim() method removes leading and trailing whitespaces from the string.

```
<script>
```

```
var s1=" javascript trim ";
```

```
var s2=s1.trim();
```

```
document.write(s2);
```

```
</script>
```

Output:

```
javascript trim
```

9) JavaScript String split() Method

```
<script>
```

```
var str="This is JavaTpoint website";
```

```
document.write(str.split(" ")); //splits the given string.
```

```
</script>
```

10) JavaScript String substring() Method

```
var text = "Hello world!";
```

```
var result = text.substring(1, 4);
```

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

output: ell

11) JavaScript String substr() Method

```
var text = "Hello world!";
```

```
var result = text.substr(1, 4);
```

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

output: ello

12) JavaScript String replace() Method

```
var text = "Visit Microsoft!";
```

```
var result = text.replace("Microsoft", "W3Schools");  
document.write(result);
```

output: Visit W3Schools

13) JavaScript String search() Method

```
var text = "Mr. Blue has a blue house";  
  
var position = text.search("Blue");  
  
document.write(position);
```

output: 4

JavaScript code to show the working of string.length property:

```
<script>  
  
  // Taking some strings  
  var x = 'geeksforgeeks';  
  var y = 'gfg';  
  var z = '';  
  
  // Returning the length of the string.  
  document.write(x.length + "<br>");  
  document.write(y.length + "<br>");  
  document.write(z.length);  
  
</script>
```

Output:

13

3

0

substr() Vs. substring()

The JavaScript string is an object that represents a sequence of characters. The **substr()** method extracts parts of a string, beginning at the character at the specified position, and returns the specified number of characters. The **substring()** method returns the part of the string between the start and end indexes, or to the end of the string.

string.substr(start, length)

start: The position where to start the extraction, index starting from 0.

length: The number of characters to extract (optional).

```
var s = "JavaScript";  
var st = s.substr(4, 6);  
alert(st)
```

The above code would return "Sc".

string.substring(start, end)

start: The position where to start the extraction, index starting from 0. **end:** The position (up to, but not including) where to end the extraction (optional).

```
var s = "JavaScript";  
var st = s.substr(4, 6);  
alert(st);
```

The above code would return "Scr"

substr() Vs. substring()

The difference is in the second argument. The **second argument** to substring is the index to stop at (but not include), but the second argument to substr is the maximum **length** to return. Moreover, substr() accepts a negative starting position as an offset from the end of the string. substring() does not.

JavaScript's string `substring()` and `slice()`

JavaScript's string `substring()` and `slice()` functions both let you extract substrings from a string. But they have a couple of key differences that you need to be aware of.

Negative Values

With `slice()`, when you enter a negative number as an argument, the `slice()` interprets it as counting from the end of the string. With `substring()`, it will treat a negative value as zero.

```
const sentence = 'Mastering JS is a very helpful website';  
sentence.slice(-7); // 'website'  
sentence.substring(-5, 12); // 'Mastering JS'  
  
sentence.slice(0, -26); // 'Mastering JS'
```

Parameter Consistency

A big difference with `substring()` is that if the 1st argument is greater than the 2nd argument, `substring()` will swap them. `slice()` returns an empty string if the 1st argument is greater than the 2nd argument.

```
const sentence = 'Mastering JS is a very helpful website';
sentence.substring(12, 0); // 'Mastering JS'
sentence.slice(12, 0); // ''
sentence.slice(0, 12); // 'Mastering JS'
```

JavaScript Events

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

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

Some of the HTML events and their event handlers are:

Mouse events:

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

Keyboard events:

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

Form events:

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

Window/Document events

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

Let's discuss some examples over events and their handlers.

Click Event

```

<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
  <!--

```

```

function clickevent()
{
    document.write("This is JavaTpoint");
}
//-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>

```

MouseOver Event

```

<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
    function mouseoverevent()
    {
        alert("This is JavaTpoint");
    }
    //-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>

```

Focus Event

```

html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
function focusevent()
{
    document.getElementById("input1").style.background=" aqua";
}

```

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

Keydown Event

```
<html>  
<head> Javascript Events</head>  
<body>  
<h2> Enter something here</h2>  
<input type="text" id="input1" onkeydown="keydownevent()" />  
<script>  
<!--  
    function keydownevent()  
    {  
        document.getElementById("input1");  
        alert("Pressed a key");  
    }  
//-->  
</script>  
</body>  
</html>
```

Load event

```
<html>  
<head> Javascript Events</head>  
<br>  
<body onload="window.alert('Page successfully loaded');">  
<script>  
<!--  
document.write("The page is loaded successfully");  
//-->  
</script>  
</body>  
</html>
```

Difference between var, let and const keywords in JavaScript

In [JavaScript](#), users can declare a variable using 3 keywords that are [var, let, and const](#).so, we will see the differences between the var, let, and const keywords. We will discuss the scope and other required concepts about each keyword.

var keyword in JavaScript: The *var* is the oldest keyword to declare a variable in JavaScript.

Scope: Global scoped or function scoped. The scope of the *var* keyword is the global or function scope. It means variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function.

Example 1: Variable 'a' is declared globally. So, the scope of the variable 'a' is global, and it can be accessible everywhere in the program. The output shown is in the console.

- Javascript

```
<script>
  var a = 10
  function f(){
    console.log(a)
  }
  f();
  console.log(a);
</script>
```

Output:

10

10

Example 2: The variable 'a' is declared inside the function. If the user tries to access it outside the function, it will display the error. Users can declare the 2 variables with the same name using the *var* keyword. Also, the user can reassign the value into the *var* variable. The output shown in the console.

- Javascript

```
<script>
  function f() {

    // It can be accessible any
    // where within this function
    var a = 10;
    console.log(a)
  }
  f();
  // A cannot be accessible
  // outside of function
  console.log(a);
</script>
```

Output:

10

ReferenceError: a is not defined

Example 3: User can re-declare variable using *var* and user can update *var* variable. The output is shown in the console.

- Javascript

```
<script>
  var a = 10

  // User can re-declare
  // variable using var
  var a = 8

  // User can update var variable
  a = 7
document.write(a);
function abc()
{
  b=10;
  b=15;

}
</script>
```

Output:

7

Example 4: If users use the *var* variable before the declaration, it initializes with the *undefined* value. The output is shown in the console.

- Javascript

```
<script>
  console.log(a);
  var a = 10;
</script>
```

Output:

undefined

let keyword in JavaScript: The *let* keyword is an improved version of the *var* keyword.

Scope: block scoped: The scope of a *let* variable is only block scoped. It can't be accessible outside the particular block ({block}). Let's see the below example.

Example 1: The output is shown in the console.

- Javascript

```
<script>
  let a = 10;
  function f() {
    let b = 9
    console.log(b);
    console.log(a);
  }
  f();
</script>
```

Output:

9
10

Example 2: The code returns an error because we are accessing the *let* variable outside the function block. The output is shown in the console.

- Javascript

```
<script>
  let a = 10;
  function f() {
    if (true) {
      let b = 9

      // It prints 9
      console.log(b);
    }

    // It gives error as it
    // defined in if block

  }    console.log(b);
  f()

  // It prints 10
  console.log(a)
</script>
```

Output:

9
ReferenceError: b is not defined

Example 3: Users cannot re-declare the variable defined with the *let* keyword but can update it.

- Javascript

```
<script>

  let a = 10

  // It is not allowed
```

```
let a = 10
```

```
// It is allowed
```

```
a = 10
```

```
</script>
```

Output:

Uncaught SyntaxError: Identifier 'a' has already been declared

Example 4: Users can declare the variable with the same name in different blocks using the *let* keyword.

- Javascript

```
<script>
```

```
let a = 10
```

```
if (true) {
```

```
let a=9
```

```
console.log(a) // It prints 9
```

```
}
```

```
console.log(a) // It prints 10
```

```
</script>
```

Output:

9

10

Example 5: If users use the *let* variable before the declaration, it does not initialize with *undefined* just like a *var* variable and return an error.

- Javascript

```
<script>
```

```
console.log(a);
```

```
let a = 10;
```

```
</script>
```

Output:

Uncaught ReferenceError: Cannot access 'a' before initialization

const keyword in JavaScript: The *const* keyword has all the properties that are the same as the *let* keyword, except the user cannot update it.

Scope: block scoped: When users declare a *const* variable, they need to initialize it, otherwise, it returns an error. The user cannot update the *const* variable once it is declared.

Example 1: We are changing the value of the *const* variable so that it returns an error. The output is shown in the console.

- Javascript

```
<script>
```



```
const a = 10;
function f() {
  a = 9
  console.log(a)
}
f();
</script>
```

Output:

TypeError: Assignment to constant variable.

Differences between var, let, and const

Var	let	const
The scope of a <i>var</i> variable is functional scope.	The scope of a <i>let</i> variable is block scope.	The scope of a <i>const</i> variable is block scope.
It can be updated and re-declared into the scope.	It can be updated but cannot be re-declared into the scope.	It cannot be updated or re-declared into the scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
It can be accessed without initialization as its default value is “undefined”.	It cannot be accessed without initialization, as it returns an error.	It cannot be accessed without initialization, as it cannot be declared without initialization.


Note: Sometimes, users face the problem while working with the *var* variable as they change the value of it in the particular block. So, users should use the *let* and *const* keyword to declare a variable in JavaScript.

JavaScript Objects

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

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

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

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:

```
let car = "Fiat";
```

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

```
const car = {type:"Fiat", model:"500", color:"white"};
```

The values are written as **name:value** pairs (name and value separated by a colon).

It is a common practice to declare objects with the const keyword.

Object Definition

You define (and create) a JavaScript object with an **object literal**:

Example

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Spaces and line breaks are not important. An object definition can span multiple lines:

Example

```
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};
```

Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

Property	Property Value
firstName	John
lastName	Doe
Age	50
eyeColor	Blue

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

or

objectName["propertyName"]

Example1

```
person.lastName;
```

Example2

```
person["lastName"];
```

JavaScript objects are containers for **named values** called properties.

Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	Blue
fullName	function() {return this.firstName + " " + this.lastName;}

A method is a function stored as a property.

Example

```
const person = {  
  firstName: "John",
```

```
lastName : "Doe",  
id       : 5566,  
fullName : function() {  
    return this.firstName + " " + this.lastName;  
}  
};
```

In the example above, this refers to the **person object**.

I.E. **this.firstName** means the **firstName** property of **this**.

I.E. **this.firstName** means the **firstName** property of **person**.

What is this?

In JavaScript, the this keyword refers to an **object**.

Which object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

In an object method, this refers to the **object**.

Alone, this refers to the **global object**.

In a function, this refers to the **global object**.

In a function, in strict mode, this is undefined.

In an event, this refers to the **element** that received the event.

Methods like call(), apply(), and bind() can refer this to **any object**.

Note

this is not a variable. It is a keyword. You cannot change the value of this.

See Also:

The this Keyword

In a function definition, this refers to the "owner" of the function.

In the example above, this is the **person object** that "owns" the fullName function.

In other words, this.firstName means the firstName property of **this object**.

Accessing Object Methods

You access an object method with the following syntax:

```
objectName.methodName()
```

Example

```
name = person.fullName();
```

If you access a method **without** the () parentheses, it will return the **function definition**:

Example

```
name = person.fullName;
```

Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
x = new String();    // Declares x as a String object
y = new Number();    // Declares y as a Number object
z = new Boolean();    // Declares z as a Boolean object
```

Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

JavaScript Objects

A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we directly create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

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

1) JavaScript Object by object literal

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

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

As you can see, property and value is separated by : (colon).

Let's see the simple example of creating object in JavaScript.

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Output of the above example

```
102 Shyam Kumar 40000
```

2) By creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, **new keyword** is used to create object.

Let's see the example of creating object directly.

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Output of the above example

101 Ravi 50000

3) By using an Object constructor

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

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

```
<script>
function emp(id,name,salary)
{
  this.id=id;
  this.name=name;
  this.salary=salary;
}
Emp();
e=new emp(103,"Vimal Jaiswal",30000);

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

Output of the above example

103 Vimal Jaiswal 30000

Defining method in JavaScript Object

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

The example of defining method in object is given below.

```
<script>
function emp(id,name,salary){
  this.id=id;
  this.name=name;
  this.salary=salary;

  this.changeSalary=changeSalary;
```



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

Output of the above example

103	Sonoo	Jaiswal	30000
103 Sonoo Jaiswal 45000			

JavaScript Date Object

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

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

Constructor

You can use 4 variant of Date constructor to create date object.

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

JavaScript Date Methods

Let's see the list of JavaScript date methods with their description.

Methods	Description	Example
<u>getDate()</u>	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("December 25, 1995 23:15:00"); document.write("getDate() : " + dt.getDate()); </script></pre> <p>getDate() : 25</p>

<u>getDay()</u>	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("December 25, 1995 23:15:00"); document.write("getDay() : " + dt.getDay()); </script></pre> <p>getDay() : 1</p>
<u>getFullYear()</u>	It returns the integer value that represents the year on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("December 25, 1995 23:15:00"); document.write("getFullYear() : " + dt.getFullYear()); </script></pre> <p>getFullYear() : 1995</p>
<u>getHours()</u>	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("December 25, 1995 23:15:00"); document.write("getHours() : " + dt.getHours()); </script></pre> <p>getHours() : 23</p>
<u>getMilliseconds()</u>	It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date(); document.write("getMilliseconds() : " + dt.getMilliseconds()); </script></pre> <p>getMilliseconds() : 632</p>
<u>getMinutes()</u>	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("December 25, 1995 23:15:00"); document.write("getMinutes() : " + dt.getMinutes()); </script></pre> <p>getMinutes() : 15</p>
<u>getMonth()</u>	It returns the integer value between 0 and 11 that represents the month on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("December 25, 1995 23:15:00"); document.write("getMonth() : " + dt.getMonth()); </script></pre> <p>getMonth() : 11</p>
<u>getSeconds()</u>	It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("December 25, 1995 23:15:20"); document.write("getSeconds() : " + dt.getSeconds()); </script></pre> <p>getSeconds() : 20</p>

setDate()	It sets the day value for the specified date on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("Aug 28, 2008 23:30:00"); dt.setDate(24); document.write(dt);</pre> Sun Aug 24 2008 23:30:00 GMT+0530 (India Standard Time)
setDay()	It sets the particular day of the week on the basis of local time.	
setFullYear()	It sets the year value for the specified date on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("Aug 28, 2008 23:30:00"); dt.setFullYear(2000); document.write(dt);</pre> Mon Aug 28 2000 23:30:00 GMT+0530 (India Standard Time)
<u>setHours()</u>	It sets the hour value for the specified date on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("Aug 28, 2008 23:30:00"); dt.setHours(02); document.write(dt);</pre> Thu Aug 28 2008 02:30:00 GMT+0530 (India Standard Time)
<u>setMilliseconds()</u>	It sets the millisecond value for the specified date on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("Aug 28, 2008 23:30:00"); dt.setMilliseconds(1010); document.write(dt);</pre> Thu Aug 28 2008 23:30:01 GMT+0530 (India Standard Time)
<u>setMinutes()</u>	It sets the minute value for the specified date on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("Aug 28, 2008 23:30:00"); dt.setMinutes(45); document.write(dt);</pre> Thu Aug 28 2008 23:45:00 GMT+0530 (India Standard Time)
setMonth()	It sets the month value for the specified date on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("Aug 28, 2008 23:30:00"); dt.setMonth(2); document.write(dt);</pre>

		Fri Mar 28 2008 23:30:00 GMT+0530 (India Standard Time)
<u>setSeconds()</u>	It sets the second value for the specified date on the basis of local time.	<pre><script type = "text/javascript"> var dt = new Date("Aug 28, 2008 23:30:00"); dt.setSeconds(80); document.write(dt); </script></pre>
		Thu Aug 28 2008 23:31:20 GMT+0530 (India Standard Time)
<u>toString()</u>	It returns the date in the form of string.	<pre><script type = "text/javascript"> var dateobject = new Date(1993, 6, 28, 14, 39, 7); stringobj = dateobject.toString(); document.write("String Object : " + stringobj); </script></pre>
		String Object : Wed Jul 28 1993 14:39:07 GMT+0530 (India Standard Time)
<u>Date()</u>	Returns today's date and time	<pre><script type = "text/javascript"> var dt = Date(); document.write("Date and Time : " + dt); </script></pre>
		Date and Time : Fri Aug 05 2022 21:16:18 GMT+0530 (India Standard Time)
<u>setTime()</u>	Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.	<pre><script type = "text/javascript"> var dt = new Date("Aug 28, 2008 23:30:00"); dt.setTime(5000000); document.write(dt); </script></pre>
		Thu Jan 01 1970 06:53:20 GMT+0530 (India Standard Time)

JavaScript Date Output

By default, JavaScript will use the browser's time zone and display a date as a full text string:

Mon Aug 01 2022 19:08:29 GMT+0530 (India Standard Time)

Creating Date Objects

Date objects are created with the new `Date()` constructor.

There are **4 ways** to create a new date object:

```
new Date()  
new Date(year, month, day, hours, minutes, seconds, milliseconds)  
new Date(milliseconds)  
new Date(date string)
```

new Date()

`new Date()` creates a new date object with the **current date and time**:

Example

```
const d = new Date();
```

Date objects are static. The computer time is ticking, but date objects are not.

new Date(year, month, ...)

`new Date(year, month, ...)` creates a new date object with a **specified date and time**.

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order):

Example

```
const d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

Note: JavaScript counts months from **0** to **11**:

January = 0.

December = 11.

Specifying a month higher than 11, will not result in an error but add the overflow to the next year:

Specifying:

```
const d = new Date(2018, 15, 24, 10, 33, 30);
```

Is the same as:

```
const d = new Date(2019, 3, 24, 10, 33, 30);
```

Specifying a day higher than max, will not result in an error but add the overflow to the next month:

Specifying:

```
const d = new Date(2018, 5, 35, 10, 33, 30);
```

Is the same as:

```
const d = new Date(2018, 6, 5, 10, 33, 30);
```

Using 6, 4, 3, or 2 Numbers

6 numbers specify year, month, day, hour, minute, second:

Example

```
const d = new Date(2018, 11, 24, 10, 33, 30);
```

5 numbers specify year, month, day, hour, and minute:

Example

```
const d = new Date(2018, 11, 24, 10, 33);
```

4 numbers specify year, month, day, and hour:

Example

```
const d = new Date(2018, 11, 24, 10);
```

3 numbers specify year, month, and day:

Example

```
const d = new Date(2018, 11, 24);
```

2 numbers specify year and month:

Example

```
const d = new Date(2018, 11);
```

You cannot omit month. If you supply only one parameter it will be treated as milliseconds.

Example

```
const d = new Date(2018);
```

new Date(*dateString*)

new Date(*dateString*) creates a new date object from a **date string**:

Example

```
const d = new Date("October 13, 2014 11:13:00");
```

JavaScript Stores Dates as Milliseconds

JavaScript stores dates as number of milliseconds since January 01, 1970, 00:00:00 UTC (Universal Time Coordinated).

Zero time is January 01, 1970 00:00:00 UTC.

Now the time is: **1659361109593** milliseconds past January 01, 1970

new Date(*milliseconds*)

new Date(*milliseconds*) creates a new date object as **zero time plus milliseconds**:

Example

```
const d = new Date(0);
```

01 January 1970 **plus** 100 000 000 000 milliseconds is approximately 03 March 1973:

Example

```
const d = new Date(100000000000);
```

January 01 1970 **minus** 100 000 000 000 milliseconds is approximately October 31 1966:

Example

```
const d = new Date(-100000000000);
```

Example

```
const d = new Date(86400000);
```

One day (24 hours) is 86 400 000 milliseconds.

JavaScript setTimeout() and clearTimeout() method

The **setTimeout()** method in JavaScript is used to execute a function after waiting for the specified time interval. This method returns a numeric value that represents the ID value of the timer.

Unlike the **setInterval()** method, the **setTimeout()** method executes the function only once. This method can be written with or without the **window** prefix.

We can use the **clearTimeout()** method to stop the timeout or to prevent the execution of the function specified in the **setTimeout()** method. The value returned by the **setTimeout()** method can be used as the argument of the **clearTimeout()** method to cancel the timer.

The commonly used syntax of the **setTimeout()** method is given below.

Syntax

```
window.setTimeout(function, milliseconds);
```

Parameter values

This method takes two parameter values **function** and **milliseconds** that are defined as follows.

function: It is the function containing the block of code that will be executed.

milliseconds: This parameter represents the time-interval after which the execution of the function takes place. The interval is in milliseconds. Its default value is 0. It defines how often the code will be executed. If it is not specified, the value **0** is used.

Let's understand the use of **setTimeout()** method by using some illustrations.

Example1

This is a simple example of using the **setTimeout()** method. Here, an alert dialog box will display at an interval of two seconds. We are not using any method to prevent the execution of the function specified in **setTimeout()** method. So the **setTimeout()** method executes the specified function only once, after the given time interval.

```
<html>
<head>
<title> setTimeout() method </title>
</head>
<body>
<h1> Hello World :) </h1>
<h3> This is an example of using the setTimeout() method </h3>
<p> Here, an alert dialog box will display after two seconds. </p>

<script>
var a;

a = setTimeout(fun, 2000);

function fun() {
alert(" Welcome to the javaTpoint.com ");
}
```


</script>

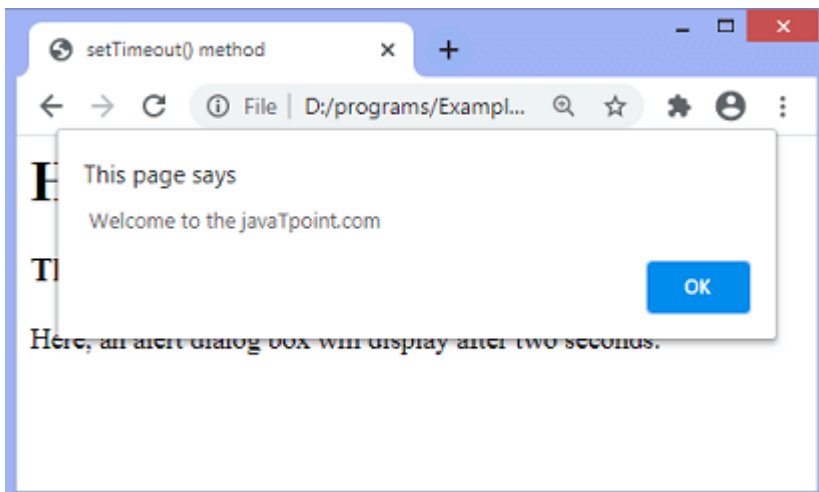
</body>

</html>

Output



After an interval of two seconds, the output will be -



Example2

In the above examples, we have not used any method to prevent the execution of function specified in **setTimeout()**. Here, we are using the **clearTimeout()** method to stop the function's execution.

We have to click the given **stop** button before two seconds to see the effect.

<html>

<head>

<title> setTimeout() method </title>

</head>

<body>

<h1> Hello World :) :) </h1>

<h3> This is an example of using the setTimeout() method </h3>

<p> Click the following button before 2 seconds to see the effect. </p>

<button onclick = "stop()"> Stop </button>

<script>

```
var a = setTimeout(fun1, 2000);
function fun1()
{
var win1 = window.open();
win1.document.write(" <b2> Welcome to the javaTpoint.com </b2>");
setTimeout(function(){win1.close()}, 2000);
}
function stop() {
clearTimeout(a);
}
```

</script>

</body>

</html>

Output



The output will remain same if the user clicks the **stop** button before two seconds. Otherwise, a new tab will open after two seconds and close after two seconds of opening.

JavaScript setInterval() and clearInterval() method

The **setInterval()** method in JavaScript is used to repeat a specified function at every given time-interval. It evaluates an expression or calls a function at given intervals. This method continues the calling of function until the window is closed or the **clearInterval()** method is called. This method returns a numeric value or a non-zero number that identifies the created timer.

Unlike the **setTimeout()** method, the **setInterval()** method invokes the function multiple times. This method can be written with or without the **window** prefix.

The commonly used syntax of **setInterval()** method is given below:

Syntax

```
window.setInterval(function, milliseconds);
```

Parameter values

This method takes two parameter values ***function*** and ***milliseconds*** that are defined as follows.

function: It is the function containing the block of code that will be executed.

milliseconds: This parameter represents the length of the time interval between each execution. The interval is in milliseconds. It defines how often the code will be executed. If its value is less than 10, the value 10 is used.

How to stop the execution?

We can use the **clearInterval()** method to stop the execution of the function specified in **setInterval()** method. The value returned by the **setInterval()** method can be used as the argument of **clearInterval()** method to cancel the timeout.

Let's understand the use of **setInterval()** method by using some illustrations.

Example1

This is a simple example of using the **setInterval()** method. Here, an alert dialog box displays at an interval of 3 seconds. We are not using any method to stop the execution of the function specified in **setInterval()** method. So the method continues the execution of the function until the window is closed.

```
<html>
<head>
<title> setInterval() method </title>
</head>
<body>
<h1> Hello World :) :) </h1>
<h3> This is an example of using the setInterval() method </h3>
<p> Here, an alert dialog box displays on every three seconds. </p>

<script>

var a;

a = setInterval(fun, 3000);

function fun() {
alert(" Welcome to the javaTpoint.com ");
}</script>

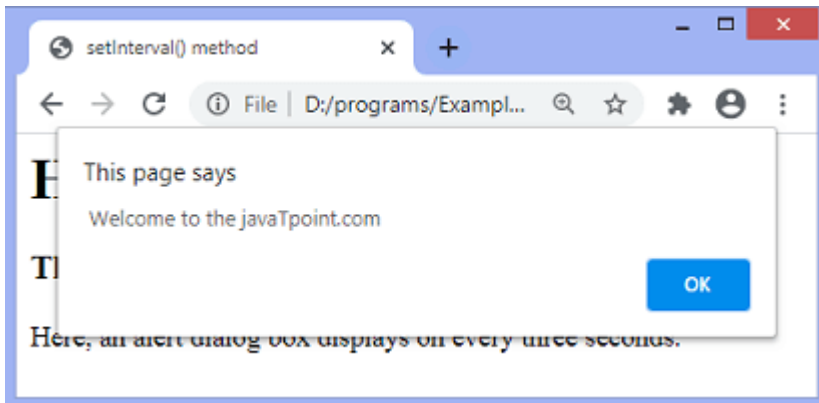
</body>

</html>
```

Output



After the time interval of three seconds, the output will be -



Now, there is another example of using the **setInterval()** method.

Example2

Here, the background color will change on every 200 milliseconds. We are not using any method to stop the execution of the function specified in **setInterval()** method. So the method continues the execution of the function until the window is closed.

```
<html>
<head>
<title> setInterval() method </title>
</head>
<body>
<h1> Hello World :) :) </h1>
<h3> This is an example of using the setInterval() method </h3>
<p> Here, the background color changes on every 200 milliseconds. </p>

<script>
var var1 = setInterval(color, 200);

function color() {
var var2 = document.body;
var2var2.style.backgroundColor = var2.style.backgroundColor == "lightblue" ? "lightgreen" : "lightblue";
}

</script>

</body>
```

</html>

Output



The background will keep changing from **lightgreen** to **lightblue** on an interval of 200 milliseconds. After 200 milliseconds, the output will be -



Example3

In the above example, we have not used any method to stop the toggling between the colors. Here, we are using the **clearInterval()** method to stop the toggling of colors in the previous example.

We have to click the specified **stop** button to see the effect.

```
<html>
<head>
<title> setInterval() method </title>
</head>
<body>
<h1> Hello World :) :) </h1>
<h3> This is an example of using the setInterval() method </h3>
<p> Here, the background color changes on every 200 milliseconds. </p>
<button onclick = "stop()"> Stop </button>
```

```
<script>
var var1 = setInterval(color, 200);

function color() {
var var2 = document.body;
var2var2.style.backgroundColor = var2.style.backgroundColor == "lightblue" ? "lightgreen" : "lightblue";
}
function stop() {
clearInterval(var1);
}
```

</script>

</body>

</html>

Output



The color of the background will start changing after 200 milliseconds. On clicking the specified **stop** button, the toggling between the colors will be stopped on the corresponding background color. The output after clicking the button will be -



Document Object Model (DOM model)

The **document object** represents the whole html document.

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

As mentioned earlier, it is the object of window. So

```
window.document
```

Is same as

```
document
```

According to W3C - *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

DOM (Document Object Model)

The Document Object Model (DOM) is a **programming interface** for **HTML(HyperText Markup Language)** and **XML(Extensible markup language)** documents. It defines the **logical structure** of documents and the way a document is accessed and manipulated.

Note: It is called a Logical structure because DOM doesn't specify any relationship between objects.

DOM is a way to represent the webpage in a structured hierarchical way so that it will become easier for programmers and users to glide through the document. With DOM, we can easily access and manipulate tags, IDs, classes, Attributes, or Elements of HTML using commands or methods provided by the Document object. Using DOM, the JavaScript gets access to HTML as well as CSS of the web page and can also add behavior to the HTML elements. so basically **Document Object Model is an API that represents and interacts with HTML or XML documents.**

Why DOM is required?

HTML is used to **structure** the web pages and Javascript is used to add **behavior** to our web pages. When an HTML file is loaded into the browser, the javascript can not understand the HTML document directly. So, a corresponding document is created(DOM). **DOM is basically the representation of the same HTML document but in a different format with the use of objects.** Javascript interprets DOM easily i.e javascript can not understand the tags(<h1>H</h1>) in HTML document but can understand object h1 in DOM. Now, Javascript can access each of the objects (h1, p, etc) by using different functions.

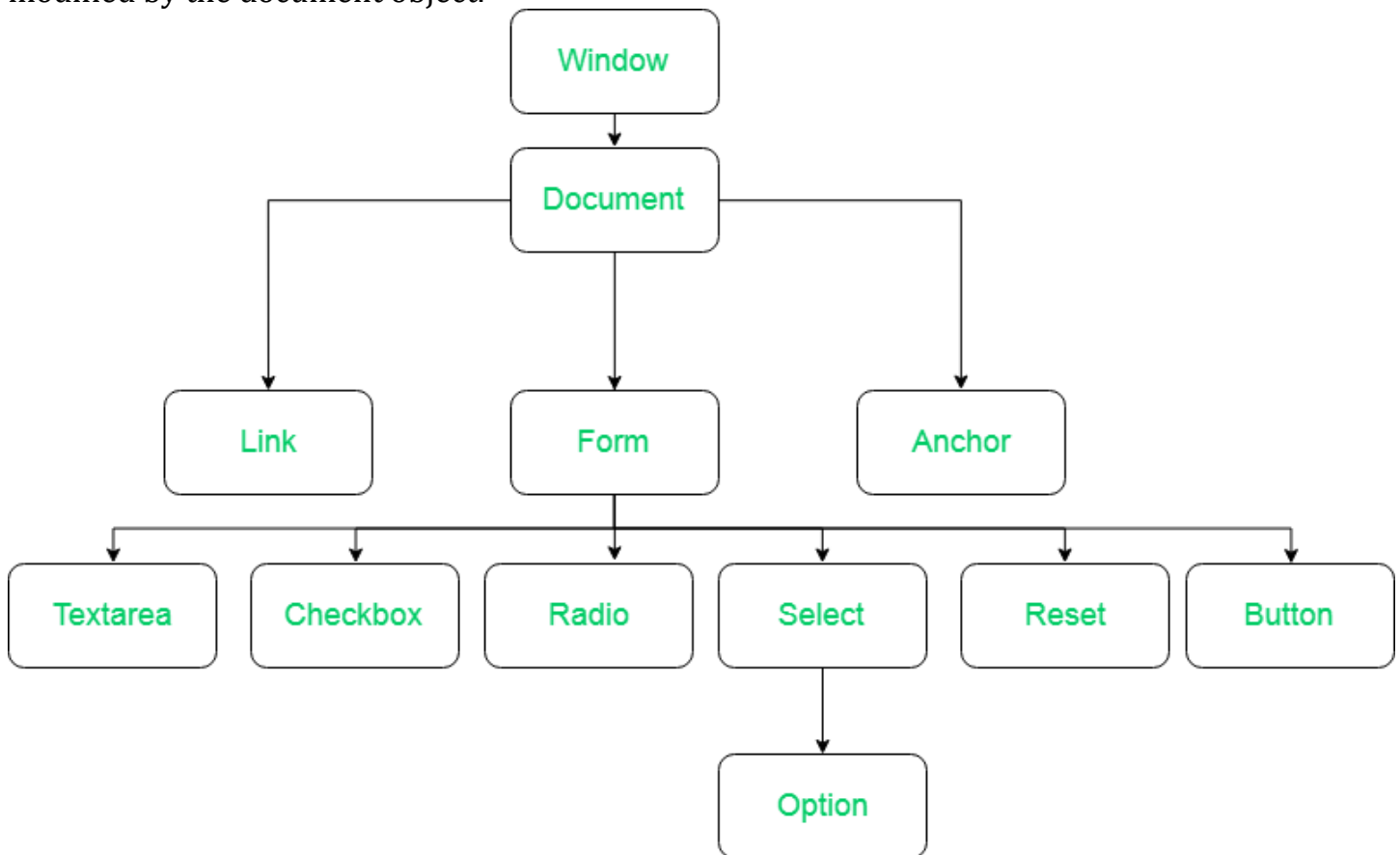
Structure of DOM: DOM can be thought of as a Tree or Forest(more than one tree). The term **structure model** is sometimes used to describe the tree-like representation of a document. Each branch of the tree ends in a node, and each node contains objects Event listeners

can be added to nodes and triggered on an occurrence of a given event. One important property of DOM structure models is **structural isomorphism**: if any two DOM implementations are used to create a representation of the same document, they will create the same structure model, with precisely the same objects and relationships.

Why called an Object Model?

Documents are modeled using objects, and the model includes not only the structure of a document but also the behavior of a document and the objects of which it is composed like tag elements with attributes in HTML.

Properties of DOM: Let's see the properties of the document object that can be accessed and modified by the document object.



Representation of the DOM

- **Window Object:** Window Object is object of the browser which is always at top of the hierarchy. It is like an API that is used to set and access all the properties and methods of the browser. It is automatically created by the browser.
- **Document object:** When an HTML document is loaded into a window, it becomes a document object. The 'document' object has various properties that refer to other objects which allow access to and modification of the content of the web page. If there is a need to access any element in an HTML page, we always start with accessing the 'document' object. Document object is property of window object.
- **Form Object:** It is represented by *form* tags.
- **Link Object:** It is represented by *link* tags.
- **Anchor Object:** It is represented by *a href* tags.
- **Form Control Elements::** Form can have many control elements such as text fields, buttons, radio buttons, checkboxes, etc.

Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

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

Accessing field value by document object

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

Here, **document** is the root element that represents the html document.

form1 is the name of the form.

name is the attribute name of the input text.

value is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>
```

```
<form name="form1">
Enter Name:<input type="text" name="name"/>
<input type="button" onclick="printvalue()" value="print name"/>
</form>
```

Output of the above example

Enter Name: khyati

print name

This page says

Welcome: khyati

OK

Example: In this example, We use HTML element id to find the DOM HTML element.

- HTML

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>GeeksforGeeks</h2>
```

```
<!-- Finding the HTML Elements by their Id in DOM -->
```

```
<p id="intro">A Computer Science portal for geeks.</p>
```

```
<p>This example illustrates the <b>getElementById</b> method.</p>
```

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

```
<script>
```

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

```
document.getElementById("demo").innerHTML = "GeeksforGeeks introduction is: " +  
element.innerHTML;
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:

GeeksforGeeks

A Computer Science portal for geeks.

This example illustrates the **getElementById** method.

GeeksforGeeks introduction is: A Computer Science portal for geeks.

Getting the HTML element by getElementById() Method

Javascript - document.getElementById() method

The **document.getElementById()** method returns the element of specified id.

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

Let's see the simple example of **document.getElementById()** method that prints cube of the given number.

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

Output of the above example



Javascript - document.getElementsByName() method

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the getElementsByName() method is given below:

```
document.getElementsByName("name")
```

Here, name is required.

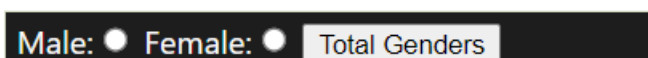
Example of document.getElementsByName() method

In this example, we going to count total number of genders. Here, we are using getElementsByName() method to get all the genders.

```
<script type="text/javascript">
function totalelements()
{
var allgenders=document.getElementsByName("gender");
alert("Total Genders:"+allgenders.length);
}
</script>
<form>
Male:<input type="radio" name="gender" value="male">
Female:<input type="radio" name="gender" value="female">

<input type="button" onclick="totalelements()" value="Total Genders">
</form>
```

Output of the above example



Javascript-document.getElementsByTagName() method

The **document.getElementsByTagName()** method returns all the element of specified tag name.

The syntax of the getElementsByTagName() method is given below:

```
document.getElementsByTagName("name")
```

Here, name is required.

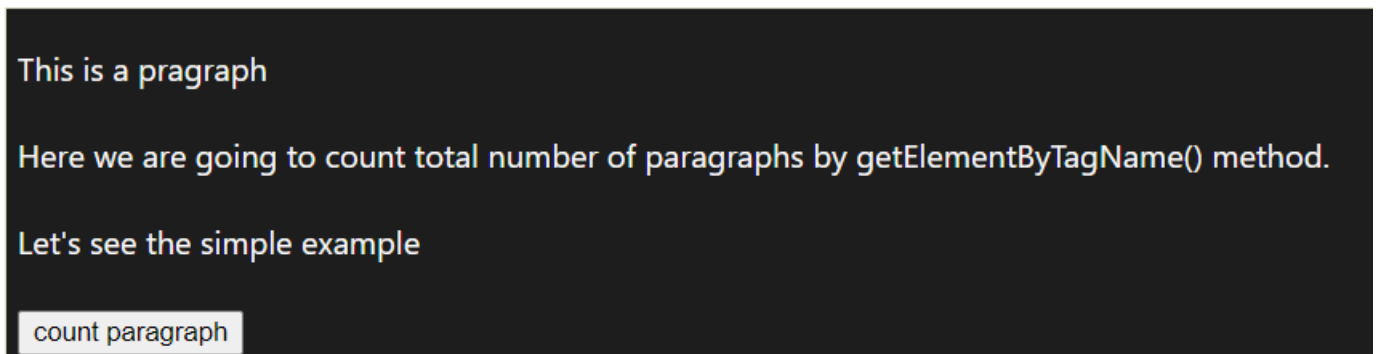
Example of document.getElementsByTagName() method

In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

```
<script type="text/javascript">
function countpara(){
var totalpara=document.getElementsByTagName("p");
alert("total p tags are: "+totalpara.length);

}
</script>
<p>This is a paragraph</p>
<p>Here we are going to count total number of paragraphs by getElementByTagName() method.</p>
<p>Let's see the simple example</p>
<button onclick="countpara()">count paragraph</button>
```

Output of the above example



Another example of document.getElementsByTagName() method

In this example, we going to count total number of h2 and h3 tags used in the document.

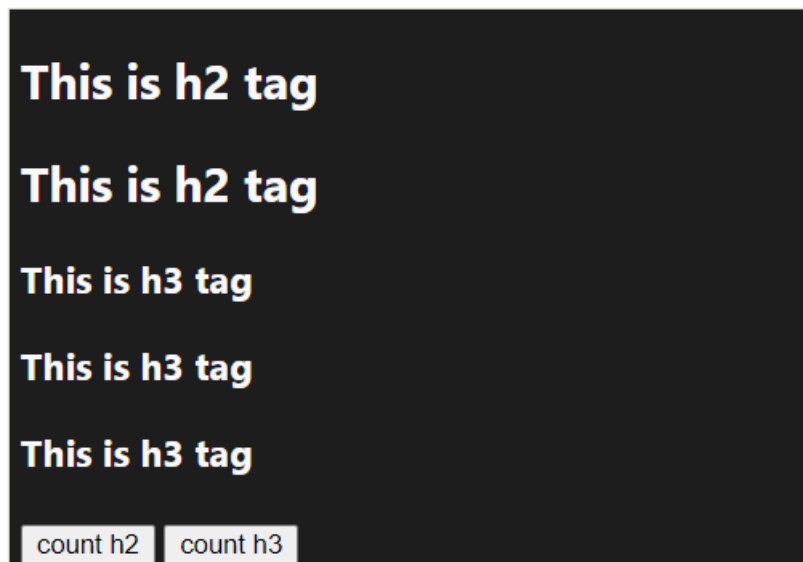
```
<script type="text/javascript">
function counth2(){
var totalh2=document.getElementsByTagName("h2");
alert("total h2 tags are: "+totalh2.length);
}
function counth3(){
var totalh3=document.getElementsByTagName("h3");
```

```

alert("total h3 tags are: "+totalh3.length);
}
</script>
<h2>This is h2 tag</h2>
<h2>This is h2 tag</h2>
<h3>This is h3 tag</h3>
<h3>This is h3 tag</h3>
<h3>This is h3 tag</h3>
<button onclick="counth2()">count h2</button>
<button onclick="counth3()">count h3</button>

```

Output of the above example



Note: Output of the given examples may differ on this page because it will count the total number of para , total number of h2 and total number of h3 tags used in this document.

Javascript - innerHTML

The **innerHTML** property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example of innerHTML property

In this example, we are going to create the html form when user clicks on the button.

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifying this position by calling the document.getElementById() method.

```

<script type="text/javascript" >
function showcommentform() {
var data="Name:<input type='text' name='name'><br>Comment:<br><textarea rows='5' cols='80'></textare
a>
<br><input type='submit' value='Post Comment'>";
document.getElementById('mylocation').innerHTML=data;
}

```

```

</script>
<form name="myForm">
<input type="button" value="comment" onclick="showcommentform()">
<div id="mylocation"></div>
</form>

```

Name:

 Comment:

Javascript - innerText

The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

Javascript innerText Example

In this example, we are going to display the password strength when releases the key after press.

```

<script type="text/javascript" >
function validate() {
var msg;
if(document.myForm.userPass.value.length>5){
msg="good";
}
else{
msg="poor";
}
document.getElementById('mylocation').innerText=msg;
}
</script>
<form name="myForm">
<input type="password" value="" name="userPass" onkeyup="validate()">
Strength:<span id="mylocation">no strength</span>
</form>

```

Output of the above example

Strength:no strength

HTML DOM Document write()

Definition and Usage

The `write()` method writes directly to an open (HTML) document stream.

Warning

The `write()` method deletes all existing HTML when used on a loaded document.

The `write()` method cannot be used in XHTML or XML.

Note

The `write()` method is most often used to write to output streams opened by the `open()` method.

Syntax

```
document.write(exp1, exp2, ..., expN)
```

Parameters

Parameter	Description
<i>exp1</i> ,...	Optional. The output stream. Multiple arguments are appended to the document in order of occurrence.

Return Value

NONE

Examples

Write some text directly to the HTML output:

```
document.write("Hello World!");
```

Write some HTML elements directly to the HTML output:

```
document.write("<h2>Hello World!</h2><p>Have a nice day!</p>");
```

More Examples

Write a date object directly to the HTML output:

```
document.write(Date());
```

Open an output stream, add some HTML, then close the output stream:

```
document.open();  
document.write("<h1>Hello World</h1>");  
document.close();
```

Open a new window and write some HTML into it:

```
const myWindow = window.open();  
myWindow.document.write("<h1>New Window</h1>");  
myWindow.document.write("<p>Hello World!</p>");
```

HTML DOM Document `writeln()`

Definition and Usage

The `writeln()` method writes directly to an open (HTML) document stream.

The `writeln()` method is identical to the `write()` method, with the addition of writing a newline character (U+000A) after each statement.

Warning

The `writeln()` method deletes all existing HTML when used on a loaded document.

The `writeln()` method cannot be used in XHTML or XML.

Example

```
document.writeln("Hello World!");  
document.writeln("Have a nice day!");
```

Note

It makes no sense to use **`writeln()`** in HTML.

It is only useful when writing to text documents (type=".txt").

Newline characters are ignored in HTML.

If you want new lines in HTML, you must use paragraphs or
:

Examples

```
document.write("Hello World!");  
document.write("<br>");  
document.write("Have a nice day!");
```

Syntax

```
document.writeln(exp1, exp2, ..., expN)
```

Parameters

Parameter	Description
<i>exp1,...</i>	Optional. The output stream. Multiple arguments are appended to the document in order of occurrence.

Return Value

NONE

The Difference Between write() and writeln()

The writeln() method is only useful when writing to text documents (type=".txt").

Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>The Document Object</h1>
```

```
<h2>The write() and writeln() Methods</h2>
```

```
<p>write() does NOT add a new line (CR) after each statement.</p>
```

```
<p>writeln() DOES add a new line (CR) after each statement.</p>
```

```
<pre>
<script>
document.write("Hello World!");
document.write("Have a nice day!");
document.write("<br>");
document.writeln("Hello World!");
document.writeln("Have a nice day!");
</script>
</pre>

<p>It makes no sense to use writeln() in HTML.</p>
<p>Carriage return (CR) is ignored in HTML.</p>

</body>
</html>
```

The Document Object

The write() and writeln() Methods

write() does NOT add a new line (CR) after each statement.

writeln() DOES add a new line (CR) after each statement.

```
Hello World!Have a nice day!
Hello World!
Have a nice day!
```

It makes no sense to use writeln() in HTML.

Carriage return (CR) is ignored in HTML.

JavaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

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

JavaScript Function Syntax

The syntax of declaring function is given below.

```
function functionName([arg1, arg2, ...argN]){  
  //code to be executed  
}
```

JavaScript Functions can have 0 or more arguments.

JavaScript Function Example

Let's see the simple example of function in JavaScript that does not has arguments.

```
<script>  
function msg(){  
  alert("hello! this is message");  
}  
</script>  
  
<input type="button" onclick="msg()" value="call function"/>
```

Output of the above example



JavaScript Function Arguments

We can call function by passing arguments. Let's see the example of function that has one argument.

```
<script>  
function getcube(number){  
  alert(number*number*number);  
}  
</script>
```

```
<form>
<input type="button" value="click" onclick="getcube(4)"/>
</form>
```

Output of the above example



Function with Return Value

We can call function that returns a value and use it in our program. Let's see the example of function that returns value.

```
<script>
function getInfo(){
return "hello javatpoint! How r u?";
}
</script>
<script>
document.write(getInfo());
</script>
```

Output of the above example

hello javatpoint! How r u?

Dialog boxes

There are three types of dialog boxes supported in JavaScript that are **alert**, **confirm**, and **prompt**. These dialog boxes can be used to perform specific tasks such as raise an alert, to get confirmation of an event or an input, and to get input from the user.

Let's discuss each dialog box.

Alert Dialog box

It is used to provide a **warning message** to users. It is one of the most widely used dialog box in JavaScript. It has only one **'OK'** button to continue and select the next task.

We can understand it by an example like suppose a textfield is mandatory to be filled out, but the user has not given any input value to that text field, then we can display a warning message by using the **alert box**.

Syntax

```
alert(message);
```

Example

Let us see the demonstration of an alert dialog box by using the following example.

```
<html>

<head>
  <script type="text/javascript">
    function show() {
      alert("It is an Alert dialog box");
    }
  </script>
</head>

<body>
  <center>
    <h1>Hello World :) :)</h1>
    <h2>Welcome to javaTpoint</h2>
    <p>Click the following button </p>
    <input type="button" value="Click Me" onclick="show();" />
  </center>
</body>

</html>
```

Output

After the successful execution of the above code, you will get the following output.

Hello World :) :)

Welcome to javaTpoint

Click the following button

After clicking on the **Click Me** button, you will get the following output:

This page says

It is an Alert dialog box

Click the following button

Confirmation Dialog box

It is widely used for **taking the opinion from the user on the specific option**. It includes two buttons, which are **OK** and **Cancel**. As an example, suppose a user is required to delete some data, then the page can confirm it by using the confirmation box that whether he/she wants to delete it or not.

If a user clicks on the **OK** button, then the method **confirm()** returns **true**. But if the user clicks on the **cancel** button, then the **confirm() method** returns false.

Syntax

```
confirm(message);
```

Example

Let us understand the demonstration of this dialog box by using the following example.

```
<html>

<head>
  <script type="text/javascript">
    function show() {
      var con = confirm ("It is a Confirm dialog box");
      if(con == true) {
        document.write ("User Want to continue");
      }
      else {
        document.write ("User does not want to continue");
      }
    }
  </script>
</head>

<body>
  <center>
    <h1>Hello World :)</h1>
    <h2>Welcome to javaTpoint</h2>
    <p>Click the following button </p>
    <input type="button" value="Click Me" onclick="show();" />
  </center>
</body>

</html>
```

Output

After the successful execution of the above code, you will get the following output.

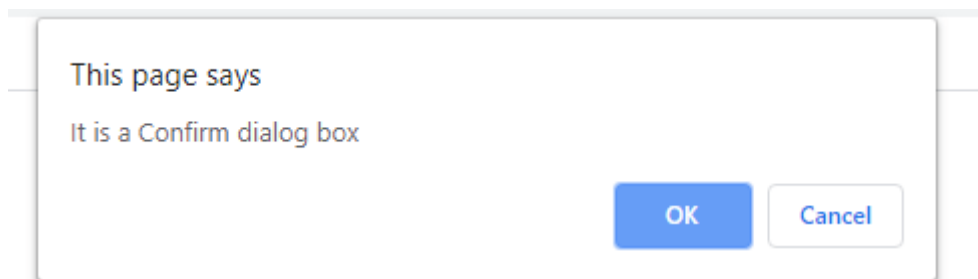
Hello World :) :)

Welcome to javaTpoint

Click the following button

Click Me

When you click on the given button, then you will get the following output:



Click the following button

Click Me

After clicking the **OK** button, you will get:

User Want to continue

On clicking the **Cancel** button, you will get:

User does not want to continue

Prompt Dialog box

The prompt dialog box is used when it is required to pop-up a text box for getting the user input. Thus, it enables interaction with the user.

The prompt dialog box also has two buttons, which are **OK** and **Cancel**. The user needs to provide input in the textbox and then click OK. When a user clicks on the OK button, then the dialog box reads that value and returns it to the user. But on clicking the **Cancel** button, **prompt()** method returns **null**.

Syntax

```
prompt(message, default_string);
```

Let us understand the prompt dialog box by using the following illustration.

Example

```
<html>

<head>
  <script type="text/javascript">
    function show() {
      var value = prompt("Enter your Name : ", "Enter your name");
      document.write("Your Name is : " + value);
    }
  </script>
</head>

<body>
  <center>
    <h1>Hello World :)</h1>
    <h2>Welcome to javaTpoint</h2>
    <p>Click the following button </p>
    <input type="button" value="Click Me" onclick="show();" />
  </center>
</body>

</html>
```

Output

After executing the above code successfully, you will get the following output.

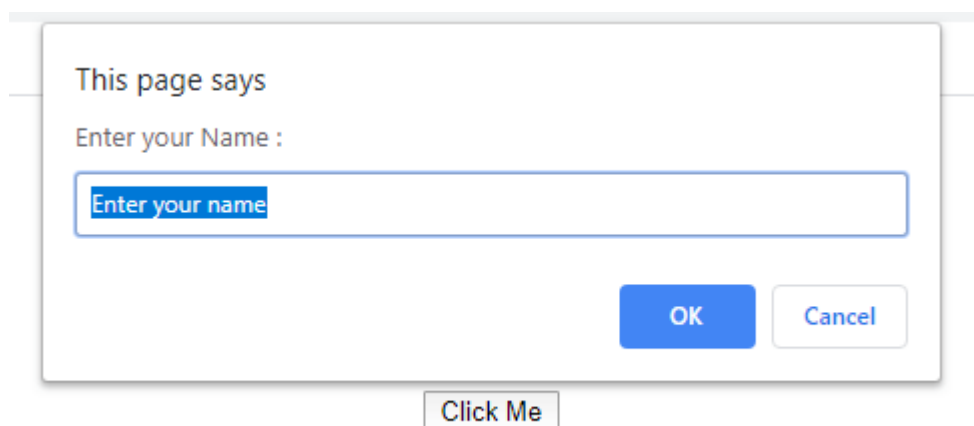
Hello World :) :)

Welcome to javaTpoint

Click the following button

Click Me

When you click on the **Click Me** button, you will get the following output:



Enter your name and click OK button, you will get:

Your Name is : javaTpoint