HTML NOTES

HTML stands for **Hypertext Markup Language**. HTML is the basic building block of World Wide Web. **Hypertext** is text displayed on a computer or other electronic device with references to other text that the user can instantly access, by a **mouse click** or **key press**. Apart from **text**, hypertext may have **tables**, **lists**, **forms**, **images**, and other presentational elements. **Markup languages** use sets of **markup tags** to characterize text elements within a document, which gives instructions to the web browsers on how the document should appear. **HTML** was originally developed by **Tim Berners-Lee** in **1990**. He is also known as the father of the web. In 1996, the World Wide Web Consortium (W3C) became the authority to maintain the HTML specifications. **HTML** also became an international standard (ISO) in 2000. **HTML5** is the latest version of HTML. **HTML5** provides a faster and more robust approach to web development.

WHAT YOU CAN DO WITH HTML

There are lot more things you can do with HTML.

- You can publish documents online with text, images, lists, tables, etc.
- You can access web resources such as images, videos or other HTML document via hyperlinks.
- You can create forms to collect user inputs like name, e-mail address, comments, etc.
- You can include images, videos, sound clips, flash movies, applications and other HTML documents directly inside an HTML document.
- You can create offline version of your website that work without internet.
- You can store data in the user's web browser and access later on.
- You can find the current location of your website's visitor.

The list does not end here, there are many other interesting things that you can do with HTML. You will learn about all of them in detail in upcoming chapters.

Note: HTML as described earlier is a markup language not a programming language, like Java, Ruby, PHP, etc. You need a web browser to view the HTML pages. The web browsers do not display the HTML tags, but uses the tags to interpret the content of the web pages.

HTML GET STARTED

An HTML file is simply a text file saved with an .html or .htm extension.

Getting Started

In this tutorial you will learn how easy it is to create an HTML document or a web page. To begin coding HTML, you need only two stuff: a simple-text editor and a web browser. Well, let's get started with creating your first HTML page.

Creating Your First HTML Document

Let's walk through the following steps.

Step 1: Creating the HTML file

Open up your computer's plain text editor and create a new file.

Tip: I suggest you to use an HTML Editor; don't use Word or WordPad!

Step 2: Type some HTML code

Start with an empty window and type the following code:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Getting Started with HTML </title>
</head>
<body>
    <p>Hello World!<p>
</body>
</html>
```

Step 3: Saving the file

Now save the file on your desktop as "**myfirstpage.html** ".

Note: It is important that the extension **`.html`** is specified — some text editors, such as Notepad, will automatically save it as **`.txt`** otherwise.

To open the file in a browser. Navigate to your file then double click on it. It will open in your default Web browser. If it does not, open your browser and drag the file to it.

Explanation of code

You might think what that code was all about. Well, let's find out.

- The first line **`<!DOCTYPE html>`** is the document type declaration. It instructs the web browser that this document is an HTML5 document. It is case-insensitive.
- The **`<head>`** element is a container for the tags that provides information about the document, for example, **`<title>`** tag defines the title of the document.
- The **`<body>`** element contains the document's actual content (paragraphs, links, images, tables, and so on) that is rendered in the web browser and displayed to the user.

For now, just focus on the basic structure of the HTML document.

Note: A **DOCTYPE** declaration appears at the top of a web page before all other elements; however, the doctype declaration itself is not an HTML tag. Every HTML document requires a document type declaration to ensure that your pages are displayed correctly.

Tip: The **<html>, <head>,** and **<body>** tags make up the basic skeleton of every web page. Content inside the **<head>** and **</head>** are invisible to users with one exception: the text between **<title>** and **</title>** tags which appears as the title on a browser tab.

HTML Tags and Elements

HTML is written in the form of **HTML elements** consisting of markup tags. These markup tags are the fundamental characteristic of HTML. Every markup tag is composed of a keyword, surrounded by angle brackets, such as `<html>`, `<head>`, `<body>`, `<title>`, `<p>`, and so on. HTML tags normally come in pairs like `<html>` and `</html>`. The first tag in a pair is often called the opening tag (or start tag), and the second tag is called the closing tag (or end tag). An opening tag and a closing tag are identical, except a slash (`/`) after the opening angle bracket of the closing tag, to tell the browser that the command has been completed. In between the **start** and **end tags** you can place appropriate contents. E.g., a paragraph, which is represented by the **p** element, would be written as:

Example
Try this code »

```
<p>This is a paragraph.</p> <!-- Paragraph with nested element -->
<p>
    This is <b>another</b> paragraph.
</p>
```

You will learn about the various HTML elements in upcoming chapter.

HTML Element Syntax

An HTML element is an individual component of an HTML document. It represents semantics, or meaning. For example, the **title** element represents the title of the document. Most HTML elements are written with a *start tag* (or opening tag) and an **end tag** (or **closing tag**), with content in between. Elements can also contain attributes that defines its additional properties. For example, a paragraph, which is represented by the `p` element, would be written as:

We will learn about the HTML attributes in the next chapter.

Note: All elements don't require the end tag or closing tag to be present. These are referred as *empty elements*, *self-closing elements* or *void elements*.

HTML Tags Vs Elements

Technically, an HTML element is the collection of start tag, its attributes, an end tag and everything in between. On the other hand an HTML tag (either opening or closing) is used to mark the start or end of an element, as you can see in the above illustration. However, in common usage the terms HTML element and HTML tag are interchangeable i.e. a tag is an element is a tag. For simplicity's sake of this website, the terms "tag" and "element" are used to mean the same thing — as it will define something on your web page.

Case Insensitivity in HTML Tags and Attributes

In HTML, tag and attribute names are not case-sensitive (but most attribute values are case-sensitive). It means the tag `<P>`, and the tag `<p>` defines the same thing in HTML which is a paragraph. But in XHTML they are case-sensitive and the tag `<P>` is different from the tag `<p>`.

Example
Try this code »

```
<p>This is a paragraph.</p>
<P>This is also a valid paragraph.</P>
```

Tip: We recommend using lowercase for tag and attributing names in HTML, since by doing this you can make your document more compliant for future upgrades.

Empty HTML Elements

Empty elements (also called self-closing or void elements) are not container tags — that means, you cannot write `<hr>some content</hr>` or `<br>some content</br>`. A typical example of an empty element, is the `<br>` element, which represents a line break. Some other common empty elements are `<img>`, `<input>`, `<link>`, `<meta>`, `<hr>`, etc.

Example
Try this code »

```
<p>This paragraph contains <br> a line break.</p>
<img src="images/sky.jpg" alt="Cloudy Sky">
<input type="text" name="username">
```

Note: In HTML, a self-closing element is written simply as <br>. In XHTML, a self-closing element requires a space and a trailing slash, such as <br />.

Nesting HTML Elements

Most HTML elements can contain any number of further elements (except empty elements), which are, in turn, made up of tags, attributes, and content or other elements. The following example shows some elements nested inside the <p> element.

Example
Try this code »

```
<p>Here is some <b>bold</b> text.</p>
<p>Here is some <em>emphasized</em> text.</p>
<p>Here is some <mark>highlighted</mark> text.</p>
```

Tip: Placing one element inside another is called nesting. A nested element, also called a child element, can be a parent element too if other elements are nested within it.

HTML tags should be nested in correct order. They must be closed in the inverse order of how they are defined, that means the last tag opened must be closed first.

Example
Try this code »

```
<p><strong>These tags are nested properly.</strong></p>
<p><strong>These tags are not nested properly.</p></strong>
```

Writing Comments in HTML

Comments are usually added with the purpose of making the source code easier to understand. It may help other developer (or you in the future when you edit the source code) to understand what you were trying to do with the HTML. Comments are not displayed in the browser. An HTML comment begins with <!--, and ends with -->, as shown in the example below:

Example
Try this code »

```
<!-- This is an HTML comment -->
<!-- This is a multi-line HTML comment
     that spans across more than one line -->
<p>This is a normal piece of text.</p>
```

You can also comment out part of your HTML code for debugging purpose, as shown here:

Example
Try this code »

```
<!-- Hiding this image for testing
<img src="images/smiley.png" alt="Smiley">
-->
```

HTML Elements Types

Elements can be placed in two groups: ***block level*** and ***inline level*** elements. The former makes up the document's structure, while the latter dress up the contents of a block. Also, a block element occupies 100% of the available width and it is rendered with a line break before and after. Whereas, an inline element will take up only as much space as it needs.

The most commonly used block-level elements are `<div>`, `<p>`, `<h1>` through `<h6>`, `<form>`, `<ol>`, `<ul>`, `<li>`, and so on. Whereas, the commonly used inline-level elements are `<img>`, `<a>`, `<span>`, `<strong>`, `<b>`, `<em>`, `<i>`, `<code>`, `<input>`, `<button>`, etc.

You will learn about these elements in detail in upcoming chapters.

Note: The block-level elements should not be placed within inline-level elements. For example, the `<p>` element should not be placed inside the `<b>` element.

HTML Attributes. *What are Attributes*

Attributes define extra characteristics or properties of the element such as width and height of an image. Attributes are always specified in the start tag (or opening tag) and usually consists of name/value pairs like `name="value"`. Attribute values should always be enclosed in quotation marks. Also, some attributes are required for certain elements. For instance, an **`<img>`** tag must contain a **`src`** and **`alt`** attributes. Let's take a look at some examples of the attributes usages:

Example
Try this code »

```
<img src="images/smiley.png" width="30" height="30" alt="Smiley">
<a href="https://www.google.com/" title="Search Engine">Google</a>
<abbr title="Hyper Text Markup Language">HTML</abbr>
<input type="text" value="John Doe">
```

In the above example **src** inside the **<img>** tag is an attribute and image path provided is its value. Also, **href** inside the **<a>** tag is an attribute and the link provided is its value, etc.

Tip: Both **single** and **double quotes** can be used to quote attribute values. However, double quotes are most common. In situations where the attribute value itself contains double quotes it is necessary to wrap the value in single quotes, e.g., `value='John "Williams" Jr.'`

There are several attributes in HTML5 that do not consist of name/value pairs but consists of just name. Such attributes are called Boolean attributes. Examples of some commonly used Boolean attributes are **checked**, **disabled**, **readonly**, **required**, etc.

Example
Try this code »

```
<input type="email" required>
<input type="submit" value="Submit" disabled>
<input type="checkbox" checked>
<input type="text" value="Read only text" readonly>
```

You will learn about all these elements in detail in upcoming chapters.

Note: Attribute values are case-insensitive, except, like the **id** and **class** attributes. However, World Wide Web Consortium (W3C) recommends lowercase for attributes values in their specification.

General Purpose Attributes

There are some attributes, such as `id`, `title`, `class`, `style`, etc. that you can use on the majority of HTML elements. The following section describes their usages.

The **id** Attribute

The `id` attribute is used to give a unique name or identifier to an element within a document. This makes it easier to select the element using CSS or JavaScript.

Example
Try this code »

```
<input type="text" id="firstName">
<div id="container">Some content</div>
<p id="infoText">This is a paragraph.</p>
```

Note: The `id` of an element must be unique within a single document. No two elements in the same document can be named with the same `id`, and each element can have only one `id`.

The **class** Attribute

Like the **id** attribute, the **class** attribute is also used to identify elements. But unlike **id**, the **class** attribute does not have to be unique in a document. This means you can apply the same class to multiple elements in a document, as shown in the following example:

Example
Try this code »

```
<input type="text" class="highlight">
<div class="box highlight">Some content</div>
<p class="highlight">This is a paragraph.</p>
```

Tip: Since a **class** can be applied to multiple elements, therefore any style rules that are written to that **class** will be applied to all the elements having that `class`.

The **title** Attribute

The `title` attribute to is used to provide advisory text about an element or its content. Try out the following example to understand how this actually works.

Example
Try this code »

```
<abbr title="World Wide Web Consortium">W3C</abbr>
<a href="images/kites.jpg" title="Click to view a larger image">
    <img src="images/kites-thumb.jpg" alt="kites">
</a>
```

Tip: The value of the `title` attribute (i.e. title text) is displayed as a tooltip by the web browsers when the user place mouse cursor over the element.

The **style** Attribute

The `style` attribute allows you to specify CSS styling rules such as color, font, border, etc. directly within the element. Let's check out an example to see how it works:

Example
Try this code »

```
<p style="color: blue;">This is a paragraph.</p>
<img src="images/sky.jpg" style="width: 300px;" alt="Cloudy Sky">
<div style="border: 1px solid red;">Some content</div>
```

You will learn more about styling HTML elements in HTML styles chapter.

The attributes we've discussed above are also called global attributes. For more global attributes please check out the HTML5 global attributes reference.

A complete list of attributes for each HTML element is listed inside HTML5 tag reference.

HTML Headings

Organizing Content with Headings

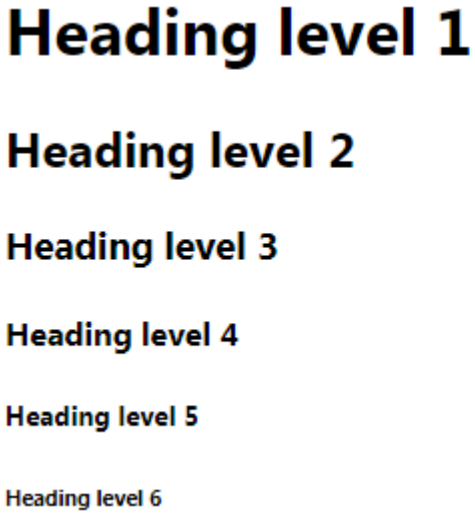
Headings help in defining the hierarchy and the structure of the web page content. HTML offers **six levels** of heading tags, **<h1>** through **<h6>**; the lower the heading level number, the greater its importance — thus **<h1>** tag defines the most important heading, whereas the **<h6>** tag defines the least important heading in the document.

By default, browsers display headings in larger and bolder font than normal text.
Also, `<h1>` headings are displayed in largest font, whereas `<h6>` headings are displayed in smallest font.

Example
Try this code »

```
<h1>Heading level 1</h1>
<h2>Heading level 2</h2>
<h3>Heading level 3</h3>
<h4>Heading level 4</h4>
<h5>Heading level 5</h5>
<h6>Heading level 6</h6>
```

— The output of the above example will look something like this:



Note: Each time you place a heading tag on a web page, the web browser built-in style sheets automatically create some empty space (called margin) before and after each heading. You can use the CSS **margin** property to override the browser's default style sheet.

Tip: You can easily customize the appearance of HTML heading tags such as their font size, boldness, typeface, etc. using the CSS **font** properties.

Importance of Headings

- HTML headings provide valuable information by highlighting important topics and the structure of the document, so optimize them carefully to improve user engagement.
- Don't use headings to make your text look BIG or bold. Use them only for highlighting the heading of your document and to show the document structure.
- Since search engines, such as Google, use headings to index the structure and content of the web pages so use them very wisely in your webpage.
- Use the `<h1>` headings as main headings of your web page, followed by the `<h2>` headings, then the less important `<h3>` headings, and so on.

Tip: Use the `<h1>` tag to mark the most important heading which is usually at the top of the page. An HTML document generally should have exactly one `<h1>` heading, followed by the lower-level headings such as `<h2>`, `<h3>`, `<h4>`, and so on.

HTML Paragraphs. *Creating Paragraphs*

Paragraph element is used to publish text on the web pages. Paragraphs are defined with the `<p>` tag. Paragraph tag is a very basic and typically the first tag you will need to publish your text on the web pages. Here's an example:

Example
Try this code »

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

Note: Browsers built-in style sheets automatically create some space above and below the content of a paragraph (called margin), but you can override it using CSS.

Closing a Paragraph Element

In HTML 4 and earlier versions, it was enough to initiate a new paragraph using the opening tag. Most browsers will display HTML correctly even if you forget the end tag. For example:

Example
Try this code »

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

The HTML code in the example above will work in most of the web browsers, but don't rely on it. Forgetting the end tag can produce unexpected results or errors.

Note: For the purposes of forwards-compatibility and good coding practice, it's advisable to use both the opening and closing tags for the paragraphs.

Creating Line Breaks

The `<br>` tag is used to insert a line break on the web page. Since the `<br>` is an empty element, so there is no need of corresponding `</br>` tag.

Example
Try this code »

```
<p>This is a paragraph <br> with line break.</p>
<p>This is <br>another paragraph <br> with line breaks.</p>
```

Note: Don't use the empty paragraph i.e. `<p></p>` to add extra space in your web pages. The browser may ignore the empty paragraphs since it is logical tag. Use the CSS `margin` property instead to adjust the space around the elements.

Creating Horizontal Rules

You can use the `<hr>` tag to create horizontal rules or lines to visually separate content sections on a web page. Like `<br>`, the `<hr>` tag is also an empty element. Here's an example:

Example
Try this code »

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

Managing White Spaces

Normally the browser will display the multiple spaces created inside the HTML code by pressing the *space-bar key* or *tab key* on the keyboard as a single space. Multiple line breaks created inside the HTML code through pressing the enter key is also displayed as a single space.

The following paragraphs will be displayed in a single line without any extra space:

Example
Try this code »

```
<p>This paragraph contains  multiple   spaces    in the source code.</p>
<p>
    This paragraph
    contains multiple tabs and line breaks
    in the source code.
</p>
```

Insert ` ` for creating extra consecutive spaces, while insert `<br>` tag for creating line breaks on your web pages, as demonstrated in the following example:

Example
Try this code »

```
<p>This paragraph has multiple   spaces.</p>
<p>This paragraph has multiple<br><br>line<br><br><br>breaks.</p>
```

Defining Preformatted Text

Sometimes, using ` `, `<br>`, etc. for managing spaces isn't very convenient. Alternatively, you can use the `<pre>` tag to display spaces, tabs, line breaks, etc. exactly as written in the HTML file. It is very helpful in presenting text where spaces and line breaks are important like poem or code.

The following example will display the text in the browser as it is in the source code:

Example
Try this code »

```
<pre>
    Twinkle, twinkle, little star,
    How I wonder what you are!
    Up above the world so high,
    Like a diamond in the sky.
</pre>
```

Tip: Text within the `<pre>` element is typically rendered by the browsers in a monospace or fixed-width font, such as Courier, but you can override this using the CSS `font` property.

HTML Links. *Creating Links in HTML*

A link or hyperlink is a connection from one web resource to another. Links allow users to move seamlessly from one page to another, on any server anywhere in the world.

A link has two ends, called anchors. The link starts at the source anchor and points to the destination anchor, which may be any web resource, for example, an image, an audio or video clip, a PDF file, an HTML document or an element within the document itself, and so on.

By default, links will appear as follow in most of the browsers:

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

However, you can overwrite this using CSS. Learn more about styling links.

HTML Link Syntax

Links are specified in HTML using the `<a>` tag.

A link or hyperlink could be a word, group of words, or image.

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

Anything between the opening `<a>` tag and the closing `</a>` tag becomes the part of the link that the user sees and clicks in a browser. Here are some examples of the links:

Example
Try this code »

```
<a href="https://www.google.com/">Google Search</a>
<a href="https://www.tutorialrepublic.com/">Tutorial Republic</a>
<a href="images/kites.jpg">
    <img src="kites-thumb.jpg" alt="kites">
</a>
```

The `href` attribute specifies the target of the link. Its value can be an absolute or relative URL.

An absolute URL is the URL that includes every part of the URL format, such as protocol, host name, and path of the document,
e.g., `https://www.google.com/`, `https://www.example.com/form.php`, etc. While, relative URLs are page-relative paths, e.g., `contact.html`, `images/smiley.png`, and so on. A relative URL never includes the `http://` or `https://` prefix.

Setting the Targets for Links

The `target` attribute tells the browser where to open the linked document. There are four defined targets, and each target name starts with an underscore(_) character:

- **`_blank`** — Opens the linked document in a new window or tab.

- `_parent` — Opens the linked document in the parent window.
- `_self` — Opens the linked document in the same window or tab as the source document. This is the default, hence it is not necessary to explicitly specify this value.
- `_top` — Opens the linked document in the full browser window.

Try out the following example to understand how the link's target basically works:

Example
Try this code »

```
<a href="/about-us.php" target="_top">About Us</a>
<a href="https://www.google.com/" target="_blank">Google</a>
<a href="images/sky.jpg" target="_parent">
    <img src="sky-thumb.jpg" alt="Cloudy Sky">
</a>
```

Tip: If your web page is placed inside an iframe, you can use the `target="_top"` on the links to break out of the iframe and show the target page in full browser window.

Creating Bookmark Anchors

You can also create bookmark anchors to allow users to jump to a specific section of a web page. Bookmarks are especially helpful if you have a very long web page. Creating bookmarks is a two-step process: first add the `id` attribute on the element where you want to jump, then use that `id` attribute value preceded by the hash sign (`#`) as the value of the `href` attribute of the `<a>` tag, as shown in the following example:

Example
Try this code »

```
<a href="#sectionA">Jump to Section A</a>
<h2 id="sectionA">Section A</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit...</p>
```

Tip: You can even jump to a section of another web page by specifying the URL of that page along with the anchor (i.e. *`#elementId`*) in the `href` attribute, for example, `<a href="mypage.html#topicA">Go to TopicA</a>`.

Creating Download Links

You can also create the file download link in exactly the same fashion as placing text links. Just point the destination URL to the file you want to be available for download. In the following example we've created the download links for **ZIP**, **PDF** and **JPG** files.

Example
Try this code »

```
<a href="downloads/test.zip">Download Zip file</a>
<a href="downloads/masters.pdf">Download PDF file</a>
<a href="downloads/sample.jpg">Download Image file</a>
```

Note: When you click a link that points to a PDF or image file, the file is not downloaded to your hard drive directly. It will only open the file in your web browser. Further you can save or download it to your hard drive on a permanent basis.

HTML Text Formatting

Formatting Text with HTML

HTML provides several tags that you can use to make some text on your web pages to appear differently than normal text, for example, you can use the tag `<b>` to make the text bold, tag `<i>` to make the text italic, tag `<mark>` to highlight the text, tag `<code>` to display a fragment of computer code, tags `<ins>` and `<del>` for marking editorial insertions and deletions, and more. The following example demonstrates the most commonly used formatting tags in action. Now, let's try this out to understand how these tags basically work:

Example
Try this code »

```
<p>This is <b>bold text</b>.</p>
<p>This is <strong>strongly important text</strong>.</p>
<p>This is <i>italic text</i>.</p>
<p>This is <em>emphasized text</em>.</p>
<p>This is <mark>highlighted text</mark>.</p>
<p>This is <code>computer code</code>.</p>
<p>This is <small>smaller text</small>.</p>
<p>This is <sub>subscript</sub> and <sup>superscript</sup> text.</p>
<p>This is <del>deleted text</del>.</p>
<p>This is <ins>inserted text</ins>.</p>
```

By default, the `<strong>` tag is typically rendered in the browser as `<b>`, whereas the `<em>` tag is rendered as `<i>`. However, there is a difference in the meaning of these tags.

Difference between **** and **<b> tags**

Both `<strong>` and `<b>` tags render the enclosed text in a bold typeface by default, but the `<strong>` tag indicates that its contents have strong importance, whereas the `<b>` tag is simply used to draw the reader's attention without conveying any special importance.

Example
Try this code »

```
<p><strong>WARNING!</strong> Please proceed with caution.</p>
<p>The concert will be held at <b>Hyde Park</b> in London.</p>
```

Difference between **** and **<i>** **tag**

Similarly, both `<em>` and `<i>` tags render the enclosed text in italic type by default, but the `<em>` tag indicates that its contents have stressed emphasis compared to surrounding text, whereas the `<i>` tag is used for marking up text that is set off from the normal text for readability reasons, such as a technical term, an idiomatic phrase from another language, a thought, etc.

Example
Try this code »

```
<p>Cats are <em>cute</em> animals.</p>
<p>The <i>Royal Cruise</i> sailed last night.</p>
```

Note: Use the `<em>` and `<strong>` tags when the content of your page requires that certain words or phrases should have strong emphasis or importance. Also, in HTML5 the `<b>` and `<i>` tags have been redefined, earlier they don't have semantic meaning.

Formatting Quotations

You can easily format the quotation blocks from other sources with the HTML `<blockquote>` tag.

Blockquotes are generally displayed with indented left and right margins, along with a little extra space added above and below. Let's try an example to see how it works:

Example
Try this code »

```
<blockquote>
    <p>Learn from yesterday, live for today, hope for tomorrow. The
important thing is not to stop questioning.</p>
    <cite>— Albert Einstein</cite>
</blockquote>
```

Tip: The `cite` tag is used to describe a reference to a creative work. It must include the title of that work or the name of the author (people or organization) or an URL reference.

For short inline quotations, you can use the HTML `<q>` tag. Most browsers display inline quotes by surrounding the text in quotation marks. Here's an example:

Example
Try this code »

```
<p>According to the World Health Organization (WHO): <q>Health is a state of complete physical, mental, and social well-being.</q></p>
```

Showing **Abbreviations**

An abbreviation is a shortened form of a word, phrase, or name. You can use the `<abbr>` tag to denote an abbreviation. The `title` attribute is used inside this tag to provide the full expansion of the abbreviation, which is displayed by the browsers as a tooltip when the mouse cursor is hovered over the element. Let's try out an example:

Example
Try this code »

```
<p>The <abbr title="World Wide Web Consortium">W3C</abbr> is the main international standards organization for the <abbr title="World Wide Web">WWW or W3</abbr>. It was was founded by Tim Berners-Lee.</p>
```

Marking **Contact Addresses**

Web pages often include street or postal addresses. HTML provides a special tag `<address>` to represent contact information (physical and/or digital) for a person, people or organization.

This tag should ideally used to display contact information related to the document itself, such as article's author. Most browsers display an address block in italic. Here's an example:

Example
Try this code »

```
<address>
Mozilla Foundation<br>
331 E. Evelyn Avenue<br>
Mountain View, CA 94041, USA
</address>
```

Please check out the HTML reference section for a complete list of HTML formatting tags.

HTML Styles.*Styling HTML Elements*

HTML is quite limited when it comes to the presentation of a web page. It was originally designed as a simple way of presenting information. CSS (Cascading Style Sheets) was introduced in December 1996 by the World Wide Web Consortium (W3C) to provide a better

way to style HTML elements. With CSS, it becomes easy to specify the things like, size and typeface for the fonts, colors for the text and backgrounds, alignment of the text and images, amount of space between the elements, border and outlines for the elements, and lots of other styling properties.

Adding Styles to HTML Elements

Style information can either be attached as a separate document or embedded in the HTML document itself. These are the three methods of implementing styling information to an HTML document.

- **Inline styles** — Using the `style` attribute in the HTML start tag.
- **Embedded style** — Using the `<style>` element in the head section of the document.
- **External style sheet** — Using the `<link>` element, pointing to an external CSS files.

In this tutorial we will cover all these different types of style sheet one by one.

Note: The inline styles have the highest priority, and the external style sheets have the lowest. It means if you specify styles for your paragraphs in both *embedded* and *external* style sheets, the conflicting style rules in the embedded style sheet would override the external style sheet.

Inline Styles:

Inline styles are used to apply the unique style rules to an element, by putting the CSS rules directly into the start tag. It can be attached to an element using the `style` attribute. *Style attribute has a series of CSS property and value pairs. Each `property: value` pair is separated by a semicolon (;), just as you would write into an embedded or external style sheet. But it needs to be all in one line i.e. no line break after the semicolon.*

The following example demonstrates how to set the `color` and `font-size` of the text:

Example
Try this code »

```
<h1 style="color:red; font-size:30px;">This is a heading</h1>
<p style="color:green; font-size:18px;">This is a paragraph.</p>
<div style="color:green; font-size:18px;">This is some text.</div>
```

Using the inline styles are generally considered as a bad practice. Because style rules are embedded directly inside the html tag, it causes the presentation to become mixed with the content of the document, which makes updating or maintaining a website very difficult.

To learn about the various CSS properties in detail, please check out CSS tutorial section.

Note: It's become impossible to style pseudo-elements and pseudo-classes with inline styles. You should, therefore, avoid the use of `style` attributes in your markup. Using external style sheet is the preferred way to add style information to an HTML document.

Embedded Style Sheets

Embedded or internal style sheets only affect the document they are embedded in.

Embedded style sheets are defined in the `<head>` section of an HTML document using the `<style>` tag. You can define any number of `<style>` elements inside the `<head>` section.

The following example demonstrates how style rules are embedded inside a web page.

Example
Try this code »

```
<head>
    <style>
        body { background-color: YellowGreen; }
             h1 { color: blue; }
        p { color: red; }
    </style>
</head>
```

External Style Sheets

An external style sheet is ideal when the style is applied to many pages. An external style sheet holds all the style rules in a separate document that you can link from any HTML document on your site. External style sheets are the most flexible because with an external style sheet, you can change the look of an entire website by updating just one file.

You can attach external style sheets in two ways — *linking* and *importing*:

Linking External Style Sheets

An external style sheet can be linked to an HTML document using the `<link>` tag.

The `<link>` tag goes inside the `<head>` section, as shown here:

Example
Try this code »

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

Importing External Style Sheets

The `@import` rule is another way of loading an external style sheet. The `@import` statement instructs the browser to load an external style sheet and use its styles. You can use it in two ways. The simplest way is to use it within the `<style>` element in your `<head>` section. Note that, other CSS rules may still be included in the `<style>` element.

Example
Try this code »

```
<style>
    @import url("css/style.css");
    p {
        color: blue;
        font-size: 16px;
    }
</style>
```

Similarly, you can use the `@import` rule to import a style sheet within another style sheet.

Example
Try this code »

```
@import url("css/layout.css");
@import url("css/color.css");
body {
    color: blue;
    font-size: 14px;
}
```

Note: All `@import` rules must occur at the start of the style sheet. Any style rule defined in the style sheet itself override conflicting rule in the imported style sheets. The `@import` rule may cause performance issues, you should generally avoid importing style sheets.

HTML Images

Images into Web Pages Inserting

Images enhance visual appearance of the web pages by making them more interesting and colorful. The `<img>` tag is used to insert images in the HTML documents. It is an empty element and contains attributes only. The syntax of the `<img>` tag can be given with:

```
<img src="url" alt="some_text">
```

The following example inserts three images on the web page:

Example
Try this code »

```
<img src="kites.jpg" alt="Flying Kites">
<img src="sky.jpg" alt="Cloudy Sky">
<img src="balloons.jpg" alt="Balloons">
```

Each image must carry at least two attributes: the `src` attribute, and an `alt` attribute.

The `src` attribute tells the browser where to find the image. Its value is the URL of the image file.

Whereas, the `alt` attribute provides an alternative text for the image, if it is unavailable or cannot be displayed for some reason. Its value should be a meaningful substitute for the image.

Note: Like `<br>` , the `<img>` element is also an empty element, and does not have a closing tag. However, in XHTML it closes itself ending with "`/>`".

Tip: The required `alt` attribute provides alternative text description for an image if a user for some reason cannot able to view it because of slow connection, image is not available at the specified URL, or if the user uses a screen reader or non-graphical browser.

Setting the Width and Height of an Image

The `width` and `height` attributes are used to specify the width and height of an image. The values of these attributes are interpreted in pixels by default.

Example
Try this code »

```
<img src="kites.jpg" alt="Flying Kites" width="300" height="300">
<img src="sky.jpg" alt="Cloudy Sky" width="250" height="150">
<img src="balloons.jpg" alt="Balloons" width="200" height="200">
```

You can also use the `style` attribute to specify width and height for the images. It prevents style sheets from changing the image size accidently, since inline style has the highest priority.

Example
Try this code »

```
<img src="kites.jpg" alt="Flying Kites" style="width: 300px; height:
300px;">
<img src="sky.jpg" alt="Cloudy Sky" style="width: 250px; height: 150px;">
<img src="balloons.jpg" alt="Balloons" style="width: 200px; height:
200px;">
```

Note: It's a good practice to specify both the `width` and `height` attributes for an image, so that browser can allocate that much of space for the image before it is downloaded. Otherwise, image loading may cause distortion or flicker in your website layout.

Using the HTML5 Picture Element

Scaling an image up or down to fit different devices (screen sizes) doesn't work as expected. Also, reducing the image dimension using the `width` and `height` attribute or property doesn't reduce the original file size. To address these problems HTML5 has introduced the `<picture>` tag that allows you to define multiple versions of an image to target different types of devices.

The `<picture>` element contains zero or more `<source>` elements, each referring to different image source, and one `<img>` element at the end. Also each `<source>` element has the `media` attribute which specifies a media condition (similar to the media query) that is used by the browser to determine when a particular source should be used. Let's try out an example:

Example
Try this code »

```
<picture>
    <source media="(min-width: 1000px)" srcset="logo-large.png">
    <source media="(max-width: 500px)" srcset="logo-small.png">
    <img src="logo-default.png" alt="My logo">
</picture>
```

Note: The browser will evaluate each child `<source>` element and choose the best match among them; if no matches are found, the URL of the `<img>` element's `src` attribute is used. Also, the `<img>` tag must be specified as the last child of the `<picture>` element.

Working with Image Maps

An image map allows you to define hotspots on an image that acts just like a hyperlink.

The basic idea behind creating image map is to provide an easy way of linking various parts of an image without dividing it into separate image files. For example, a map of the world may have each country hyperlinked to further information about that country.

Let's try out a simple example to understand how it actually works:

Example
Try this code »

```
<img src="objects.png" usemap="#objects" alt="Objects">
```

```
<map name="objects">
    <area shape="circle" coords="137,231,71" href="clock.html"
alt="Clock">
    <area shape="poly" coords="363,146,273,302,452,300" href="sign.html"
alt="Sign">
    <area shape="rect" coords="520,160,641,302" href="book.html"
alt="Book">
</map>
```

The `name` attribute of the `<map>` tag is used to reference the map from the `<img>` tag using its `usemap` attribute. The `<area>` tag is used inside the `<map>` element to define the clickable areas on an image. You can define any number of clickable areas within an image.

Note: The image map should not be used for website navigation. They are not search engine friendly. A valid use of an image map is with a geographical map.

Tip: There are many online tools available for creating image maps. Some advanced editors like Adobe Dreamweaver also provides a set of tools for easily creating image maps.

HTML Tables

Creating Tables in HTML

HTML table allows you to arrange data into rows and columns. They are commonly used to display tabular data like product listings, customer's details, financial reports, and so on.

You can create a table using the `<table>` element. Inside the `<table>` element, you can use the `<tr>` elements to create rows, and to create columns inside a row you can use the `<td>` elements. You can also define a cell as a header for a group of table cells using the `<th>` element.

The following example demonstrates the most basic structure of a table.

Example
Try this code »

```
<table>
    <tr>
        <th>No.</th>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>1</td>
        <td>Peter Parker</td>
        <td>16</td>
    </tr>
    <tr>
```

```
        <td>2</td>
        <td>Clark Kent</td>
        <td>34</td>
    </tr>
</table>
```

Tables do not have any borders by default. You can use the CSS `border` property to add borders to the tables. Also, table cells are sized just large enough to fit the contents by default. To add more space around the content in the table cells you can use the CSS `padding` property.

The following style rules add a 1-pixel border to the table and 10-pixels of padding to its cells.

Example
Try this code »

```
table, th, td {
    border: 1px solid black;
}
th, td {
    padding: 10px;
}
```

By default, borders around the table and their cells are separated from each other. But you can collapse them into one by using the `border-collapse` property on the `<table>` element.

Also, text inside the `<th>` elements are displayed in bold font, aligned horizontally center in the cell by default. To change the default alignment you can use the CSS `text-align` property.

The following style rules collapse the table borders and align the table header text to left.

Example
Try this code »

```
table {
    border-collapse: collapse;
}
th {
    text-align: left;
}
```

Please check out the tutorial on CSS tables to learn about styling HTML tables in details.

Note: Most of the `<table>` element's attribute such as `border`, `cellpadding`, `cellspacing`, `width`, `align`, etc. for styling table appearances in earlier versions has been dropped in HTML5, so avoid using them. Use CSS to style HTML tables instead.

Spanning Multiple Rows and Columns

Spanning allow you to extend table rows and columns across multiple other rows and columns. Normally, a table cell cannot pass over into the space below or above another table cell. But, you can use the `rowspan` or `colspan` attributes to span multiple rows or columns in a table.

Let's try out the following example to understand how `colspan` basically works:

Example
Try this code »

```
<table>
    <tr>
        <th>Name</th>
        <th colspan="2">Phone</th>
    </tr>
    <tr>
        <td>John Carter</td>
        <td>5550192</td>
        <td>5550152</td>
    </tr>
</table>
```

Similarly, you can use the `rowspan` attribute to create a cell that spans more than one row. Let's try out an example to understand how row spanning basically works:

Example
Try this code »

```
<table>
    <tr>
        <th>Name:</th>
        <td>John Carter</td>
    </tr>
    <tr>
        <th rowspan="2">Phone:</th>
        <td>55577854</td>
    </tr>
    <tr>
        <td>55577855</td>
    </tr>
</table>
```

Adding Captions to Tables

You can specify a caption (or title) for your tables using the `<caption>` element.

The `<caption>` element must be placed directly after the opening `<table>` tag. By default, caption appears at the top of the table, but you can change its position using the CSS `caption-side` property. The following example shows how to use this element in a table.

Example
Try this code »

```
<table>
    <caption>Users Info</caption>
    <tr>
        <th>No.</th>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>1</td>
        <td>Peter Parker</td>
        <td>16</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Clark Kent</td>
        <td>34</td>
    </tr>
</table>
```

Defining a Table Header, Body, and Footer

HTML provides a series of tags `<thead>`, `<tbody>`, and `<tfoot>` that helps you to create more structured table, by defining header, body and footer regions, respectively.

The following example demonstrates the use of these elements.

Example
Try this code »

```
<table>
    <thead>
        <tr>
            <th>Items</th>
            <th>Expenditure</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Stationary</td>
            <td>2,000</td>
        </tr>
        <tr>
```

```
            <td>Furniture</td>
            <td>10,000</td>
        </tr>
    </tbody>
    <tfoot>
        <tr>
            <th>Total</th>
            <td>12,000</td>
        </tr>
    </tfoot>
</table>
```

Note: In HTML5, the `<tfoot>` element can be placed either before or after the `<tbody>` and `<tr>` elements, but must appear after any `<caption>`, `<colgroup>`, and `<thead>` elements.

Tip: Do not use tables for creating web page layouts. Table layouts are slower at rendering, and very difficult to maintain. It should be used only to display tabular data.

HTML Lists. Working with HTML Lists

HTML lists are used to present list of information in well-formed and semantic way. There are three different types of list in HTML and each one has a specific purpose and meaning.

- **Unordered list** — Used to create a list of related items, in no particular order.
- **Ordered list** — Used to create a list of related items, in a specific order.
- **Description list** — Used to create a list of terms and their descriptions.

Note: Inside a list item you can put text, images, links, line breaks, etc. You can also place an entire list inside a list item to create the nested list.

In the following sections we will cover all the three types of list one by one:

HTML Unordered Lists

An unordered list created using the `<ul>` element, and each list item starts with the `<li>` element. The list items in unordered lists are marked with bullets. Here's an example:

Example
Try this code »

```
<ul>
    <li>Chocolate Cake</li>
    <li>Black Forest Cake</li>
    <li>Pineapple Cake</li>
</ul>
```

You can also change the bullet type in your unordered list using the CSS `list-style-type` property. The following style rule changes the type of bullet from the default *disc* to *square*:

Example
Try this code »

```
ul {
    list-style-type: square;
}
```

Please check out the tutorial on CSS lists to learn about styling HTML lists in details.

HTML Ordered Lists

An ordered list created using the `<ol>` element, and each list item starts with the `<li>` element. Ordered lists are used when the order of the list's items is important.

The list items in an ordered list are marked with numbers. Here's an example:

Example
Try this code »

```
<ol>
    <li>Fasten your seatbelt</li>
    <li>Starts the car's engine</li>
    <li>Look around and go</li>
</ol>
```

— The output of the above example will look something like this:

1. Fasten your seatbelt
2. Starts the car's engine
3. Look around and go

The numbering of items in an ordered list typically starts with 1. However, if you want to change that you can use the `start` attribute, as shown in the following example:

Example
Try this code »

```
<ol start="10">
    <li>Mix ingredients</li>
    <li>Bake in oven for an hour</li>
    <li>Allow to stand for ten minutes</li>
</ol>
```

— The output of the above example will look something like this:

10. Mix ingredients
11. Bake in oven for an hour
12. Allow to stand for ten minutes

Like unordered list, you can also use the CSS `list-style-type` property to change the numbering type in an ordered list. The following style rule changes the marker type to roman numbers.

Example
Try this code »

```
ol {
    list-style-type: upper-roman;
}
```

Tip: You can also use the `type` attribute to change the numbering type e.g. `type="I"`. However, you should avoid this attribute, use the CSS `list-style-type` property instead.

HTML Description Lists

A description list is a list of items with a description or definition of each item.

The description list is created using `<dl>` element. The `<dl>` element is used in conjunction with the `<dt>` element which specify a term, and the `<dd>` element which specify the term's definition.

Browsers usually render the definition lists by placing the terms and definitions in separate lines, where the term's definitions are slightly indented. Here's an example:

Example
Try this code »

```
<dl>
    <dt>Bread</dt>
    <dd>A baked food made of flour.</dd>
    <dt>Coffee</dt>
    <dd>A drink made from roasted coffee beans.</dd>
</dl>
```

— The output of the above example will look something like this:

Bread

: A baked food made of flour.

Coffee

: A drink made from roasted coffee beans.

HTML Forms

What is HTML Form

HTML Forms are required to collect different kinds of user inputs, such as contact details like name, email address, phone numbers, or details like credit card information, etc.

Forms contain special elements called controls like inputbox, checkboxes, radio-buttons, submit buttons, etc. Users generally complete a form by modifying its controls e.g. entering text, selecting items, etc. and submitting this form to a web server for further processing.

The `<form>` tag is used to create an HTML form. Here's a simple example of a login form:

Example
Try this code »

```
<form>
    <label>Username: <input type="text"></label>
    <label>Password: <input type="password"></label>
    <input type="submit" value="Submit">
</form>
```

The following section describes different types of controls that you can use in your form.

Input Element

This is the most commonly used element within HTML forms. It allows you to specify various types of user input fields, depending on the `type` attribute. An input element can be of type *text field*, *password field*, *checkbox*, *radio button*, *submit button*, *reset button*, *file select box*, as well as several new input types introduced in HTML5. The most frequently used input types are described below.

Text Fields

Text fields are one line areas that allow the user to input text. Single-line text input controls are created using an `<input>` element, whose `type` attribute has a value of `text`. Here's an example of a single-line text input used to take username:

Example
Try this code »

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

```
</form>
```
— The output of the above example will look something like this:

Username:

Note: The `<label>` tag is used to define the labels for `<input>` elements. If you want your user to enter several lines you should use a `<textarea>` instead.

Password Field

Password fields are similar to text fields. The only difference is; characters in a password field are masked, i.e. they are shown as asterisks or dots. This is to prevent someone else from reading the password on the screen. This is also a single-line text input controls created using an `<input>` element whose `type` attribute has a value of `password`. Here's an example of a single-line password input used to take user password:

Example
Try this code »

```
<form>
    <label for="user-pwd">Password:</label>
    <input type="password" name="user-password" id="user-pwd">
</form>
```
— The output of the above example will look something like this:

Password:

Radio Buttons

Radio buttons are used to let the user select exactly one option from a pre-defined set of options. It is created using an `<input>` element whose `type` attribute has a value of `radio`.

Here's an example of radio buttons that can be used to collect user's gender information:

Example
Try this code »

```
<form>
    <input type="radio" name="gender" id="male">
    <label for="male">Male</label>
    <input type="radio" name="gender" id="female">
    <label for="female">Female</label>
</form>
```
— The output of the above example will look something like this:

Male Female

Checkboxes

Checkboxes allows the user to select one or more option from a pre-defined set of options. It is created using an `<input>` element whose `type` attribute has a value of `checkbox`. Here's an example of checkboxes that can be used to collect information about user's hobbies:

Example
Try this code »

```
<form>
    <input type="checkbox" name="sports" id="soccer">
    <label for="soccer">Soccer</label>
    <input type="checkbox" name="sports" id="cricket">
    <label for="cricket">Cricket</label>
    <input type="checkbox" name="sports" id="baseball">
    <label for="baseball">Baseball</label>
</form>
```

— The output of the above example will look something like this:

Soccer Cricket Baseball

Note: If you want to make a radio button or checkbox selected by default, you can add the attribute `checked` to the input element, like `<input type="checkbox" checked>`.

File Select box

The file fields allow a user to browse for a local file and send it as an attachment with the form data. Web browsers such as Google Chrome and Firefox render a file select input field with a Browse button that enables the user to navigate the local hard drive and select a file.

This is also created using an `<input>` element, whose `type` attribute value is set to `file`.

Example
Try this code »

```
<form>
    <label for="file-select">Upload:</label>
    <input type="file" name="upload" id="file-select">
</form>
```

— The output of the above example will look something like this:

Upload: Choose File No file chosen

Tip: There are several other input types. Please check out the chapter on HTML5 new input types to learn more about the newly introduced input types.

Textarea

Textarea is a multiple-line text input control that allows a user to enter more than one line of text. Multi-line text input controls are created using an `<textarea>` element.

Example
Try this code »

```
<form>
    <label for="address">Address:</label>
    <textarea rows="3" cols="30" name="address" id="address"></textarea>
</form>
```

— The output of the above example will look something like this:

Select Boxes

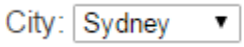
A select box is a dropdown list of options that allows user to select one or more option from a pull-down list of options. Select box is created using the `<select>` element and `<option>` element.

The `<option>` elements within the `<select>` element define each list item.

Example
Try this code »

```
<form>
    <label for="city">City:</label>
    <select name="city" id="city">
        <option value="sydney">Sydney</option>
        <option value="melbourne">Melbourne</option>
        <option value="cromwell">Cromwell</option>
    </select>
</form>
```

— The output of the above example will look something like this:



Submit and Reset Buttons

A submit button is used to send the form data to a web server. When submit button is clicked the form data is sent to the file specified in the form's `action` attribute to process the submitted data. A reset button resets all the forms control to default values. Try out the following example by typing your name in the text field, and click on submit button to see it in action.

Example
Try this code »

```
<form action="action.php" method="post">
    <label for="first-name">First Name:</label>
    <input type="text" name="first-name" id="first-name">
    <input type="submit" value="Submit">
    <input type="reset" value="Reset">
</form>
```

Type your name in the text field above, and click on submit button to see it in action.

Note: You can also create buttons using the `<button>` element. Buttons created with the `<button>` element function just like buttons created with the input element, but they offer richer rendering possibilities by allowing the embedding of other HTML elements.

Grouping Form Controls

You also group logically related controls and labels within a web form using the `<legend>` element. Grouping form controls into categories makes it easier for users to locate a control which makes the form more user-friendly. Let's try out the following example to see how it works:

Example
Try this code »

```
<form>
    <fieldset>
        <legend>Contact Details</legend>
        <label>Email Address: <input type="email" name="email"></label>
        <label>Phone Number: <input type="text" name="phone"></label>
    </fieldset>
</form>
```

Frequently Used Form Attributes

The following table lists the most frequently used form element's attributes:

Attribute	Description
`name`	Specifies the name of the form.
`action`	Specifies the URL of the program or script on the web server that will be used for processing the information submitted via form.
`method`	Specifies the HTTP method used for sending the data to the web server by the browser. The value can be either `get` (the default) and `post`.
`target`	Specifies where to display the response that is received after submitting the form. Possible values are `_blank`, `_self`, `_parent` and `_top`.
`enctype`	Specifies how the form data should be encoded when submitting the form to the server. Applicable only when the value of the `method` attribute is `post`.

There are several other attributes, to know about them please see the `<form>` tag reference.

Note: The `name` attribute represents the form's name within the forms collection. Its value must be unique among the forms in a document, and must not be an empty string.

Tip: All the data sent via `get` method is visible in the browser's address bar. But, the data sent via `post` is not visible to the user. Please check out the tutorial on GET vs. POST to learn about the difference between these two HTTP methods in detail.

HTML Doctypes

Understanding the HTML5 Doctype

A Document Type Declaration, or DOCTYPE for short, is an instruction to the web browser about the version of markup language in which a web page is written.

A DOCTYPE declaration appears at the top of a web page before all other elements. According to the HTML specification or standards, every HTML document requires a valid document type declaration to insure that your web pages are displayed the way they are intended to be displayed.

The doctype declaration is usually the very first thing defined in an HTML document (even before the opening `<html>` tag); however the doctype declaration itself is not an HTML tag.

The DOCTYPE for HTML5 is very short, concise, and case-insensitive.

```
<!DOCTYPE html>
```

Doctypes for earlier versions of HTML were longer because the HTML language was SGML-based and therefore required a reference to a DTD, but they are obsolete now.

With HTML5 this is no longer the case and the doctype declaration is only needed to enable the standard mode for documents written using the HTML syntax.

You can use the following markup as a template to create a new HTML5 document.

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title><!-- Insert your title here --></title>
</head>
<body>
    <!-- Insert your content here -->
</body>
</html>
```

Note: The doctype declaration refers to a Document Type Definition (DTD). It is an instruction to the web browser about what version of the markup language the page is written in. The World Wide Web Consortium (W3C) provides DTDs for all HTML versions.

Tip: Must add a doctype declaration when creating an HTML document. Also, use the W3C's Validator to check your HTML for markup or syntax error before publishing online.

HTML Layout

Creating Website Layouts

Creating a website layout is the activity of positioning the various elements that make a web page in a well-structured manner and give appealing look to the website.

You have seen most websites on the internet usually display their content in multiple rows and columns, formatted like a magazine or newspaper to provide the users a better reading and writing environment. This can be easily achieved by using the HTML tags, such as `<table>`, `<div>`, `<header>`, `<footer>`, `<section>`, etc. and adding some CSS styles to them.

HTML Table Based Layout

Table provides the simplest way for creating layouts in HTML. Generally, this involves the process of putting the contents such as text, images, and so on into rows and columns.

The following layout is created using an HTML table with 3 rows and 2 columns — the first and last row spans both columns using the table's `colspan` attribute:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>HTML Table Layout</title>
</head>
<body style="margin:0px;">
    <table style="width:100%; border-collapse:collapse; font:14px
Arial,sans-serif;">
        <tr>
            <td colspan="2" style="padding:10px 20px; background-
color:#acb3b9;">
                <h1 style="font-size:24px;">Tutorial Republic</h1>
            </td>
        </tr>
        <tr style="height:170px;">
            <td style="width:20%; padding:20px; background-color:#d4d7dc;
vertical-align: top;">
                <ul style="list-style:none; padding:0px; line-
height:24px;">
                    <li><a href="#" style="color:#333;">Home</a></li>
                    <li><a href="#" style="color:#333;">About</a></li>
                    <li><a href="#" style="color:#333;">Contact</a></li>
                </ul>
            </td>
            <td style="padding:20px; background-color:#f2f2f2; vertical-
align:top;">
                <h2>Welcome to our site</h2>
                <p>Here you will learn how to create websites...</p>
            </td>
        </tr>
        <tr>
            <td colspan="2" style="padding:5px; background-color:#acb3b9;
text-align:center;">
                <p>copyright &copy; tutorialrepublic.com</p>
            </td>
        </tr>
    </table>
</body>
```

```
</html>
```
— The HTML code above will produce the following output:

Warning: The method used for creating layout in the above example is not wrong, but it's not recommended. Avoid tables and inline styles for creating layouts. Layouts created using tables often rendered very slowly. Tables should only be used to display tabular data.

HTML Div Based Layout

Using the `<div>` elements is the most common method of creating layouts in HTML.
The `<div>` (stands for division) element is used for marking out a block of content, or set of other elements inside an HTML document. It can contain further other div elements if required. The following example uses the div elements to create a multiple column layout. It will produce the same result as in the previous example that uses table element:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>HTML Div Layout</title>
<style>
    body {
        font: 14px Arial,sans-serif;
        margin: 0px;
    }
    .header {
        padding: 10px 20px;
        background: #acb3b9;
    }
    .header h1 {
        font-size: 24px;
    }
    .container {
        width: 100%;
        background: #f2f2f2;
    }
    .nav, .section {
        float: left;
        padding: 20px;
        min-height: 170px;
        box-sizing: border-box;
    }
    .nav {
        width: 20%;
        background: #d4d7dc;
    }
```

```
    .section {
        width: 80%;
    }
    .nav ul {
        list-style: none;
        line-height: 24px;
        padding: 0px;
    }
    .nav ul li a {
        color: #333;
    }
    .clearfix:after {
        content: ".";
        display: block;
        height: 0;
        clear: both;
        visibility: hidden;
    }
    .footer {
        background: #acb3b9;
        text-align: center;
        padding: 5px;
    }
</style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>Tutorial Republic</h1>
        </div>
        <div class="wrapper clearfix">
            <div class="nav">
                <ul>
                    <li><a href="#">Home</a></li>
                    <li><a href="#">About</a></li>
                    <li><a href="#">Contact</a></li>
                </ul>
            </div>
            <div class="section">
                <h2>Welcome to our site</h2>
                <p>Here you will learn how to create websites...</p>
            </div>
        </div>
        <div class="footer">
            <p>copyright &copy; tutorialrepublic.com</p>
        </div>
    </div>
</body>
</html>
```

— The HTML code above will produce the same output as the previous example:

We've created this layout using the CSS float techniques, since most web browsers support it. Alternatively, you can also use the CSS3 flexbox to create modern and more flexible layouts. See the tutorial on CSS3 flexible box layouts to learn about flexbox in detail.

Tip: Better web page layouts can be created by using the DIV elements and CSS. You can change the layout of all the pages of your website by editing just one CSS file. To learn about CSS in detail, please check out our CSS tutorial section.

Using the HTML5 Structural Elements

HTML5 has introduced the new structural elements such as `<header>`, `<footer>`, `<nav>`, `<section>`, etc. to define the different parts of a web page in a more semantic way.

You can consider these elements as a replacement for commonly used classes such as `.header`, `.footer`, `.nav`, `.section`, etc. The following example uses the new HTML5 structural elements to create the same layout that we have created in the previous examples.

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>HTML5 Web Page Layout</title>
<style>
    body {
        font: 14px Arial,sans-serif;
        margin: 0px;
    }
    header {
        padding: 10px 20px;
        background: #acb3b9;
    }
    header h1 {
        font-size: 24px;
    }
    .container {
        width: 100%;
        background: #f2f2f2;
    }
    nav, section {
        float: left;
        padding: 20px;
        min-height: 170px;
        box-sizing: border-box;
    }
```

```
    section {
        width: 80%;
    }
    nav {
        width: 20%;
        background: #d4d7dc;
    }
    nav ul {
        list-style: none;
        line-height: 24px;
        padding: 0px;
    }
    nav ul li a {
        color: #333;
    }
    .clearfix:after {
        content: ".";
        display: block;
        height: 0;
        clear: both;
        visibility: hidden;
    }
    footer {
        background: #acb3b9;
        text-align: center;
        padding: 5px;
    }
</style>
</head>
<body>
    <div class="container">
        <header>
            <h1>Tutorial Republic</h1>
        </header>
        <div class="wrapper clearfix">
            <nav>
                <ul>
                    <li><a href="#">Home</a></li>
                    <li><a href="#">About</a></li>
                    <li><a href="#">Contact</a></li>
                </ul>
            </nav>
            <section>
                <h2>Welcome to our site</h2>
                <p>Here you will learn how to create websites...</p>
            </section>
        </div>
        <footer>
            <p>copyright &copy; tutorialrepublic.com</p>
        </footer>
    </div>
</body>
</html>
```

— The HTML code above will also produce the same output as the previous example:

The following table provides a brief overview of new HTML5 structural elements.

Tag	Description
<header>	Represents the header of a document or a section.
<footer>	Represents the footer of a document or a section.
<nav>	Represents a section of navigation links.
<section>	Represents a section of a document, such as header, footer etc.
<article>	Represents an article, blog post, or other self-contained unit of information.
<aside>	Represents some content loosely related to the page content.

Please check out the reference on HTML5 tags to know about the newly introduced tags.

HTML5 Tags/Elements

HTML5 Tags

The following section contains a complete list of standard tags belonging to the latest HTML5 and XHTML 1.1 specifications. All the tags are listed alphabetically.

Tag	Description
<a>	Defines a hyperlink.
<abbr>	Defines an abbreviated form of a longer word or phrase.
<acronym>	Obsolete Defines an acronym. Use <abbr> instead.
<address>	Specifies the author's contact information.
<applet>	Obsolete Embeds a Java applet (mini Java applications) on the page. Use <object> instead.
<area>	Defines a specific area within an image map.

Tag	Description
<article>	Defines an article.
<aside>	Defines some content loosely related to the page content.
<audio>	Embeds a sound, or an audio stream in an HTML document.
<b>	Displays text in a bold style.
<base>	Defines the base URL for all relative URLs in a document.
<basefont>	Obsolete Specifies the base font for a page. Use CSS instead.
<bdi>	Represents text that is isolated from its surrounding for the purposes of bidirectional text formatting.
<bdo>	Overrides the current text direction.
<big>	Obsolete Displays text in a large size. Use CSS instead.
<blockquote>	Represents a section that is quoted from another source.
<body>	Defines the document's body.
 	Produces a single line break.
<button>	Creates a clickable button.
<canvas>	Defines a region in the document, which can be used to draw graphics on the fly via scripting (usually JavaScript).
<caption>	Defines the caption or title of the table.
	Obsolete Align contents in the center. Use CSS instead.
<cite>	Indicates a citation or reference to another source.
<code>	Specifies text as computer code.

Tag	Description
`<col>`	Defines attribute values for one or more columns in a table.
`<colgroup>`	Specifies attributes for multiple columns in a table.
`<data>`	Links a piece of content with a machine-readable translation.
`<datalist>`	Represents a set of pre-defined options for an `<input>` element.
`<dd>`	Specifies a description, or value for the term (`<dt>`) in a description list (`<dl>`).
`<del>`	Represents text that has been deleted from the document.
`<details>`	Represents a widget from which the user can obtain additional information or controls on-demand.
`<dfn>`	Specifies a definition.
`<dialog>`	Defines a dialog box or subwindow.
`<dir>`	Obsolete Defines a directory list. Use `<ul>` instead.
`<div>`	Specifies a division or a section in a document.
`<dl>`	Defines a description list.
`<dt>`	Defines a term (an item) in a description list.
`<em>`	Defines emphasized text.
`<embed>`	Embeds external application, typically multimedia content like audio or video into an HTML document.
`<fieldset>`	Specifies a set of related form fields.
`<figcaption>`	Defines a caption or legend for a figure.
`<figure>`	Represents a figure illustrated as part of the document.

Tag	Description
<font>	Obsolete Defines font, color, and size for text. Use CSS instead.
<footer>	Represents the footer of a document or a section.
<form>	Defines an HTML form for user input.
<frame>	Obsolete Defines a single frame within a frameset.
<frameset>	Obsolete Defines a collection of frames or other frameset.
<head>	Defines the head portion of the document that contains information about the document such as title.
<header>	Represents the header of a document or a section.
<hgroup>	Defines a group of headings.
<h1> to <h6>	Defines HTML headings.
<hr>	Produce a horizontal line.
<html>	Defines the root of an HTML document.
<i>	Displays text in an italic style.
<iframe>	Displays a URL in an inline frame.
<img>	Represents an image.
<input>	Defines an input control.
<ins>	Defines a block of text that has been inserted into a document.
<kbd>	Specifies text as keyboard input.
<keygen>	Represents a control for generating a public-private key pair.

Tag	Description
`<label>`	Defines a label for an `<input>` control.
`<legend>`	Defines a caption for a `<fieldset>` element.
`<li>`	Defines a list item.
`<link>`	Defines the relationship between the current document and an external resource.
`<main>`	Represents the main or dominant content of the document.
`<map>`	Defines a client-side image-map.
`<mark>`	Represents text highlighted for reference purposes.
`<menu>`	Represents a list of commands.
`<menuitem>`	Defines a list (or menuitem) of commands that a user can perform.
`<meta>`	Provides structured metadata about the document content.
`<meter>`	Represents a scalar measurement within a known range.
`<nav>`	Defines a section of navigation links.
`<noframes>`	Obsolete Defines an alternate content that displays in browsers that do not support frames.
`<noscript>`	Defines alternative content to display when the browser doesn't support scripting.
`<object>`	Defines an embedded object.
`<ol>`	Defines an ordered list.
`<optgroup>`	Defines a group of related options in a selection list.
`<option>`	Defines an option in a selection list.
`<output>`	Represents the result of a calculation.

Tag	Description
<p>	Defines a paragraph.
<param>	Defines a parameter for an object or applet element.
<picture>	Defines a container for multiple image sources.
<pre>	Defines a block of preformatted text.
<progress>	Represents the completion progress of a task.
<q>	Defines a short inline quotation.
<rp>	Provides fall-back parenthesis for browsers that that don't support ruby annotations.
<rt>	Defines the pronunciation of character presented in a ruby annotations.
<ruby>	Represents a ruby annotation.
<s>	Represents contents that are no longer accurate or no longer relevant.
<samp>	Specifies text as sample output from a computer program.
<script>	Places script in the document for client-side processing.
<section>	Defines a section of a document, such as header, footer etc.
<select>	Defines a selection list within a form.
<small>	Displays text in a smaller size.
<source>	Defines alternative media resources for the media elements like `<audio>` or `<video>`.
	Defines an inline styleless section in a document.
<strike>	Obsolete Displays text in strikethrough style.
	Indicate strongly emphasized text.

Tag	Description
`<style>`	Inserts style information (commonly CSS) into the head of a document.
`<sub>`	Defines subscripted text.
`<summary>`	Defines a summary for the `<details>` element.
`<sup>`	Defines superscripted text.
`<svg>`	Embed SVG (Scalable Vector Graphics) content in an HTML document.
`<table>`	Defines a data table.
`<tbody>`	Groups a set of rows defining the main body of the table data.
`<td>`	Defines a cell in a table.
`<template>`	Defines the fragments of HTML that should be hidden when the page is loaded, but can be cloned and inserted in the document by JavaScript.
`<textarea>`	Defines a multi-line text input control (text area).
`<tfoot>`	Groups a set of rows summarizing the columns of the table.
`<th>`	Defines a header cell in a table.
`<thead>`	Groups a set of rows that describes the column labels of a table.
`<time>`	Represents a time and/or date.
`<title>`	Defines a title for the document.
`<tr>`	Defines a row of cells in a table.
`<track>`	Defines text tracks for the media elements like `<audio>` or `<video>`.
`<tt>`	Obsolete Displays text in a teletype style.

Tag	Description
`<u>`	Displays text with an underline.
`<ul>`	Defines an unordered list.
`<var>`	Defines a variable.
`<video>`	Embeds video content in an HTML document.
`<wbr>`	Represents a line break opportunity.

HTML Head

The HTML head Element

The `<head>` element primarily is the container for all the head elements, which provide extra information about the document (metadata), or reference to other resources that are required for the document to display or behave correctly in a web browser. The head elements collectively describes the properties of the document such as title, provide meta information like character set, instruct the browser where to find the style sheets or scripts that allows you to extend the HTML document in a highly active and interactive ways.

The HTML elements that can be used inside the `<head>` element are: `<title>`, `<base>`, `<link>`, `<style>`, `<meta>`, `<script>` and the `<noscript>` element.

The HTML title Element

The `<title>` element defines the title of the document.

The title element is required in all HTML/XHTML documents to produce a valid document. Only one title element is permitted in a document and it must be placed within the `<head>` element. The title element contains plain text and entities; it may not contain other markup tags.

The title of the document may be used for different purposes. For example:

- To display a title in the browser title bar and in the task bar.
- To provide a title for the page when it is added to favorites or bookmarked.

- To displays a title for the page in search-engine results.

The following example demonstrates how to place title in an HTML document.

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>A simple HTML document</title>
</head>
<body>
    <p>Hello World!</p>
</body>
</html>
```

Tip: A good title should be short and specific to the document's content, because search engine's web crawlers pay particular attention to the words used in the title. The title should ideally be less than 65 characters in length. See the guidelines for titles.

The HTML base Element

The HTML `<base>` element is used to define a base URL for all relative links contained in the document, e.g. you can set the base URL once at the top of your page, and then all subsequent relative links will use that URL as a starting point. Here's an example:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Defining a base URL</title>
    <base href="https://www.tutorialrepublic.com/">
</head>
<body>
    <p><a href="html-tutorial/html-head.php">HTML Head</a>.</p>
</body>
</html>
```

The hyperlink in the example above will actually resolve to `https://www.tutorialrepublic.com/html-tutorial/html-head.php` regardless of the URL of the current page. This is because the relative URL specified in the link: `html-tutorial/html-head.php` is added to the end of the base URL: `https://www.tutorialrepublic.com/`.

Warning: The HTML `<base>` element must appear before any element that refers to an external resource. HTML permits only one base element for each document.

The HTML link Element

The `<link>` element defines the relationship between the current document and an external documents or resources. A common use of link element is to link to external style sheets.

Example
Try this code »

```
<head>
    <title>Linking Style Sheets</title>
    <link rel="stylesheet" href="style.css">
</head>
```

Please check out the CSS tutorial section to learn about style sheets in detail.

Note: An HTML document's `<head>` element may contain any number of `<link>` elements. The `<link>` element has attributes, but no contents.

The HTML style Element

The `<style>` element is used to define embedded style information for an HTML document. The style rules inside the `<style>` element specify how HTML elements render in a browser.

Example
Try this code »

```
<head>
    <title>Embedding Style Sheets</title>
    <style>
        body { background-color: YellowGreen; }
        h1 { color: red; }
        p { color: green; }
    </style>
</head>
```

Note: An embedded style sheet should be used when a single document has a unique style. If the same style sheet is used in multiple documents, then an external style sheet would be more appropriate. See the tutorial on HTML styles to learn more about it.

The HTML meta Element

The `<meta>` element provides metadata about the HTML document. Metadata is a set of data that describes and gives information about other data. Here's an example:

Example
Try this code »

```
<head>
    <title>Specifying Metadata</title>
    <meta charset="utf-8">
    <meta name="author" content="John Smith">
</head>
```

The meta element will be explained in more detail in the next chapter.

The HTML script Element

The `<script>` element is used to define client-side script, such as JavaScript in HTML documents.

The following example will display a greeting message in the browser:

Example
Try this code »

```
<head>
    <title>Adding JavaScript</title>
    <script>
        document.write("<h1>Hello World!</h1>")
    </script>
</head>
```

HTML Meta

Defining Metadata

The `<meta>` tags are typically used to provide structured metadata such as a document's *keywords*, *description*, *author name*, *character encoding*, and other metadata. Any number of meta tags can be placed inside the head section of an HTML or XHTML document.

Metadata will not be displayed on the web page, but will be machine parsable, and can be used by the browsers, search engines like Google or other web services.

The following section describes the use of meta tags for various purposes.

Declaring Character Encoding in HTML

Meta tag typically used to declare character encoding inside HTML document.

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Declaring Character Encoding</title>
    <meta charset="utf-8">
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

To set the character encoding inside a CSS document, the `@charset` at-rule is used.

Note: UTF-8 is a very versatile and recommended character encoding to choose. However, if this is not specified, then the default encoding of the platform is used.

Defining the Author of a Document

You can also use the meta tag to clearly define who is the author or creator of the web page.

The author can be an individual, the company as a whole or a third party.

Example
Try this code »

```
<head>
    <title>Defining Document's Author</title>
    <meta name="author" content="Alexander Howick">
</head>
```

Note: The `name` attribute of the meta tag defines the name of a piece of document-level metadata, while the `content` attribute gives the corresponding value. The `content` attribute value may contain text and entities, but it may not contain HTML tags.

Keywords and Description for Search Engines

Some search engines use metadata especially keywords and descriptions to index web pages; however this may not necessarily be true. Keywords giving extra weight to a document's keywords and description provide a short synopsis of the page. Here's an example:

Example
Try this code »

```
<head>
    <title>Defining Keywords and Description</title>
    <meta name="keywords" content="HTML, CSS, javaScript">
    <meta name="description" content="Easy to understand tutorials and
references on HTML, CSS, javaScript and more...">
</head>
```

Tip: Search engines will often use the meta description of a page to create short synopsis for the page when it appear in search results. See the guidelines for meta description.

Configuring the Viewport for Mobile Devices

You can use the viewport meta tag to display the web pages correctly on mobile devices.

Without a viewport meta tag, mobile browsers render the web pages at typical desktop screen widths, and then scale it down to fit the mobile screen. As a result, it requires pinch-and-zoom to view the web page properly in mobile devices, which is very inconvenient.The following demonstration shows two web pages — one *with viewport meta tag* and other *without viewport meta tag* set. Open these links on mobile devices to see how it works.

With Viewport Meta Tag

Without Viewport Meta Tag

The viewport meta tag allows you to set the best viewport size and scaling limits for viewing the web pages on mobile devices. A typical viewport meta tag definition will look as follows:

Example
Try this code »

```
<head>
    <title>Configuring the Viewport</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
```

The `width=device-width` key-value pair inside the `content` attribute sets the width of the viewport to same as the screen width of the device, whereas the `initial-scale=1` sets the initial scale or zoom level to 100% when the page is first loaded by the browser.

Tip: Always use the `<meta>` viewport tag in your web pages. It will make your website user-friendly and more accessible on mobile devices like cell phones and tablets.

HTML Script

Working with Client-side Script

Client-side scripting refers to the type of computer programs that are executed by the user's web browser. JavaScript (JS) is the most popular client-side scripting language on the web.

The `<script>` element is used to embed or reference JavaScript within an HTML document to add interactivity to web pages and provide a significantly better user experience.

Some of the most common uses of JavaScript are form validation, generating alert messages, creating image gallery, show hide content, DOM manipulation, and many more.

Adding JavaScript to HTML Pages

JavaScript can either be embedded directly inside the HTML page or placed in an external script file and referenced inside the HTML page. Both methods use the `<script>` element.

Embedding JavaScript

To embed JavaScript in an HTML file, just add the code as the content of a `<script>` element.

The JavaScript in the following example write a string of text to a web page.

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Embedding JavaScript</title>
</head>
<body>
    <div id="greet"></div>
    <script>
        document.getElementById("greet").innerHTML = "Hello World!";
    </script>
</body>
</html>
```

Tip: Ideally, script elements should be placed at the bottom of the page, before the closing body tag i.e. `</body>`, because when browser encounters a script it pauses rendering the rest of the page until it parses the script that may significantly impact your site performance.

Calling External JavaScript File

You can also place your JavaScript code into a separate file (with a `.js` extension), and call that file in your HTML document through the `src` attribute of the `<script>` tag. This is useful if you want the same script available to multiple documents. It saves you from repeating the same task over and over again and makes your website much easier to maintain. The following example demonstrates how to link external JS file in HTML.

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Linking External JavaScript</title>
</head>
<body>
    <div id="greet"></div>
    <script src="hello.js"></script>
</body>
</html>
```

Note: When the `src` attribute is specified, the `<script>` element must be empty. This simply means that you cannot use the same `<script>` element to both embed the JavaScript and to link to an external JavaScript file in an HTML document.

Tip: JavaScript files are normal text files with `.js` extension, such as "hello.js". Also, the external JavaScript file contains only JavaScript statements; it does not contain the `<script>...</script>` element like embedded JavaScript.

The HTML noscript Element

The `<noscript>` element is used to provide an alternate content for users that have either disabled JavaScript in their browser or have a browser that doesn't support client-side scripting.

This element can contain any HTML elements, aside from `<script>`, that can be included in the `<body>` element of a normal HTML page. Let's check out an example:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Noscript Demo</title>
</head>
<body>
    <div id="greet"></div>
    <script>
        document.getElementById("greet").innerHTML = "Hello World!";
    </script>
    <noscript>
        <p>Oops! This website requires a JavaScript-enabled browser.</p>
    </noscript>
</body>
</html>
```

Note: The content inside the `noscript` element will only be displayed if the user's browser doesn't support scripting, or scripting is disabled in the browser.

HTML Entities. What is HTML Entity?

Some characters are reserved in HTML, e.g. you cannot use the less than (<) or greater than (>) signs or angle brackets within your text, because the browser could mistake them for markup, while some characters are not present on the keyboard like copyright symbol ©.

To display these special characters, they must be replaced with the character entities. Character entity references, or entities for short, enable you to use the characters that cannot be expressed in the document's character encoding or that cannot be entered by a keyboard.

Frequently Used HTML Character Entities

Result	Description	Entity Name	Numerical reference
	non-breaking space		
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	quotation mark	&quot;	&#34;
'	apostrophe	&apos;	&#39;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
©	copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;

™	trademark	™	™

You can use numeric character references, instead of entity names. A key benefit of using numeric character references is that, they have stronger browser support, and can be used to specify any Unicode character, whereas entities are limited to a subset of this.

Note: HTML entities names are case-sensitive! Please check out the HTML character entities reference for a complete list of character entities of special characters and symbols.

Tip: Nonbreaking space (` `) is used to create a blank space between two items that cannot be separated by a line break. They are also used to display multiple spaces since web browsers display only one space if multiple spaces are created using the spacebar key.

HTML URL. What is URL?

URL stands for **U**niform **R**esource **L**ocator is the global address of documents and other resources on the World Wide Web. Its main purpose is to identify the location of a document and other resources available on the internet, and specify the mechanism for accessing it through a web browser.

For instance, if you look at the address bar of your browser you will see:

```
https://www.tutorialrepublic.com/html-tutorial/html-url.php
```

— This is the URL of the web page you are viewing right now.

The URL Syntax

The general syntax of URLs is the following:

```
scheme://host:port/path?query-string#fragment-id
```

A URL has a linear structure and normally consists of some of the following:

Scheme name — The scheme identifies the protocol to be used to access the resource on the Internet. The scheme names followed by the three characters `://` (a colon and two slashes). The most commonly used protocols are `http://`, `https://`, `ftp://`, and `mailto://`.

Host name — The host name identifies the host where resource is located. A hostname is a domain name assigned to a host computer. This is usually a combination of the host's local name with its parent domain's name. For example, `www.tutorialrepublic.com` consists of host's machine name `www` and the domain name `tutorialrepublic.com`.

Port Number — Servers often deliver more than one type of service, so you must also tell the server what service is being requested. These requests are made by port number. Well-known

port numbers for a service are normally omitted from the URL. For example, web service HTTP runs by default over port 80, HTTPS runs by default over port 443.

Path — The path identifies the specific resource within the host that the user wants to access. For example, `/html/html-url.php`, `/news/technology/`, etc.

Query String — The query string contains data to be passed to server-side scripts, running on the web server. For example, parameters for a search. The query string preceded by a question mark (`?`), is usually a string of name and value pairs separated by ampersand (`&`), for example, `?first_name=John&last_name=Corner`, `q=mobile+phone`, and so on.

Fragment identifier — The fragment identifier, if present, specifies a location within the page. Browser may scroll to display that part of the page. The fragment identifier introduced by a hash character (`#`) is the optional last part of a URL for a document.

Note: Scheme and host components of a URL are not case-sensitive, but path and query string are case-sensitive. Usually the whole URL is specified in lower case.

HTML URL Encoding

What is URL Encoding

According to RFC 3986, the characters in a URL only limited to a defined set of reserved and unreserved US-ASCII characters. Any other characters are not allowed in a URL. But URL often contains characters outside the US-ASCII character set, so they must be converted to a valid US-ASCII format for worldwide interoperability. URL-encoding, also known as percent-encoding is a process of encoding URL information so that it can be safely transmitted over the internet. To map the wide range of characters that is used worldwide, a two-step process is used:

- At first the data is encoded according to the UTF-8 character encoding.
- Then only those bytes that do not correspond to characters in the unreserved set should be percent-encoded like %HH, where HH is the hexadecimal value of the byte.

For example, the string: **François** would be encoded as: **Fran%C3%A7ois**

Ç, ç (c-cedilla) is a Latin script letter.

Reserved Characters

Certain characters are reserved or restricted from use in a URL because they may (or may not) be defined as delimiters by the generic syntax in a particular URL scheme. For example, forward slash `/` characters are used to separate different parts of a URL.

If data for a URL component contains character that would conflict with a reserved set of characters, which is defined as a delimiter in the URL scheme then the conflicting character must be percent-encoded before the URL is formed. Reserved characters in a URL are:

!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

Unreserved Characters

Characters that are allowed in a URL but do not have a reserved purpose are called unreserved. These include uppercase and lowercase letters, decimal digits, hyphen, period, underscore, and tilde. The following table lists all the unreserved characters in a URL:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	-	_	.	~												

URL Encoding Converter

The following converter encodes and decodes the characters according to RFC 3986.

HTML Validation

Why Validate Your HTML Code

As a beginner it is very common that you will make mistake in writing your HTML code. Incorrect or non-standard code may cause unexpected results in how your page displayed or function in browsers. To prevent this you can test or validate your HTML code against the formal guidelines and standards set by the Wide Web Consortium (W3C) for HTML/XHTML web pages. The World Wide Web Consortium provide a simple online tool (https://validator.w3.org/) that automatically check your HTML code and point out any problems/errors your code might have, such as missing closing tags or missing quotes around attributes.

Validating a Web Page

Validating a web page is a process of ensuring that it conforms to the norms or standards defined by the World Wide Web Consortium (W3C) which is the authority to maintain HTML standards.

There are several specific reasons for validating a web page, some of them are:

- It helps to create web pages that are cross-browser, cross-platform compatible. It also likely to be compatible with the future version of web browsers and web standards.
- Standards compliant web pages increase the search engine spiders and crawlers visibility, as a result your web pages will more likely be appear in search results.
- It will reduce unexpected errors and make your web pages more accessible to the visitor.

Note: Validation is important. It will ensure that your web pages are interpreted in the same way (the way you want it) by the various web browsers, search engines etc.

Follow the link given below to validate your HTML/XHTML document. It will automatically detect whether you're using HTML or XHTML, and which doctype you're using.

`W3C Markup Validation Service`

HTML5 New Input Types

New Input Types in HTML5

HTML5 introduces several new `<input>` types like email, date, time, color, range, and so on. to improve the user experience and to make the forms more interactive. However, if a browser failed to recognize these new input types, it will treat them like a normal text box.

In this section we're going to take a brief look at each of the following new input types:

- color
- date
- datetime-local
- email
- month
- number
- range

- search
- tel
- time
- url
- week

There was also a `datetime` input type for entering a date and time, but it is now obsolete.

Input Type Color

The `color` input type allows the user to select a color from a color picker and returns the color value in hexadecimal format (`#rrggbb`). If you don't specify a value, the default is `#000000`, which is black.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
<form>
    <label for="mycolor">Select Color:</label>
    <input type="color" value="#00ff00" id="mycolor">
</form>
```

Note: The color input (i.e. `type="color"`) is supported in all major modern web browsers such as Firefox, Chrome, Opera, Safari (12.1+), Edge (14+). Not supported by the Microsoft Internet Explorer and older version of Apple Safari browsers.

Input Type Date

The `date` input type allows the user to select a date from a drop-down calendar.

The date value includes the year, month, and day, but not the time.

Example
Try this code »

```
<form>
    <label for="mydate">Select Date:</label>
    <input type="date" value="2019-04-15" id="mydate">
</form>
```

Note: The date input (i.e. `type="date"`) is supported by the Chrome, Firefox, Opera and Edge browsers. Not supported by the Internet Explorer and Safari browsers.

Input Type Datetime-local

The `datetime-local` input type allows the user to select both local date and time, including the year, month, and day as well as the time in hours and minutes.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
<form>
    <label for="mydatetime">Choose Date and Time:</label>
    <input type="datetime-local" id="mydatetime">
</form>
```

Warning: The input `type="datetime-local"` is not supported by Firefox, Safari, and Internet Explorer browsers. Currently supported by Chrome, Edge, and Opera browsers.

Input Type Email

The `email` input type allows the user to enter e-mail address. It is very similar to a standard text input type, but if it is used in combination with the `required` attribute, the browser may look for the patterns to ensure a properly-formatted e-mail address should be entered.

Let's try out this example by entering any e-mail address to see how it actually works:

Example
Try this code »

```
<form>
    <label for="myemail">Enter Email Address:</label>
    <input type="email" id="myemail" required>
</form>
```

Tip: You can style the email input field for different validation states, when an value is entered using the `:valid`, `:invalid` or `:required` pseudo-classes.

Note: The validation for the email input (i.e. `type="email"`) is supported by all major browsers like Firefox, Chrome, Safari, Opera, Internet Explorer 10 and above.

Input Type Month

The `month` input type allows the user to select a month and year from a drop-down calendar.

The value is a string in the format "YYYY-MM", where YYYY is the four-digit year and MM is the month number. Let's try out an example to see how this basically works:

Example
Try this code »

```
<form>
    <label for="mymonth">Select Month:</label>
    <input type="month" id="mymonth">
</form>
```

Warning: The input `type="month"` is not supported by Firefox, Safari and Internet Explorer browsers. Currently supported in Chrome, Edge, and Opera browsers.

Input Type Number

The `number` input type can be used for entering a numerical value. You can also restrict the user to enter only acceptable values using the additional attributes `min`, `max`, and `step`.

The following example will allow you to enter a numeric value between 1 to 10.

Example
Try this code »

```
<form>
    <label for="mynumber">Enter a Number:</label>
    <input type="number" min="1" max="10" step="0.5" id="mynumber">
</form>
```

Note: The number input (i.e. `type="number"`) is supported by all major web browsers such as Firefox, Chrome, Safari, Opera, Internet Explorer 10 and above. Internet Explorer however recognized the number but do not provide increment and decrement spin buttons.

Input Type Range

The `range` input type can be used for entering a numerical value within a specified range. It works very similar to `number` input, but it offers a simpler control for entering a number.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
<form>
    <label for="mynumber">Select a Number:</label>
    <input type="range" min="1" max="10" step="0.5" id="mynumber">
</form>
```

Note: The range input (i.e. `type="range"`) is supported by all major web browsers such as Firefox, Chrome, Safari, Opera, Internet Explorer 10 and above.

Input Type Search

The `search` input type can be used for creating search input fields.

A search field typically behaves like a regular text field, but in some browsers like Chrome and Safari as soon as you start typing in the search box a small cross appears on the right side of the field that lets you quickly clear the search field. Let's try out an example to see how it works:

Example
Try this code »

```
<form>
    <label for="mysearch">Search Website:</label>
    <input type="search" id="mysearch">
</form>
```

Note: The search input (i.e. `type="search"`) is supported by all major web browsers such as Firefox, Chrome, Safari, Opera, Internet Explorer 10 and above.

Input Type Tel

The `tel` input type can be used for entering a telephone number.

Browsers don't support tel input validation natively. However, you can use the `placeholder` attribute to help users in entering the correct format for a phone number, or specify a regular expression to validate the user input using the `pattern` attribute. Let's check out an example:

Example
Try this code »

```
<form>
    <label for="myphone">Telephone Number:</label>
    <input type="tel" id="myphone" placeholder="xx-xxxx-xxxx" required>
</form>
```

Note: The validation for tel input (i.e. `type="tel"`) is currently not supported by any browser because format for phone numbers vary so much across countries, but it is still useful. Mobile browsers display a numeric keyboard for tel input field for entering phone numbers.

Input Type Time

The `time` input type can be used for entering a time (hours and minutes).

Browser may use 12- or 24-hour format for inputting times, based on local system's time setting.

Example
Try this code »

```
<form>
    <label for="mytime">Select Time:</label>
    <input type="time" id="mytime">
</form>
```

Warning: The input `type="time"` is not supported by Internet Explorer and Safari browsers. Currently supported by Chrome, Firefox, Edge, and Opera browsers.

Input Type URL

The `url` input type can be used for entering URL's or web addresses.

You can use the `multiple` attribute to enter more than one URL. Also, if `required` attribute is specified browser will automatically carry out validation to ensure that only text that matches the standard format for URLs is entered into the input box. Let's see how this works:

Example
Try this code »

```
<form>
    <label for="myurl">Enter Website URL:</label>
    <input type="url" id="myurl" required>
</form>
```

Note: The validation for the url input (i.e. `type="url"`) is supported by all major browsers like Firefox, Chrome, Safari, Opera, Internet Explorer 10 and above.

Input Type Week

The `week` input type allows the user to select a week and year from a drop-down calendar.

Let's try out the following example to understand how this works:

Example
Try this code »

```
<form>
    <label for="myweek">Select Week:</label>
    <input type="week" id="myweek">
</form>
```

Warning: The input `type="week"` is not supported by Firefox, Safari and Internet Explorer browsers. Currently supported by Chrome, Edge, and Opera browsers.

HTML5 Canvas. What is Canvas?

The HTML5 canvas element can be used to draw graphics on the webpage via JavaScript. The canvas was originally introduced by Apple for the Mac OS dashboard widgets and to power graphics in the Safari web browser. Later it was adopted by the Firefox, Google Chrome and Opera. Now the canvas is a part of the new HTML5 specification for next generation web technologies. By default the `<canvas>` element has 300px of width and 150px of height without any border and content. However, custom width and height can be defined using the CSS `height` and `width` property whereas the border can be applied using the CSS `border` property.

Differences between SVG and Canvas

The HTML5 introduced the two new graphical elements `<canvas>` and `<svg>` for creating rich graphics on the web, but they are fundamentally different.

The following table summarizes some of the basic differences between these two elements, which will help you to understand how to use these elements effectively and appropriately.

SVG	Canvas
Vector based (composed of shapes)	Raster based (composed of pixel)
Multiple graphical elements, which become the part of the page's DOM tree	Single element similar to `<img>` in behavior. Canvas diagram can be saved to PNG or JPG format
Modified through script and CSS	Modified through script only
Good text rendering capabilities	Poor text rendering capabilities
Give better performance with smaller number of objects or larger surface, or both	Give better performance with larger number of objects or smaller surface, or both
Better scalability. Can be printed with high quality at any resolution. Pixelation does not occur	Poor scalability. Not suitable for printing on higher resolution. Pixelation may occur

HTML5 Audio

Embedding Audio in HTML Document

Inserting audio onto a web page was not easy before, because web browsers did not have a uniform standard for defining embedded media files like audio.

In this chapter we'll demonstrates some of the many ways to embed sound in your webpage, from the use of a simple link to the use of the latest HTML5 `<audio>` element.

Using the HTML5 audio Element

The newly introduced HTML5 `<audio>` element provides a standard way to embed audio in web pages. However, the audio element is relatively new but it works in most of the modern web browsers.

The following example simply inserts an audio into the HTML5 document, using the browser default set of controls, with one source defined by the `src` attribute.

Example
Try this code »

```
<audio controls="controls" src="media/birds.mp3">
    Your browser does not support the HTML5 Audio element.
</audio>
```

An audio, using the browser default set of controls, with alternative sources.

Example
Try this code »

```
<audio controls="controls">
    <source src="media/birds.mp3" type="audio/mpeg">
    <source src="media/birds.ogg" type="audio/ogg">
    Your browser does not support the HTML5 Audio element.
</audio>
```

The 'ogg' track in the above example works in Firefox, Opera and Chrome, while the same track in the 'mp3' format is added to make the audio work in Internet Explorer and Safari.

Linking Audio Files

You can make links to your audio files and play it by ticking on them.

Let's try out the following example to understand how this basically works:

Example
Try this code »

```
<a href="media/sea.mp3">Track 1</a>
<a href="media/wind.mp3">Track 2</a>
```

Using the object Element

The `<object>` element is used to embed different kinds of media files into an HTML document. Initially, this element was used to insert ActiveX controls, but according to the specification, an object can be any media object such as audio, video, PDF files, Flash animations or even images.

The following example code embeds a simple audio file into a web page.

Example
Try this code »

```
<object data="media/sea.mp3"></object>
<object data="media/sea.ogg"></object>
```

Warning: The `<object>` element is not supported widely and very much depends on the type of the object that's being embedded. Other methods like HTML5 `<audio>` element or third-party HTML5 audio players could be a better choice in many cases.

Using the embed Element

The `<embed>` element is used to embed multimedia content into an HTML document.

The following code fragment embeds audio files into a web page.

Example
Try this code »

```
<embed src="media/wind.mp3">
<embed src="media/wind.ogg">
```

Warning: However the `<embed>` element is very well supported in current browsers and defined as standard in HTML5, but your audio might not played due to lack of browser support for that file format or unavailability of plugins.

HTML5 Video

Embedding Video in HTML Document

Inserting video onto a web page was not relatively easy, because web browsers did not have a uniform standard for defining embedded media files like video.

In this chapter we'll demonstrates some of the many ways of adding videos on web pages, from the latest HTML5 `<video>` element to the popular YouTube videos.

Using the HTML5 video Element

The newly introduced HTML5 `<video>` element provides a standard way to embed video in web pages. However, the video element is relatively new, but it works in most of the modern web browsers.

The following example simply inserts a video into the HTML document, using the browser default set of controls, with one source defined by the `src` attribute.

Example
Try this code »

```
<video controls="controls" src="media/shuttle.mp4">
    Your browser does not support the HTML5 Video element.
</video>
```

A video, using the browser default set of controls, with alternative sources.

Example
Try this code »

```
<video controls="controls">
    <source src="media/shuttle.mp4" type="video/mp4">
    <source src="media/shuttle.ogv" type="video/ogg">
    Your browser does not support the HTML5 Video element.
</video>
```

Using the object Element

The `<object>` element is used to embed different kinds of media files into an HTML document. Initially, this element was used to insert ActiveX controls, but according to the specification, an object can be any media object such as video, audio, PDF files, Flash animations or even images.

The following code fragment embeds a Flash video into a web page.

Example
Try this code »

```
<object data="media/blur.swf" width="400px" height="200px"></object>
```

Only browsers or applications that support Flash will play this video.

Warning: The `<object>` element is not supported widely and very much depends on the type of the object that's being embedded. Other methods could be a better choice in many cases. iPad and iPhone device cannot display Flash videos.

Using the embed Element

The `<embed>` element is used to embed multimedia content into an HTML document.

The following code fragment embeds a Flash video into a web page.

Example
Try this code »

```
<embed src="media/blur.swf" width="400px" height="200px">
```

Warning: However, the `<embed>` element is very well supported in current web browsers and it is also defined as standard in HTML5, but your video might not played due to lack of browser support for Flash or unavailability of plugins.

Embedding the YouTube Videos

This is the easiest and popular way to embed videos files in the web pages. Just upload the video on YouTube and insert HTML code to display that video in your web page.

Here's a live example followed by the explanation of whole process:

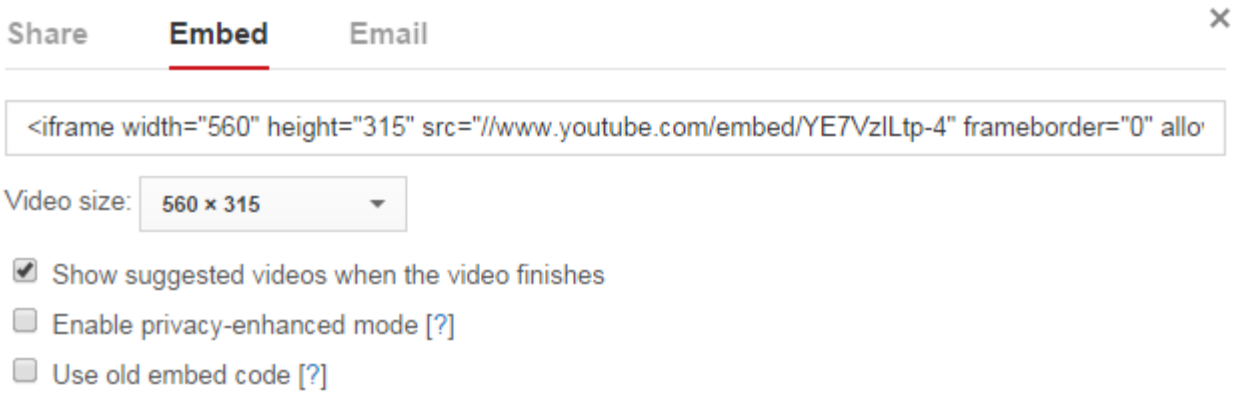
Step 1: Upload video

Go to YouTube upload video page and follow the instructions to upload your video.

Step 2: Creating the HTML Code to embed the video

When you open your uploaded video in YouTube you will see something like the following figure at the bottom of the video. Browse and open your uploaded video in YouTube. Now look for the share button which is located just below the video as shown in the figure.

When you click the share button, a share panel will open displaying some more buttons. Now click on the Embed button, it will generate the HTML code to directly embed the video into the web pages. Just copy and paste that code into your HTML document where you want to display the video and you're all set. By default video embedded inside an iframe.



You can further customize this embed code such as changing the video size by selecting the customization option given just below the embed-code input box.

The following example simply embeds a video from the YouTube. Let's try it out:

Example

Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>YouTube Video</title>
</head>
<body>
    <iframe width="560" height="315"
src="//www.youtube.com/embed/YE7VzlLtp-4" frameborder="0"
allowfullscreen></iframe>
</body>
</html>
```

HTML5 Web Storage. What is Web Storage?

The HTML5's web storage feature lets you store some information locally on the user's computer, similar to cookies, but it is faster and much better than cookies. However, web storage is no more secure than cookies. Please check out the tutorial on PHP cookies to learn more about cookies.

The information stored in the web storage isn't sent to the web server as opposed to the cookies where data sent to the server with every request. Also, where cookies let you store a small amount of data (nearly 4KB), the web storage allows you to store up to 5MB of data.

There are two types of web storage, which differ in scope and lifetime:

- **Local storage** — The local storage uses the `localStorage` object to store data for your entire website on a *permanent basis*. That means the stored local data will be available on the next day, the next week, or the next year unless you remove it.
- **Session storage** — The session storage uses the `sessionStorage` object to store data on a *temporary basis*, for a single browser window or tab. The data disappears when session ends i.e. when the user closes that browser window or tab.

Tip: The HTML5's web storage feature is supported in all major modern web browsers like Firefox, Chrome, Opera, Safari and Internet Explorer 8 and above.

HTML5 Application Cache

What is Application Cache?

Typically most web-based applications will work only if you're online. But HTML5 introduces an application cache mechanism that allows the browser to automatically save the HTML file and all the other resources that needs to display it properly on the local machine, so that the browser can still access the web page and its resources without an internet connection.

Here are some advantages of using the HTML5 application cache feature:

- **Offline browsing** — Users can use the application even when they're offline or there are unexpected disruptions in the network connection.
- **Improve performance** — Cached resources load directly from the user's machine rather than the remote server hence web pages load faster and performing better.
- **Reduce HTTP request and server load** — The browser will only have to download the updated/changed resources from the remote server that minimize the HTTP requests and saves precious bandwidth as well as reduce the load on the web server.

Tip: The HTML5's application cache feature is supported in all major modern web browsers like Firefox, Chrome, Opera, Safari and Internet Explorer 10 and above.

Caching Files with a Manifest

To cache the files for offline uses, you need to complete the following steps:

HTML5 Web Workers

What is Web Worker?

If you try to do intensive task with JavaScript that is time-consuming and require hefty calculations browser will freeze up the web page and prevent the user from doing anything until the job is completed. It happens because JavaScript code always runs in the foreground.

HTML5 introduces a new technology called *web worker* that is specifically designed to do background work independently of other user-interface scripts, without affecting the performance of the page. Unlike normal JavaScript operations, web worker doesn't interrupt the user and the web page remains responsive because they are running the tasks in the background.

Tip: The HTML5's web worker feature is supported in all major modern web browsers like Firefox, Chrome, Opera, Safari and Internet Explorer 10 and above.

Create a Web Worker File

The simplest use of web workers is for performing a time-consuming task. So here we are going to create a simple JavaScript task that counts from zero to 100,000.

Let's create an external JavaScript file named "worker.js" and type the following code.

Example
Download

```
var i = 0;
function countNumbers() {
    if(i < 100000) {
        i = i + 1;
        postMessage(i);
    }

    // Wait for sometime before running this script again
    setTimeout("countNumbers()", 500);
}
```

```
countNumbers();
```

Note: Web workers have no access to the DOM. That means you can't access any DOM elements in the JavaScript code that you intend to run using web workers.

Tip: The worker object's `postMessage()` method is used to send a message (like the numbers in the example above) back to the web page from the web worker file.

Doing Work in the Background with Web Worker

Now that we have created our web worker file. In this section we are going to initiate the web worker from an HTML document that runs the code inside the file named "worker.js" in the background and progressively displays the result on the web page. Let's see how it works:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Using Web Worker</title>
<script>
    if(window.Worker) {
        // Create a new web worker
        var worker = new Worker("worker.js");

        // Fire onMessage event handler
        worker.onmessage = function(event) {
            document.getElementById("result").innerHTML = event.data;
        };
    } else {
        alert("Sorry, your browser do not support web worker.");
    }
</script>
</head>
<body>
    <div id="result">
        <!--Received messages will be inserted here-->
    </div>
</body>
</html>
```

Example explained:

The JavaScript code in the above example has the following meaning:

- The statement **var worker = new Worker("worker.js");** creates a new web worker object, which is used to communicate with the web worker.

- When the worker posts a message, it fires the **onmessage** event handler (*line no-14*) that allows the code to receive messages from the web worker.
- The **event.data** element contains the message sent from the web worker.

Note: The code that a worker runs is always stored in a separate JavaScript file. This is to prevent web developer from writing the web worker code that attempts to use global variables or directly access the elements on the web page.

Terminate a Web Worker

So far you have learnt how to create worker and start receiving messages. However, you can also terminate a running worker in the middle of the execution.

The following example will show you how to start and stop worker from a web page through clicking the HTML buttons. It utilizes the same JavaScript file 'worker.js' what we have used in the previous example to count the numbers from zero to 100000. Let's try it out:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Start/Stop Web Worker</title>
<script>
    // Set up global variable
    var worker;

    function startWorker() {
        // Initialize web worker
        worker = new Worker("worker.js");

        // Run update function, when we get a message from worker
        worker.onmessage = update;

        // Tell worker to get started
        worker.postMessage("start");
    }

    function update(event) {
        // Update the page with current message from worker
        document.getElementById("result").innerHTML = event.data;
    }

    function stopWorker() {
        // Stop the worker
        worker.terminate();
```

```
    }
</script>
</head>
<body>
    <h1>Web Worker Demo</h1>
    <button onclick="startWorker();" type="button">Start web
worker</button>
    <button type="button" onclick="stopWorker();">Stop web worker</button>
    <div id="result">
        <!--Received messages will be inserted here-->
    </div>
</body>
</html>
```

Tip: Use the web workers for performing only heavy-weight JavaScript tasks that do not interrupt the user-interface scripts (i.e. scripts that respond to clicks or other user interactions). It's not recommended to use web workers for short tasks.

HTML5 Server-Sent Events

What is Server-Sent Event?

HTML5 server-sent event is a new way for the web pages to communicating with the web server. It is also possible with the XMLHttpRequest object that lets your JavaScript code make a request to the web server, but it's a one-for-one exchange — that means once the web server provides its response, the communication is over. XMLHttpRequest object is the core of all Ajax operations.

However, there are some situations where web pages require a longer-term connection to the web server. A typical example is stock quotes on finance websites where price updated automatically. Another example is a news ticker running on various media websites.

You can create such things with the HTML5 server-sent events. It allows a web page to hold an open connection to a web server so that the web server can send a new response automatically at any time, there's no need to reconnect, and run the same server script from scratch over and over again.

Note: Server-Sent Events (SSE) are unidirectional that means data are delivered in one direction from the server to client. A client typically is a web browser.

Tip: The HTML5's server-sent events feature is supported in all major modern web browsers like Firefox, Chrome, Safari and Opera except Internet Explorer.

Sending Messages with a Server Script

Let's create a PHP file named "server_time.php" and type the following script into it. It will simply report the current time of the web server's built-in clock in regular intervals. We will retrieve this time and update the web page accordingly later in this tutorial.

Example
Download

```php
<?php
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");

// Get the current time on server
$currentTime = date("h:i:s", time());

// Send it in a message
echo "data: " . $currentTime . "\n\n";
flush();
?>
```

The first two line of the PHP script sets two important headers. First, it sets the MIME type to `text/event-stream`, which is required by the server-side event standard. The second line tells the web server to turn off caching otherwise the output of your script may be cached.

Every message send through HTML5 server-sent events must start with the text `data:` followed by the actual message text and the new line character sequence (`\n\n`).

And finally, we have used the PHP `flush()` function to make sure that the data is sent right away, rather than buffered until the PHP code is complete.

Processing Messages in a Web Page

The `EventSource` object is used to receive server-sent event messages.

Now let's create an HTML document named "demo_sse.html" and place it in the same project directory where the "server_time.php" file is located. This HTML document simply receives the current time reported by the web server and display it to the user.

Download

```html
<!DOCTYPE html>
<html lang="en">
<head>
<title>Using Server-Sent Events</title>
<script>
    window.onload = function() {
        var source = new EventSource("server_time.php");
        source.onmessage = function(event) {
```

```
            document.getElementById("result").innerHTML += "New time
received from web server: " + event.data + "<br>";
        };
    };
</script>
</head>
<body>
    <div id="result">
        <!--Server response will be inserted here-->
    </div>
</body>
</html>
```

HTML5 Geolocation

What is Geolocation?

The HTML5 geolocation feature lets you find out the geographic coordinates (latitude and longitude numbers) of the current location of your website's visitor.

This feature is helpful for providing better browsing experience to the site visitor. For example, you can return the search results that are physically close to the user's location.

Finding a Visitor's Coordinates

Getting the position information of the site visitor using the HTML5 geolocation API is fairly simple. It utilizes the three methods that are packed into the `navigator.geolocation` object — `getCurrentPosition()`, `watchPosition()` and `clearWatch()`.

The following is a simple example of geolocation that displays your current position. But, first you need to agree to let the browser tell the web server about your position.

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Get Current Position</title>
<script>
    function showPosition() {
        if(navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(function(position) {
```

```
                var positionInfo = "Your current position is (" +
"Latitude: " + position.coords.latitude + ", " + "Longitude: " +
position.coords.longitude + ")";
                document.getElementById("result").innerHTML =
positionInfo;
            });
        } else {
            alert("Sorry, your browser does not support HTML5
geolocation.");
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

Note: The web browsers won't share the visitor location with a web page unless the visitor gives it explicit permission. The geolocation standard makes it an official rule to get user permission for every website that wants location data.

Dealing with Errors and Rejections

There may be a situation when a user does not want to share his location data with you. To deal with such situations, you can supply two functions when you call the `getCurrentLocation()` function.

The first function is called if your geolocation attempt is successful, while the second is called if your geolocation attempt ends in failure. Let's check out an example:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Handling Geolocation Errors</title>
<script>
    // Set up global variable
    var result;

    function showPosition() {
        // Store the element where the page displays the result
        result = document.getElementById("result");
```

```
        // If geolocation is available, try to get the visitor's position
        if(navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(successCallback,
errorCallback);
            result.innerHTML = "Getting the position information...";
        } else {
            alert("Sorry, your browser does not support HTML5
geolocation.");
        }
    };

    // Define callback function for successful attempt
    function successCallback(position) {
        result.innerHTML = "Your current position is (" + "Latitude: " +
position.coords.latitude + ", " + "Longitude: " +
position.coords.longitude + ")";
    }

    // Define callback function for failed attempt
    function errorCallback(error) {
        if(error.code == 1) {
            result.innerHTML = "You've decided not to share your position,
but it's OK. We won't ask you again.";
        } else if(error.code == 2) {
            result.innerHTML = "The network is down or the positioning
service can't be reached.";
        } else if(error.code == 3) {
            result.innerHTML = "The attempt timed out before it could get
the location data.";
        } else {
            result.innerHTML = "Geolocation failed due to unknown error.";
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

Showing Location on Google Map

You can do very interesting things with geolocation data, like showing the user location on Google map. The following example will show your current location on Google map based the latitude and longitude data retrieved through the HTML5 geolocation feature.

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Using the Google Maps</title>
<script>
    function showPosition() {
        navigator.geolocation.getCurrentPosition(showMap);
    }

    function showMap(position) {
        // Get location data
        var latlong = position.coords.latitude + "," +
position.coords.longitude;

        // Set Google map source url
        var mapLink =
"https://maps.googleapis.com/maps/api/staticmap?center="+latlong+"&zoom=16
&size=400x300&output=embed";

        // Create and insert Google map
        document.getElementById("embedMap").innerHTML = "<img alt='Map
Holder' src='"+ mapLink +"'>";
    }
</script>
</head>
<body>
    <button type="button" onclick="showPosition();">Show My Position on
Google Map</button>
    <div id="embedMap">
        <!--Google map will be embedded here-->
    </div>
</body>
</html>
```

The above example will simply show the location on the Google map using a static image. However, you can also create interactive Google maps with dragging, zoom in/out and other features that you have come across in your real life. Let's take a look at the following example:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Using the Google Maps</title>
<script src="https://maps.google.com/maps/api/js?sensor=false"></script>
```

```
<script>
function showPosition() {
    if(navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showMap, showError);
    } else {
        alert("Sorry, your browser does not support HTML5 geolocation.");
    }
}

// Define callback function for successful attempt
function showMap(position) {
    // Get location data
    lat = position.coords.latitude;
    long = position.coords.longitude;
    var latlong = new google.maps.LatLng(lat, long);

    var myOptions = {
        center: latlong,
        zoom: 16,
        mapTypeControl: true,
        navigationControlOptions: {
            style:google.maps.NavigationControlStyle.SMALL
        }
    }

    var map = new google.maps.Map(document.getElementById("embedMap"),
myOptions);
    var marker = new google.maps.Marker({ position:latlong, map:map,
title:"You are here!" });
}

// Define callback function for failed attempt
function showError(error) {
    if(error.code == 1) {
        result.innerHTML = "You've decided not to share your position, but
it's OK. We won't ask you again.";
    } else if(error.code == 2) {
        result.innerHTML = "The network is down or the positioning service
can't be reached.";
    } else if(error.code == 3) {
        result.innerHTML = "The attempt timed out before it could get the
location data.";
    } else {
        result.innerHTML = "Geolocation failed due to unknown error.";
    }
}
</script>
</head>
<body>
    <button type="button" onclick="showPosition();">Show My Position on
Google Map</button>
    <div id="embedMap" style="width: 400px; height: 300px;">
        <!--Google map will be embedded here-->
```

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

Please check out the following URL to learn more about the Google Maps Javascript API: https://developers.google.com/maps/documentation/javascript/reference.

Monitoring the Visitor's Movement

All the examples we've used so far have relied on the `getCurrentPosition()` method. However, the geolocation object has another method `watchPosition()` that allow you to track the visitor's movement by returning the updated position as the location changes.

The `watchPosition()` has the same input parameters as `getCurrentPosition()`.
However, `watchPosition()` may trigger the success function multiple times — when it gets the location for the first time, and again, whenever it detects a new position. Let's see how this works:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Watching Position</title>
<script>
    // Set global variable
    var watchID;

    function showPosition() {
        if(navigator.geolocation) {
            watchID =
navigator.geolocation.watchPosition(successCallback);
        } else {
            alert("Sorry, your browser does not support HTML5
geolocation.");
        }
    }

    function successCallback(position) {
        toggleWatchBtn.innerHTML = "Stop Watching";

        // Check position has been changed or not before doing anything
        if(prevLat != position.coords.latitude || prevLong !=
position.coords.longitude){

            // Set previous location
            var prevLat = position.coords.latitude;
            var prevLong = position.coords.longitude;
```

```
            // Get current position
            var positionInfo = "Your current position is (" + "Latitude: "
+ position.coords.latitude + ", " + "Longitude: " +
position.coords.longitude + ")";
            document.getElementById("result").innerHTML = positionInfo;

        }

    }

    function startWatch() {
        var result = document.getElementById("result");

        var toggleWatchBtn = document.getElementById("toggleWatchBtn");

        toggleWatchBtn.onclick = function() {
            if(watchID) {
                toggleWatchBtn.innerHTML = "Start Watching";
                navigator.geolocation.clearWatch(watchID);
                watchID = false;
            } else {
                toggleWatchBtn.innerHTML = "Aquiring Geo Location...";
                showPosition();
            }
        }
    }

    // Initialise the whole system (above)
    window.onload = startWatch;
</script>
</head>
<body>
    <button type="button" id="toggleWatchBtn">Start Watching</button>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
</body>
</html>
```