UNIT # 1: HTML5

❖ INTRODUCTION

HTML stands for Hyper Text Markup Language. It is used to design web pages using a markup language. HTML is an abbreviation of Hypertext and Markup language. Hypertext defines the link between the web pages. The markup language is used to define the text document within the tag which defines the structure of web pages. HTML 5 is the fifth and current version of HTML. It has improved the markup available for documents and has introduced application programming interfaces (API) and Document Object Model (DOM).

HTML5 is the latest and most enhanced version of HTML. Technically, HTML is not a programming language, but rather a markup language.

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.

HTML5 is cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

Browser Support

The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5 functionality.

The mobile web browsers that come pre-installed on iPhones, iPads, and Android phones all have excellent support for HTML5.

New Features

HTML5 introduces a number of new elements and attributes that can help you in building modern websites. Here is a set of some of the most prominent features introduced in HTML5.

1. New Semantic Elements − These are like <header>, <footer>, and <section>.
2. Forms 2.0 − Improvements to HTML web forms where new attributes have been introduced for <input> tag.
3. Persistent Local Storage − To achieve without resorting to third-party plugins.
4. WebSocket − A next-generation bidirectional communication technology for web applications.
5. Server-Sent Events − HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).

6. Canvas − This supports a two-dimensional drawing surface that you can program with JavaScript.
7. Audio & Video − You can embed audio or video on your webpages without resorting to third-party plugins.
8. Geolocation − Now visitors can choose to share their physical location with your web application.
9. Microdata − This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
10. Drag and drop − Drag and drop the items from one location to another location on the same webpage.

Backward Compatibility

HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. Its new features have been built on existing features and allow you to provide fallback content for older browsers.

It is suggested to detect support for individual HTML5 features using a few lines of JavaScript

Advantages:

1. All browsers supported.
2. More device friendly.
3. Easy to use and implement.
4. HTML 5 in integration with CSS, JavaScript, etc. can help build beautiful websites.

Disadvantages:

1. Long codes have to be written which is time consuming.
2. Only modern browsers support it.

- **Common infrastructure**

This specification refers to both HTML and XML attributes and IDL attributes, often in the same context. When it is not clear which is being referred to, they are referred to as content attributes for HTML and XML attributes, and IDL attributes for those defined on IDL interfaces. Similarly, the term "properties" is used for both JavaScript object properties and CSS properties. When these are ambiguous they are qualified as object properties and CSS properties respectively.

Generally, when the specification states that a feature applies to the HTML syntax or the XML syntax, it also includes the other. When a feature specifically only applies to one of the two languages, it is called out by explicitly stating that it does not apply to the other format, as in "for HTML, ... (this does not apply to XML)".

This specification uses the term document to refer to any use of HTML, ranging from short static documents to long essays or reports with rich multimedia, as well as to fully-fledged interactive applications. The term is used to refer both to Document objects and their descendant DOM

trees, and to serialized byte streams using the HTML syntax or the XML syntax, depending on context.

In the context of the DOM structures, the terms HTML document and XML document are used as defined in *DOM*, and refer specifically to two different modes that Document objects can find themselves in. [DOM]

In the context of byte streams, the term HTML document refers to resources labeled as text/html, and the term XML document refers to resources labeled with an XML MIME type.

❖ HTML5 Semantics

In any language, it is essential to understand the meaning of words during communication. And if this is a computer communication then it becomes more critical. So HTML5 provides more semantic elements which make easy understanding of the code.

Hence Semantics defines the meaning of words and phrases, i.e. Semantic elements = elements with a meaning.

Semantic elements have a simple and clear meaning for both, the browser and the developer.

Semantic HTML elements are those that clearly describe their meaning in a human- and machine-readable way.

Elements such as <header>, <footer> and <article> are all considered semantic because they accurately describe the purpose of the element and the type of content that is inside them.

HTML was originally created as a markup language to describe documents on the early internet. As the internet grew and was adopted by more people, its needs changed.

Where the internet was originally intended for sharing scientific documents, now people wanted to share other things as well. Very quickly, people started wanting to make the web look nicer.

Because the web was not initially built to be designed, programmers used different hacks to get things laid out in different ways. Rather than using the <table></table> to describe information using a table, programmers would use them to position other elements on a page.
As the use of visually designed layouts progressed, programmers started to use a generic "non-semantic" tag like . They would often give these elements a class or id attribute to describe their purpose. For example, instead of <header> this was often written as .

As HTML5 is still relatively new, this use of non-semantic elements is still very common on websites today

For example:

In HTML4 we have seen , etc. are which are non-semantic elements. They don't tell anything about its content.

On the other hand, <form>, <table>, and <article> etc. are semantic elements because they clearly define their content.

HTML5 semantic elements are supported by all major browsers.

Why to use semantic elements?

In HTML4, developers have to use their own id/class names to style elements: header, top, bottom, footer, menu, navigation, main, container, content, article, sidebar, topnav, etc.

This is so difficult for search engines to identify the correct web page content. Now in HTML5 elements (<header> <footer> <nav> <section> <article>), this will become easier. It now allows data to be shared and reused across applications, enterprises, and communities."

Semantic elements can increase the accessibility of your website, and also helps to create a better website structure.

Semantic Elements in HTML5

Index	Semantic Tag	Description
1.	<article>	Defines an article
2.	<aside>	Defines content aside from the page content
3.	<details>	Defines additional details that the user can view or hide
4.	<figcaption>	Defines a caption for a <figure> element
5.	<figure>	Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
6.	<footer>	Defines a footer for a document or section
7.	<header>	Specifies a header for a document or section
8.	<main>	Specifies the main content of a document
9.	<mark>	Defines marked/highlighted text
10.	<nav>	Defines navigation links
11.	<section>	Defines a section in a document
12.	<summary>	Defines a visible heading for a <details> element
13.	<time>	Defines a date/time

Elements such as <header>, <nav>, <section>, <article>, <aside>, and <footer> act more or less like elements. They group other elements together into page sections. However where a tag could contain any type of information, it is easy to identify what sort of information would go in a semantic <header> region.

Some important semantic elements in HTML5

- **HTML5 <article> Element**

HTML <article> element defines article content within a document, page, application, or a website. It can be used to represent a forum post, a magazine, a newspaper article, or a big story.

Example:

```
<article>
   <h2>Today's highlights</h2>
   <p>First story</p>
   <p>Second story</p>
   <p>Third story</p>
 </article>
```

- **HTML5 <aside> Element**

The <aside> element represent the content which is indirectly giving information to the main content of the page. It is frequently represented as a sidebar.

Example:

```
<body>
<h2>My last year memories</h2>
<p>I have visited Paris with my friends last month. This was the memorable journ
ey and i wish to go there again.</p>
  <aside>
    <h4>Paris</h4>
    <p>Paris, France's capital, is a major European city and a global center for art, f
ashion, gastronomy and culture</p>
  </aside>
</body>
```

- **HTML5 <section> Element**

The <section> element is used to represent the standalone section within an HTML document. A page can have various sections and each section can contain any content, but headings for each section is not mandatory.

Example:

```
<h2>Web designing Tutorial</h2>
<section>
   <h3>HTML</h3>
```

```
    <p>HTML is an acronym which stands for Hyper Text Markup Language whic
h is used for creating web pages and web applications.</p>
 </section>
 <section>
    <h3>CSS</h3>
    <p>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 langua
ge. It provides an additional feature to HTML.</p>
 </section>
```

- **Nesting <article> tag in <section> tag or Vice Versa?**

We know that the<article> element specifies independent, self-contained content and the <section> element defines section in a document.

In HTML, we can use <section> elements within <article> elements, and <article> elements within <section> elements.

We can also use <section> elements within <section> elements, and <article> elements within <article> elements.

Example:

In a newspaper, the sport <article> in the sport section, may have a technical section in each <article>.

- **HTML5 <nav> Element**

The HTML <nav> element is used to define a set of navigation links.

Example:

```
<!DOCTYPE html>
<html>
<body>
<nav>
  <a href="https://www.javatpoint.com/html-tutorial">HTML</a> |
  <a href="https://www.javatpoint.com/java-tutorial">Java</a> |
  <a href="https://www.javatpoint.com/php-tutorial">PHP</a> |
  <a href="https://www.javatpoint.com/css-tutorial">CSS</a>
</nav>
</body>
</html>
```

- **HTML5 <header> Element**

The <header> element represent the header of the document which can contain introductory content or navigation links.

Example:

```
<header>
  <h1>Welcome to Web123.com</h1>
  <nav>
    <ul>
      <li>Home |</li>
      <li>About us |</li>
      <li>Contact us</li>
    </ul>
  </nav>
</header>
```

- **HTML5 <footer> Element**

The <footer> tag defines the footer of an HTML document or page.

Example:

```
<footer>
  <p>© Copyright 2019. All rights reserved. </p>
</footer>
```

Why use semantic elements?

To look at the benefits of semantic elements, here are two pieces of HTML code. This first block of code uses semantic elements:

```
<header></header>
<section>
        <article>
                <figure>
                        <img>
                        <figcaption></figcaption>
                </figure>
        </article>
</section>
<footer></footer>
```

While this second block of code uses non-semantic elements:

```
<div id="header"></div>
<div class="section">
        <div class="article">
                <div class="figure">
                        <img>
                        <div class="figcaption"></div>
                </div>
```

```
        </div>
    </div>
    <div id="footer"></div>
```

Advantages:

1. Easier to read
2. Greater accessibility
3. Consistent code.

- **HTML5 Document Structure**

The HTML 5 language has a "custom" HTML syntax that is compatible with HTML 4 and XHTML1 documents published on the Web, but is not compatible with the more esoteric SGML features of HTML 4.

HTML 5 does not have the same syntax rules as XHTML where we needed lower case tag names, quoting our attributes, an attribute had to have a value and to close all empty elements.

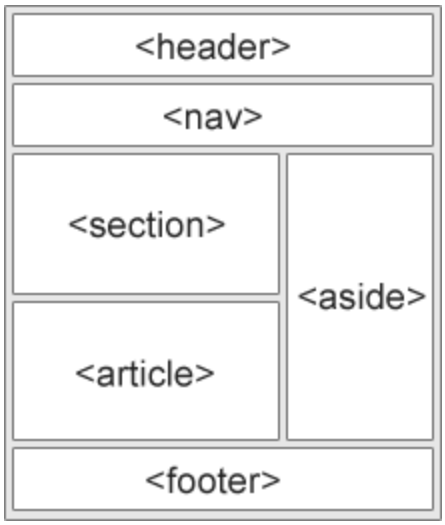
HTML5 comes with a lot of flexibility and it supports the following features −

1. Uppercase tag names.
2. Quotes are optional for attributes.
3. Attribute values are optional.
4. Closing empty elements are optional.

An HTML 5 document mainly consists of a Head and Body. The Head contains the data, which informs the browser and even web servers that it is an HTML 5 document. On the other hand, the Body contains content that web browsers actually display.

HTML Layout Elements

HTML has several semantic elements that define the different parts of a web page:



1. <header> - Defines a header for a document or a section
2. <nav> - Defines a set of navigation links
3. <section> - Defines a section in a document
4. <article> - Defines an independent, self-contained content
5. <aside> - Defines content aside from the content (like a sidebar)
6. <footer> - Defines a footer for a document or a section
7. <details> - Defines additional details that the user can open and close on demand
8. <summary> - Defines a heading for the <details> element

Establishing a Document Structure

Each and every HTML 5 document employs a unique combination of elements and content to define a page. The structure of all the property documented pages is the same and contains:

1. A declaration at the top, which indicates that it is an HTML 5 document
2. A document header
3. A document body

A collection of HTML 5 elements constitutes an HTML 5 document. Some of these elements are essential while others are optional. However, you can always find the following three elements on every page in addition to the DOC Type declaration at the top.

1. **<!DOCTYPE>** informs the browsers that it is actually an HTML 5 document. Although there are other types of DOC Types, this is the most commonly used declaration.
2. The DOCType Declaration is followed by **<html></html>** opening and closing tags. These tags contain everything inside the document, including the Head and Body
3. **<head> </head>** opening and closing tags, follow the opening Html tag. These tags contain information about the body, title of the page, definitions, labels, etc. You can only use certain markup elements in the HTML 5 head. Some of these elements include

style, title, base, link, script, and meta. In HTML 5, these elements are collectively known as HTML Head Elements.

4. After the closing head tag is the **<body> </body>** opening and closing body tags. They contain all the content which appears on the browser, as well as the related HTML 5 codes.

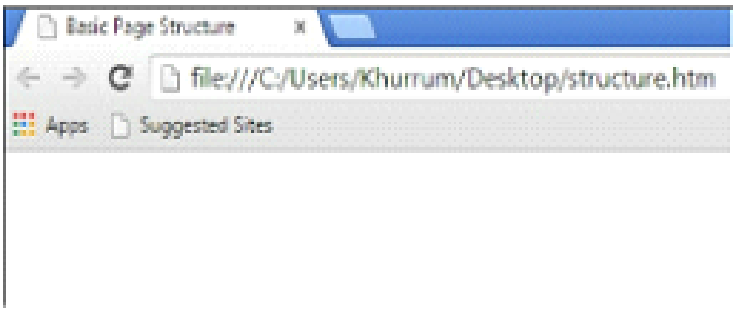
Theoretically, you can create an HTML 5 document **without** anything in the body, but you need to have a well-crafted Head and Body to index your page properly in the browser.
The following example contains all the basic elements you need to create an HTML 5 page.

HTML5 Example with basic Tags

```
<!DOCTYPE HTML>
    <html>
        <head>
            <meta charset="utf-8">
            <title>Basic Page Structure</title>
        </head>
        <body>
        </body>
    </html>
```

Notice that the head element also contains a Meta tag. Meta elements are used to specify other metadata, such as page author, description, keywords, last modified, etc.

If you save the above HTML 5 document and see it in the browser, it will display as:



Notice that the browser is showing nothing apart from the page title in the tab. This is because you have not done anything with the body so far.

The DOCTYPE

DOCTYPEs in older versions of HTML were longer because the HTML language was SGML based and therefore required a reference to a DTD.

HTML 5 authors would use simple syntax to specify DOCTYPE as follows:

```
<!DOCTYPE html>
```

Character Encoding

HTML 5 authors can use simple syntax to specify Character Encoding as follows −

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

The <script> tag

It's common practice to add a type attribute with a value of "text/javascript" to script elements as follows −

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

HTML 5 removes extra information required and you can use simply following syntax −

```
<script src="scriptfile.js"></script>
```

The Body Element in HTML5

After successfully setting up your Head, including metadata and page title, it is time to create some HTML5 markup in the body which will actually appear in the browser. The Body is like a big container that contains everything you want to see in the browser.
Take a look at Example

Body Element Example

```
<!DOCTYPE HTML>
    <html>
        <head>
            <meta charset="utf-8">
            <title>Basic Page Structure</title>
        </head>
        <body>
            <h1>Creating an HTML 5 Document</h1>
            <p>This is a body element</p>
        </body>
    </html>
```

You should see the following output in your browser.

That is fantastic. Now your browser is showing a heading at the top and some content below the heading as well. This is because you have added some HTML 5 markup in the Body element. Similarly, you can edit your HTML 5 document by adding any markup in the Body element.

Headings in HTML5

Headings and paragraphs are two of the most important and fundamental elements used in HTML 5.

Headings in HTML5

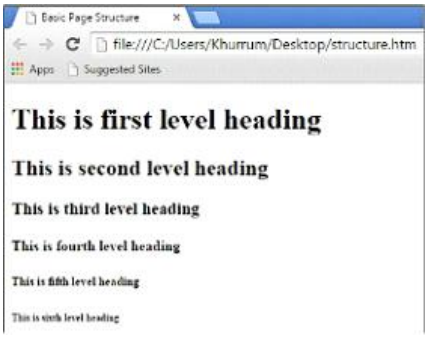
Denoted by the <h> tag, HTML 5 uses six levels of headings, from h1 to h6. In this regard, the first level heading <h1> is the largest whereas the last level heading <h6> is the smallest.

The basic purpose of headings is to break documents into sections. They help you create an organized structure, break up the text flow on the page and provide visual cues about the grouping of the content.

Headings in HTML 5 Example

```
<!DOCTYPE HTML>
    <html>
        <head>
            <meta charset="utf-8">
            <title>Basic Page Structure</title>
        </head>
        <body>
            <h1>This is first level heading</h1>
            <h2>This is second level heading</h2>
            <h3>This is third level heading</h3>
            <h4> This is fourth level heading</h4>
            <h5>This is fifth level heading</h5>
            <h6>This is sixth level heading</h6>
        </body>
    </html>
```

Different browsers will display headings differently. Google Chrome displays them as the following:



Paragraphs in HTML5

Anyone who is familiar with HTML 4 or another version of HTML must also be familiar with the paragraph tag. The paragraph is a text block that appears more than any other element on the web page.

You have to use ** ** opening and closing tags in order to group your content in paragraphs because web browsers do not recognize hard return inside HTML5 editors. It is very simple to create a paragraph in HTML5. You have to put an opening tag inside the body, write some text and close the paragraph with a tag.

Paragraphs in HTML 5 Example

```
<!DOCTYPE HTML>
    <html>
        <head>
            <meta charset="utf-8">
            <title>Basic Page Structure</title>
        </head>
        <body>
            <p>this is a paragraph</p>
            <p>this is another paragraph</p>
        </body>
    </html>
```

When you see the code in the browser, you will get something like this:

Thanks to the paragraph tag, we have two different paragraphs on our HTML 5 web page. In the same way, you can create as many paragraphs as you require or desire on your web page. Just remember to properly close your paragraphs with ; otherwise, it could cause a lot of rendering issues and your web page might not load properly.

- **HTML | Links**

What is a link?

It is a connection from one web resource to another. A link has two ends, An anchor and direction. The link starts at the "source" anchor and points to the "destination" anchor, which may be any Web resource such as an image, a video clip, a sound bite, a program, an HTML document or an element within an HTML document. This basically means that by using the 'a'

tag, you can link one element of the code to another element that may/may not be in your code.

HTML Link Syntax

Links are specified in HTML using the “a” tag.

```
<a href="url">Text link</a>
```

Syntax Explanation:

href : The href attribute is used to specify the destination address of the link used. "href" stands for Hypertext reference.
Text link : The text link is the visible part of the link. It is what the viewer clicks on.

```
<!DOCTYPE html>
    <html>
        <body>
            <h3>Example Of Adding a link</h3>
            <p>Click on the following link</p>
            <a href = "https://www.geeksforgeeks.org">GeeksforGeeks</a>
        </body>
    </html>
```

Internal Links

An internal link is a type of hyperlink whose target or destination is a resource, such as an image or document, on the same website or domain.

```
<!DOCTYPE html>
    <html>
        <body>
            <h3>Internal Link And External Link Example</h3>
            <!-- internal link-->
            <p><a href="html_contribute.asp/"> GeeksforGeeks Contribute
            </a> It is a link to the contribute page on GeeksforGeeks' website.</p>

            <!--external link-->
            <p><a href="https://www.geeksforgeeks.org">GeeksforGeeks
            </a> It is a link to the GeeksforGeeks website on the World Wide
            Web.</p>
        </body>
    </html>
```

Changing Link Colours in HTML

Different types of links appear in different formats such as:

1. An unvisited link appears underlined and blue in colour by default.
2. A visited link appears underlined and purple in colour by default.
3. An active link appears underlined and red in colour by default.

The appearances of links can be changed by using CSS.

```
<!DOCTYPE html>
        <html>
                <head>
                        <style>
                                a:link {
                                   color: red;
                                   background-color: transparent;
                                }
                                a:visited {
                                   color: green;
                                   background-color: transparent;
                                }
                                a:hover {
                                   color: blue;
                                   background-color: transparent;
                                }
                                a:active {
                                   color: yellow;
                                   background-color: transparent;
                                }
                        </style>
                </head>
                <body>
                        <p>Changing the default colors of links</p>
                        <p>Visited Link</p>
                        <a href="https://www.geeksforgeeks.org">GeeksforGeeks</a>
                        <p>Link</p>
                        <a href="https://facebook.com">facebook</a>
                        <p>hovering effect</p>
                        <a href="https://www.geeksforgeeks.org">GeeksforGeeks</a>

                </body>
        </html>
```

The Target Attribute in Links

The target attribute is used to specify the location where the linked document is opened. The various options that can be used in the target attribute are listed below in the table:

```
<!DOCTYPE html>
<html>
    <body>
        <h3>Various options available in the Target Attribute</h3>
        <p>If you set the target attribute to "_blank", the link will open in a new browser window or tab.</p>
        <a href="https://www.geeksforgeeks.org" target="_blank">GeeksforGeeks</a>
        <p>If you set the target attribute to "_self",the link will open in the same window or tab.</p>
        <a href="https://www.geeksforgeeks.org" target="_self">GeeksforGeeks</a>
        <p>If you set the target attribute to "_top",the link will open in the full body of the window.</p>
        <a href="https://www.geeksforgeeks.org" target="_top">GeeksforGeeks</a>
        <p>If you set the target attribute to "_parent",the link will open in the parent frame.</p>
        <a href="https://www.geeksforgeeks.org" target="_parent">GeeksforGeeks</a>
    </body>
</html>
```

Using Image as a Link in HTML

An image can be used to create a link to a specified URL. When the viewer clicks on the link, it redirects them to another page.

The code is

****
<img src=”file address (on device or on web)” alt=”_” style=”width:__ ; height:__ ; border:__”>

```
<!DOCTYPE html>
    <html>
        <body>
            <h3>Using Image as a link</h3>
            <p>Click on the image to visit GeeksforGeeks homepage.</p>
            <a href="https://www.geeksforgeeks.org">
            <img src="gfg_200X200.jpeg" alt="GeeksforGeeks"
            style="width:80px;height:80px;border:0">
            </a>
        </body>
    </html>
```

❖ HTML Tables

HTML Table is an arrangement of data in rows and columns, or possibly in a more complex structure. Tables are widely used in communication, research, and data analysis. Tables are useful for various tasks such as presenting text information and numerical data. It can be used to compare two or more items in the tabular form layout. Tables are used to create databases.

Defining Tables in HTML:

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

```
<!DOCTYPE html>
    <html>
        <body>
            <table>
                <tr>
                    <th>Book Name</th>
                    <th>Author Name</th>
                    <th>Genre</th>
                </tr>
                <tr>
                    <td>The Book Thief</td>
                    <td>Markus Zusak</td>
                    <td>Historical Fiction</td>
                </tr>
                <tr>
                    <td>The Cruel Prince</td>
                    <td>Holly Black</td>
                    <td>Fantasy</td>
                </tr>
                <tr>
                    <td>The Silent Patient</td>
                    <td> Alex Michaelides</td>
                    <td>Psychological Fiction</td>
                </tr>
            </table>
        </body>
    </html>
```

Book Name	Author Name	Genre
The Book Thief	Markus Zusak	Historical Fiction
The Cruel Prince	Holly Black	Fantasy
The Silent Patient	Alex Michaelides	Psychological Fiction

Adding a border to an HTML Table:

A border is set using the CSS border property. If you do not specify a border for the table, it will be displayed without borders.

```
table, th, td {
    border: 1px solid black;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
table,th,td {
border: 1px solid black;
}
</style>
</head>
<body>
<table style="width:100%">
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
</tr>
<tr>
<td>Priya</td>
<td>Sharma</td>
<td>24</td>
</tr>
<tr>
<td>Arun</td>
<td>Singh</td>
<td>32</td>
</tr>
<tr>
<td>Sam</td>
<td>Watson</td>
<td>41</td>
</tr>
</table>
</body>
</html>
```

Firstname	Lastname	Age
Priya	Sharma	24
Arun	Singh	32
Sam	Watson	41

Border Collapse Attribute:

```
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
}
```

Cell Padding:

```
th, td {
    padding: 15px;
}
```

Text Alignment:

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

Border Spacing:

```
table {
    border-spacing: 5px;
}
```

Table Caption:

```
<table style="width:100%">
<caption>DETAILS</caption>
```

Table Width and Background Color

```
table#t01 {
    width: 100%;
    background-color: #f2f2d1;
}
```

HTML Table Tags and attributes

Attributes and tags	Illustrations
Define Table	<table>..</table>
Add caption	<caption>..</caption>
Table row	<tr>..</tr>
Table header	<th>..</th>
Table Data in a cell	<td>..</td>
Cell spacing	<table cellspacing="">...</table>
Cell padding	<table cellpadding="">...</table>
Table border	<table border="">...</table>
Alignment	<table align=center/left/right>...</table>
colspan in table	<table colspan="">...</table>
rowspan in table	<table rowspan="">...</table>
Cell color	<table bgcolor="#$$$$$$">...</table>
No linebreaks	<table nowrap>...</table>

- **Forms**

An HTML form is used to collect user input. The user input is most often sent to a server for processing. The HTML <form> element is used to create an HTML form for user input:

Syntax:

```
<form>          .
        formelement    .
</form>
```

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

The <input> Element's (Previously Used)

Type	Description
<input type="text">	Displays a single-line text input field
<input type="radio">	Displays a radio button (for selecting one of many choices)
<input type="checkbox">	Displays a checkbox (for selecting zero or more of many choices)
<input type="submit">	Displays a submit button (for submitting the form)
<input type="button">	Displays a clickable button

New Introduced <input> Element's (some examples)

Sr.No.	Type & Description
1	datetime A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC.
2	datetime-local A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601, with no time zone information.
3	date A date (year, month, day) encoded according to ISO 8601.
4	month A date consisting of a year and a month encoded according to ISO 8601.
5	week A date consisting of a year and a week number encoded according to ISO 8601.
6	time A time (hour, minute, seconds, fractional seconds) encoded according to ISO 8601.
7	number It accepts only numerical value. The step attribute specifies the precision, defaulting to 1.
8	range The range type is used for input fields that should contain a value from a range of numbers.
9	email It accepts only email value. This type is used for input fields that should contain an e-mail address. If you try to submit a simple text, it forces to enter only email address in email@example.com format.
10	url It accepts only URL value. This type is used for input fields that should

contain a URL address. If you try to submit a simple text, it forces to enter only URL address either in http://www.example.com format or in http://example.com format.

Text Fields

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

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

The <label> Element

The <label> tag defines a label for many form elements.The <label> element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.The <label> element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the <label> element, it toggles the radio button/checkbox.The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together.

Radio Buttons

The <input type="radio"> defines a radio button.Radio buttons let a user select ONE of a limited number of choices

```
<p>ChooseYourFavoriteWebLanguage:</p>

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

Checkboxes

The <input type="checkbox"> defines a checkbox.
Checkboxes let a user select ZERO or MORE options of a limited number of choices.

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

The Submit Button

The <input type="submit"> defines a button for submitting the form data to a form-handler.
The form-handler is typically a file on the server with a script for processing input data.
The form-handler is specified in the form's action attribute.

```
<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>
```

The Name Attribute for <input>

Notice that each input field must have a name attribute to be submitted.
If the name attribute is omitted, the value of the input field will not be sent at all.

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

HTML Form Attributes

Attribute	Description
accept-charset	Specifies the character encodings used for form submission
action	Specifies where to send the form-data when a form is submitted
autocomplete	Specifies whether a form should have autocomplete on or off

enctype	Specifies how the form-data should be encoded when submitting it to the server (only for method="post")
method	Specifies the HTTP method to use when sending form-data
name	Specifies the name of the form
novalidate	Specifies that the form should not be validated when submitted
rel	Specifies the relationship between a linked resource and the current document
target	Specifies where to display the response that is received after submitting the form

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
_blank	The response is displayed in a new window or tab
_self	The response is displayed in the current window
_parent	The response is displayed in the parent frame
_top	The response is displayed in the full body of the window
framename	The response is displayed in a named iframe

Input Restrictions

Attribute	Description
checked	Specifies that an input field should be pre-selected when the page loads (for type="checkbox" or type="radio")
disabled	Specifies that an input field should be disabled
max	Specifies the maximum value for an input field
maxlength	Specifies the maximum number of character for an input field
min	Specifies the minimum value for an input field
pattern	Specifies a regular expression to check the input value against
readonly	Specifies that an input field is read only (cannot be changed)
required	Specifies that an input field is required (must be filled out)
size	Specifies the width (in characters) of an input field
step	Specifies the legal number intervals for an input field
value	Specifies the default value for an input field

- **HTML Script Tag**

HTML script tag is used to specify client-side script such as JavaScript. It facilitate you to place a script within your HTML document.

JavaScript is used for image manipulation, form validation, and dynamic content.

The syntax of script tag is given below:

```
<script>
  //code to be executed
</script>
```

Attributes of HTML script tag

Attribute	Description
src	It specifies the URL of an external script file.
type	It specifies the media type of the script.
async	It is a boolean value which specifies that the script is executed asynchronously.
defer	It is a boolean value which is used to indicate that script is executed after document has been parsed.

Usage of script tag

There can be two usage of HTML script tag:

1. to embed script code
2. to link script file

1. Embed script code

The script tag can be used within <body> or <head> tag to embed the scripting code. Let's see the example to have script tag within HTML body.

```
<script type="text/javascript">
document.write("JavaScript is a simple language for javatpoint learners")
</script>
```

Let's see the example to have script tag within HTML head tag.

```
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
</script>
```

2. **Link script file**

   The script tag can be used to link external script file by src attribute. It must be used within the <head> tag only.

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

❖ **Explain API's Available in HTML5**

API stands for **Application Programming Interface.** An API is a set of pre-built programs that can be used with the help of JavaScript. APIs are used to implement already written code to fulfill the needs of the project you are working on.

1. **HTML Geolocation API:**

The Geolocation API is used to get the current location of the user or the page visitor. It will show the user's location only if the user allows it to do so, as it compromises the security and privacy of the user.

Syntax:

var loc = navigator.geolocation;

Methods available in Geolocation API:

1. **getCurrentPosition() Method:** The getCurrentPosition() method returns an object with properties such as **latitude, longitude, accuracy, altitude** etc.
2. **watchPosition() Method:** This method will return the current position of the user as well as the updated location when the position of the user changes or the user travels from one location to another location.
3. **clearWatch() Method:** This method will stop the watchPosition() method to not tracing the user anymore.

Example

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content=
        "width=device-width, initial-scale=1.0">
    <title>HTML Geolocation API</title>
    <style>
```

```
            #container {
                display: flex;
                flex-direction: column;
                justify-content: center;
                align-items: center;
            }
    </style>
</head>

<body>
    <div id="container">
        <h1>Hey Geek,</h1>
        <h3>Welcome to GeeksforGeeks!</h3>
        <button type="button" onclick="geoLoc()">
            Get Location
        </button>
    </div>

    <script>
        var container = document.getElementById("container");
        var pos = navigator.geolocation;

        function geoLoc() {
            if (pos)
                pos.getCurrentPosition(getLoc);
            else
                container.innerHTML = "Your browser does "
                    + "not support the Geolocation API.";
        }

        function getLoc(loc) {
            container.innerHTML = "Latitude: "
                + loc.coords.latitude +
                "<br>Longitude: " +
                loc.coords.longitude;
        }
    </script>
</body>

</html>
```

2. <u>HTML Drag and Drop API:</u>

Drag and Drop is a common feature nowadays, where you can drag an item from one place and drop it in another.

Syntax:

To use drag and drop first you have to make the element draggable as shown below:

```
<div draggable="true">
   //content of the element
</div>
```

Example

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content=
            "width=device-width, initial-scale=1.0">
    <title>HTML Geolocation API</title>

    <style>
            #container {
                    display: flex;
                    flex-direction: column;
                    justify-content: center;
                    align-items: center;
            }

            #drop {
                    height: 105px;
                    width: 105px;
                    border: 2px solid black;
            }
    </style>
</head>

<body>
    <div id="container">
            <h1>Hey Geek,</h1>
            <h3>Welcome to GeeksforGeeks!</h3>


<p>Drag image into the container</p>


            <div id="drop" ondrop="dropItem(event)"
                    ondragover="droppable(event)"></div>

            <img src=
```

```
"https://media.geeksforgeeks.org/wp-content/uploads/20220428080551/gfglogo-
200x200.png"
                alt="GeeksforGeeks" id="image"
                draggable="true" ondragstart="dragItem(event)"
                height="100px" width="100px">
    </div>

    <script>
        function droppable(e) {
            e.preventDefault();
        }

        function dragItem(e) {
            e.dataTransfer.setData("text", e.target.id);
        }

        function dropItem(e) {
            e.preventDefault();
            var content = e.dataTransfer.getData("text");
            e.target.appendChild(document.getElementById(content));
        }
    </script>
</body>

</html>
```

3. HTML Web Storage API:

HTML web storage API is used to store the data on the web browser. Early, the data was stored in the form of cookies that can store a small amount of data and can-not transferred further to the server. But, HTML5 introduces us to the Web Storage API that can store large data as compared to cookies and can be transferred to the server. Using this API for storing data is more secure than using cookies.

Web storage API provides us with two objects to work with:

1. **window.sessionStorage:** This object temporarily stores the data on the web browser such that if the browser is refreshed or closed the data stored will be lost.
2. **window.localStorage:** localStorage permanently stores the data on the browser with no expiration such that will not be lost even if the browser is refreshed.

Example

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

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content=
        "width=device-width, initial-scale=1.0">
    <title>HTML Web Storage API</title>

    <style>
        #container {
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
        }

        #btnDiv {
            width: 20vw;
            display: flex;
            flex-direction: row;
            align-items: center;
            justify-content: space-around;
        }

        .btn {
            cursor: pointer;
        }
    </style>
</head>

<body>
    <div id="container">
        <h1>Hey Geek,</h1>
        <h2 id="heading"></h2>
        <h3 id="desc"></h3>
        <div id="btnDiv">
            <button class="btn" onclick="getContent()">
                Get stored Data
            </button>

            <button class="btn" onclick="remContent()">
                Remove stored Data
            </button>
        </div>
    </div>

    <script>
```

```
        var head = document.getElementById('heading');
        var desc = document.getElementById("desc");
        function getContent() {
            if (typeof (Storage) !== "undefined") {

                // setItem() will set store the passed attribute
                // and value in localStorage
                localStorage.setItem('heading', 'Welcome to GeeksforGeek!');
                localStorage.setItem('description',
                    'A computer science portal for Geeks.');

                // This is the way of accessing the items
                // stored in the storage
                head.innerText = localStorage.heading;
                desc.innerText = localStorage.description;
            }
            else {
                head.innerText =
                    "Your browser does not support web storage API.";
            }
        }

        function remContent() {

            // removeItem() will remove the passed attribute
            // and value from localStorage.
            localStorage.removeItem('heading');
            localStorage.removeItem('description');
            head.innerText = localStorage.heading;
            desc.innerText = localStorage.description;
        }
    </script>
</body>

</html>
```

4. **HTML Web Worker API:**

   Generally, when the JavaScript is uploading for the page, the page got stuck until uploading gets finished. The Web worker API helps us to upload the JavaScript without affecting the performance of the page. It helps to run JavaScript run in the background independent of other scripts.

Example:

```
<!DOCTYPE html>
```

```
<html>

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content=
        "width=device-width, initial-scale=1.0">

    <title>HTML Web Storage API</title>

    <style>
        #container {
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
        }

        #btnDiv {
            width: 20vw;
            display: flex;
            flex-direction: row;
            align-items: center;
            justify-content: space-around;
        }

        .btn {
            cursor: pointer;
        }
    </style>
</head>

<body>
    <div id="container">
        <h1>Hey Geek,</h1>
        <h2 id="heading"></h2>
        <div id="btnDiv">
            <button class="btn" onclick="getJS()">
                Start executing JS
            </button>
            <button class="btn" onclick="remJS()">
                Stop executing JS
            </button>
        </div>
    </div>
```

```
    <script>
        var myWorker;
        var head = document.getElementById('heading');
        function getJS() {

            // Below condition will checks for
            // the browser support.
            if (typeof (Worker) !== "undefined") {

                // The condition below will checks for
                // existence of the worker
                if (typeof (myWorker) == "undefined") {
                    myWorker = new Worker("myScript.js");
                    // Above line Will create a worker that will
                    // execute the code of myscript.js file
                }

                // onmessage event triggers a function
                // with the data stored in external js file
                myWorker.onmessage = function (props) {
                    head.innerText = props.data;
                }
            }
            else {
                head.innerText =
                    "Your browser does not support web storage API.";
            }
        }

        function remJS() {

            // Terminate() will terminate the current worker
            myWorker.terminate();
            myWorker = undefined;
        }
    </script>
</body>

</html>
```

- **The XHTML syntax**

XHTML syntax is very similar to HTML syntax and all the valid HTML elements are also valid in XHTML. But XHTML is case sensitive so you have to pay a bit extra attention while writing an XHTML document to make your HTML document compliant to XHTML.

You must remember the following important points while writing a new XHTML document or converting existing HTML document into XHTML document:

1. All documents must have a DOCTYPE.
2. All tags must be in lower case.
3. All documents must be properly formed.
4. All tags must be closed.
5. All attributes must be added properly.
6. The name attribute has changed.
7. Attributes cannot be shortened.
8. All tags must be properly nested.

- ***User Interaction***

These are the various features that allow authors to enable users to edit documents and parts of documents interactively.

The hidden attribute

All HTML elements may have the hidden content attribute set. The hidden attribute is a boolean attribute. When specified on an element, it indicates that the element is not yet, or is no longer, relevant. User agents should not render elements that have the hidden attribute specified.

In the following example, the attribute is used to hide the Web game's main screen until the user logs in:

```
<h1>The Example Game</h1>
<section id="login">
 <h2>Login</h2>
 <form>
  ...
  <!-- calls login() once the user's credentials have been checked -->
 </form>
 <script>
  function login() {
    // switch screens
    document.getElementById('login').hidden = true;
    document.getElementById('game').hidden = false;
  }
 </script>
</section>
<section id="game" hidden>
```

...
</section>

The hidden attribute must not be used to hide content that could legitimately be shown in another presentation. For example, it is incorrect to use hidden to hide panels in a tabbed dialog, because the tabbed interface is merely a kind of overflow presentation — showing all the form controls in one big page with a scrollbar would be equivalent, and no less correct.

Elements in a section hidden by the hidden attribute are still active, e.g. scripts and form controls in such sections still render execute and submit respectively. Only their presentation to the user changes.

The hidden DOM attribute must reflect the content attribute of the same name.

Activation

element . click(): Acts as if the element was clicked.

Each element has a *click in progress* flag, initially set to false.

The click() method must run these steps:

1. If the element's *click in progress* flag is set to true, then abort these steps.
2. Set the *click in progress* flag on the element to true.
3. If the element has a defined activation behavior, run synthetic click activation steps on the element. Otherwise, fire a click event at the element.
4. Set the *click in progress* flag on the element to false.

Scrolling elements into view

element . *scrollIntoView([* top *])*

Scrolls the element into view. If the *top* argument is true, then the element will be scrolled to the top of the viewport, otherwise it'll be scrolled to the bottom. The default is the top.

The scrollIntoView([*top*]) method, when called, must cause the element on which the method was called to have the attention of the user called to it.

In a speech browser, this could happen by having the current playback position move to the start of the given element.

In visual user agents, if the argument is present and has the value false, the user agent should scroll the element into view such that both the bottom and the top of the element are in the viewport, with the bottom of the element aligned with the bottom of the viewport. If it isn't possible to show the entire element in that way, or if the argument is omitted or is true, then the user agent should instead align the top of the element with the top of the viewport. If the entire scrollable part of the content is visible all at once (e.g. if a page is shorter than the viewport),

then the user agent should not scroll anything. Visual user agents should further scroll horizontally as necessary to bring the element to the attention of the user.

Non-visual user agents may ignore the argument, or may treat it in some media-specific manner most useful to the user.

Focus

When an element is focused, key events received by the document must be targeted at that element. There may be no element focused; when no element is focused, key events received by the document must be targeted at the body element.

User agents may track focus for each browsing context or Document individually, or may support only one focused element per top-level browsing context — user agents should follow platform conventions in this regard.

Which elements within a top-level browsing context currently have focus must be independent of whether or not the top-level browsing context itself has the system focus.

When an element is focused, the element matches the CSS :focus pseudo-class.

1. Sequential focus navigation

The tabindex content attribute specifies whether the element is focusable, whether it can be reached using sequential focus navigation, and the relative order of the element for the purposes of sequential focus navigation. The name "tab index" comes from the common use of the "tab" key to navigate through the focusable elements. The term "tabbing" refers to moving forward through the focusable elements that can be reached using sequential focus navigation.

The tabindex attribute, if specified, must have a value that is a valid integer.

If the attribute is specified, it must be parsed using the rules for parsing integers. The attribute's values have the following meanings:

If the attribute is omitted or parsing the value returns an error

> The user agent should follow platform conventions to determine if the element is to be focusable and, if so, whether the element can be reached using sequential focus navigation, and if so, what its relative order should be.

If the value is a negative integer

> The user agent must allow the element to be focused, but should not allow the element to be reached using sequential focus navigation.

If the value is a zero

> The user agent must allow the element to be focused, should allow the element to be reached using sequential focus navigation, and should follow platform conventions to determine the element's relative order.

If the value is greater than zero

> The user agent must allow the element to be focused, should allow the element to be reached using sequential focus navigation, and should place the element in the sequential focus navigation order so that it is:

1. before any focusable element whose tabindex attribute has been omitted or whose value, when parsed, returns an error,
2. before any focusable element whose tabindex attribute has a value equal to or less than zero,
3. after any element whose tabindex attribute has a value greater than zero but less than the value of the tabindex attribute on the element,
4. after any element whose tabindex attribute has a value equal to the value of the tabindex attribute on the element but that is earlier in the document in tree order than the element,
5. before any element whose tabindex attribute has a value equal to the value of the tabindex attribute on the element but that is later in the document in tree order than the element, and
6. before any element whose tabindex attribute has a value greater than the value of the tabindex attribute on the element.

An element is specially focusable if the tabindex attribute's definition above defines the element to be focusable.

An element that is specially focusable but does not otherwise have an activation behavior defined has an activation behavior that does nothing.

This means that an element that is only focusable because of its tabindex attribute will fire a click event in response to a non-mouse activation (e.g. hitting the "enter" key while the element is focused).

An element is focusable if the user agent's default behavior allows it to be focusable or if the element is specially focusable, but only if the element is being rendered.

2. Focus management

The focusing steps are as follows:

1. If focusing the element will remove the focus from another element, then run the unfocusing steps for that element.
2. Make the element the currently focused element in its top-level browsing context.

3. Some elements, most notably area, can correspond to more than one distinct focusable area. If a particular area was indicated when the element was focused, then that is the area that must get focus; otherwise, e.g. when using the focus() method, the first such region in tree order is the one that must be focused.
4. Fire a simple event called focus at the element.

User agents must run the focusing steps for an element whenever the user moves the focus to a focusable element.

The unfocusing steps are as follows:

1. If the element is an input element, and the change event applies to the element, and the element does not have a defined activation behavior, and the user has changed the element's value or its list of selected files while the control was focused without committing that change, then fire a simple event that bubbles called change at the element, then broadcast formchange events at the element's form owner.
2. Unfocus the element.
3. Fire a simple event called blur at the element.

When an element that is focused stops being a focusable element, or stops being focused without another element being explicitly focused in its stead, the user agent should run the focusing steps for the body element, if there is one; if there is not, then the user agent should run the unfocusing steps for the affected element only.

For example, this might happen because the element is removed from its Document, or has a hidden attribute added. It would also happen to an input element when the element gets disabled.

3. Document-level focus APIs

document . activeElement: Returns the currently focused element.

document . hasFocus(): Returns true if the document has focus; otherwise, returns false.

window . focus(): Focuses the window. Use of this method is discouraged. Allow the user to control window focus instead.

window . blur(): Unfocuses the window. Use of this method is discouraged. Allow the user to control window focus instead.

The activeElement attribute on DocumentHTML objects must return the element in the document that is focused. If no element in the Document is focused, this must return the body element.

The hasFocus() method on DocumentHTML objects must return true if the document's browsing context is focused, and all its ancestor browsing contexts are also focused, and the top-level browsing context has the system focus.

The focus() method on the Window object, when invoked, provides a hint to the user agent that the script believes the user might be interested in the contents of the browsing context of the Window object on which the method was invoked.

User agents are encouraged to have this focus() method trigger some kind of notification.

The blur() method on the Window object, when invoked, provides a hint to the user agent that the script believes the user probably is not currently interested in the contents of the browsing context of the Window object on which the method was invoked, but that the contents might become interesting again in the future.

User agents are encouraged to ignore calls to this blur() method entirely.

Historically the focus() and blur() methods actually affected the system focus, but hostile sites widely abuse this behavior to the user's detriment.

4. **Element-level focus APIs**

element . focus(): Focuses the element.

element . blur(): Unfocuses the element. Use of this method is discouraged. Focus another element instead.

The focus() method, when invoked, must run the following algorithm:

1. If the element is marked as locked for focus, then abort these steps.
2. If the element is not focusable, then abort these steps.
3. Mark the element as locked for focus.
4. If the element is not already focused, run the focusing steps for the element.
5. Unmark the element as locked for focus.

The blur() method, when invoked, should run the focusing steps for the body element, if there is one; if there is not, then it should run the unfocusing steps for the element on which the method was called instead. User agents may selectively or uniformly ignore calls to this method for usability reasons.

The accesskey attribute

All HTML elements may have the accesskey content attribute set. The accesskey attribute's value is used by the user agent as a guide for creating a keyboard shortcut that activates or focuses the element.

If specified, the value must be an ordered set of unique space-separated tokens, each of which must be exactly one Unicode code point in length.

An element's assigned access key is a key combination derived from the element's accesskey content attribute as follows:

1. If the element has no accesskey attribute, then skip to the fallback step below.
2. Otherwise, the user agent must must split the attribute's value on spaces, and let *keys* be the resulting tokens.
3. For each value in *keys* in turn, in the order the tokens appeared in the attribute's value, run the following substeps:
    a. If the value is not a string exactly one Unicode code point in length, then skip the remainder of these steps for this value.
    b. If the value does not correspond to a key on the system's keyboard, then skip the remainder of these steps for this value.
    c. If the user agent can find a combination of modifier keys that, with the key that corresponds to the value given in the attribute, can be used as a shortcut key, then the user agent may assign that combination of keys as the element's assigned access key and abort these steps.
4. Fallback: Optionally, the user agent may assign a key combination of its chosing as the element's assigned access key and then abort these steps.
5. If this step is reached, the element has no assigned access key.