**HTML Tags:** Tags are the starting and ending parts of an HTML element. They begin with < symbol and end with > symbol. Whatever written inside < and > are called tags.

**HTML elements:** Elements enclose the contents in between the tags. They consist of some kind of structure or expression. It generally consists of a start tag, content and an end tag.

List of block level elements

* p
* h1, h2, h3, h4, h5, h6
* ol, ul
* pre
* address
* blockquote
* dl
* div
* fieldset
* form
* hr
* noscript
* table

HTML inline elements

1. **HTML inline level elements** can appear in the body of an HTML page.

2. It can contain data and other**inline**elements.

3. By default, **inline elements** do not begin on new lines.

4. **inline elements** create shorter structures (than block level elements).

**List of inline elements**

* b, big, i, small, tt
* abbr, acronym, cite, code, dfn, em, kbd, strong, samp, var
* a, bdo, br, img, map, object, q, script, span, sub, sup
* button, input, label, select, textarea

| **HTML Tags** | **HTML Elements** | **HTML Attributes** |
| --- | --- | --- |
| HTML tags are used to hold the HTML element. | HTML element holds the content. | HTML attributes are used to describe the characteristic of an HTML element in detail. |
| HTML tag starts with < and ends with > | Whatever written within a HTML tag are HTML elements. | HTML attributes are found only in the starting tag. |
| HTML tags are almost like keywords where every single tag has unique meaning. | HTML elements specifies the general content. | HTML attributes specify various additional properties to the existing HTML element. |

**What are Tags and Attributes?**

Tags and attributes are the basis of HTML.

They work together but perform different functions – it is worth investing 2 minutes in differentiating the two.

**What Are HTML Tags?**

Tags are used to mark up the start of an HTML element and they are usually enclosed in angle brackets. An example of a tag is: <h1>.

Most tags must be opened <h1> and closed </h1> in order to function.

**What are HTML Attributes?**

Attributes contain additional pieces of information. Attributes take the form of an opening tag and additional info is placed inside.

* An example of an attribute is:

<img src="mydog.jpg" alt="A photo of my dog." ></img>

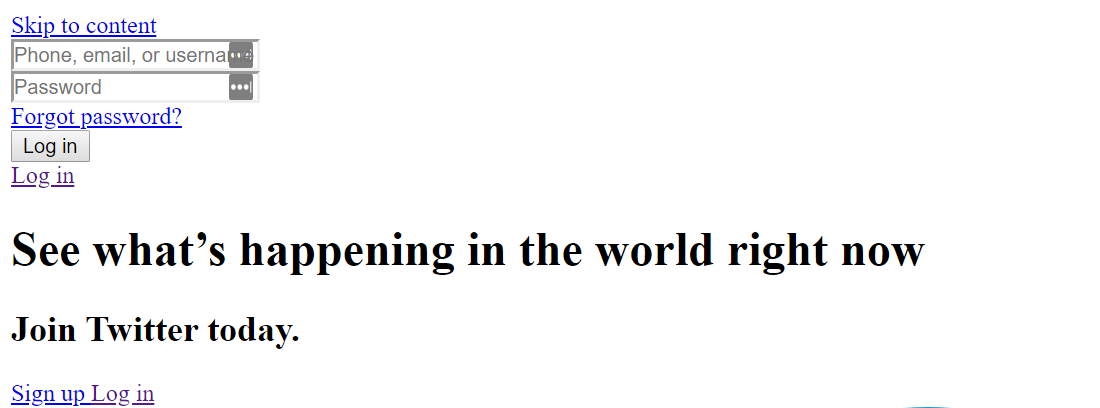
In this instance, the image source (src) and the alt text (alt) are attributes of the <img> tag.

**Golden Rules To Remember**

* The vast majority of tags must be opened (<tag>) and closed (</tag>) with the element information such as a title or text resting between the tags.
* When using multiple tags, the tags must be closed in the order in which they were opened. For example:

<https://www.w3resource.com/html/HTML-block-level-and-inline-elements.php>

<https://www.geeksforgeeks.org/tags-vs-elements-vs-attributes-in-html/>

Cascading Style Sheets (CSS) is a markup language responsible for how your web pages will look like. It controls the colors, fonts, and layouts of your website elements.

There are three ways you can use to implement CSS: internal, external, and inline styles. Let’s break them down.

### Internal CSS

Internal or embedded CSS requires you to add **<style>** tag in the **<head>** section of your HTML document.

This CSS style is an effective method of styling a single page. However, using this style for multiple pages is time-consuming as you need to put CSS rules to every page of your website.

Here’s how you can use internal CSS:

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-color: blue;

}

h1 {

color: red;

padding: 60px;

}

</style>

</head>

<body>

<h1>Hostinger Tutorials</h1>

<p>This is our paragraph.</p>

</body>

</html>

1. Open your HTML page and locate **<head>** opening tag.
2. Put the following code right after the **<head>** tag

### External CSS

With external CSS, you’ll link your web pages to an external **.css** file, which can be created by any text editor in your device (e.g., [**Notepad++**](https://notepad-plus-plus.org/)).

This CSS type is a more efficient method, especially for styling a large website. By editing one **.css** file, you can change your entire site at once.

Follow these steps to use external CSS:

1. Create a new **.css** file with the text editor, and add the style rules. For example:
2. In the **<head>** section of your HTML sheet, add a reference to your external **.css** file right after **<title>** tag:

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

### Inline CSS

Inline CSS is used to style a specific HTML element. For this CSS style, you’ll only need to add the **style** attribute to each HTML tag, without using selectors.

This CSS type is not really recommended, as each HTML tag needs to be styled individually. Managing your website may become too hard if you only use inline CSS.

However, inline CSS in HTML can be useful in some situations. For example, in cases where you don’t have access to CSS files or need to apply styles for a single element only.

Let’s take a look at an example. Here, we add an inline CSS to the **<p>** and **<h1>** tag:

<!DOCTYPE html>

<html>

<body style="background-color:black;">

<h1 style="color:white;padding:30px;">Hostinger Tutorials</h1>

<p style="color:white;">Something usefull here.</p>

</body>

</html>

## DOM stands for Document Object Model, a representation of an HTML document in nodes and objects. Browsers expose an API that you can use to interact with the DOM. This is how modern JavaScript frameworks work - they use the DOM API to tell the browser what to display on the page

## 

The DOM is the browser’s internal representation of a web page. When the browser retrieves your HTML from your server, the parser analyzes the structure of your code and creates a model of it. Based on this model, the browser then renders the page on the screen.

Browsers expose an API that you can use to interact with the DOM. This is how modern JavaScript frameworks work - they use the DOM API to tell the browser what to display on the page.

In Single Page Applications, the DOM continuously changes to reflect what appears on the screen, and as a developer you can inspect it using the [Browser Developer Tools](https://flaviocopes.com/browser-dev-tools/).

The DOM is language-agnostic, and the de-facto standard to access the DOM is by using [JavaScript](https://flaviocopes.com/javascript/), since it’s the only language that browsers can run.

## The Window object

The window object represents the window that contains the DOM document.

window.document points to the document object loaded in the window.

Properties and methods of this object can be called without referencing window explicitly, because it represents the global object. So, the previous property window.document is usually called just document.

### Properties

Here is a list of useful properties you will likely reference a lot:

* console points to the browser debugging console. Useful to print error messages or logging, using console.log, console.error and other tools (see the [Browser DevTools](https://flaviocopes.com/browser-dev-tools/) article)
* document as already said, points to the document object, key to the DOM interactions you will perform
* history gives access to the [**History API**](https://flaviocopes.com/history-api/)
* location gives access to the [Location interface](https://developer.mozilla.org/en-US/docs/Web/API/Location), from which you can determine the URL, the protocol, the hash and other useful information.
* localStorage is a reference to the [Web Storage API](https://flaviocopes.com/web-storage-api/) localStorage object
* sessionStorage is a reference to the Web Storage API sessionStorage object

### Methods

The window object also exposes useful methods:

* alert(): which you can use to display alert dialogs
* postMessage(): used by the [Channel Messaging API](https://flaviocopes.com/channel-messaging-api/)
* [requestAnimationFrame()](https://flaviocopes.com/requestanimationframe/): used to perform animations in a way that’s both performant and easy on the CPU
* setInterval(): call a function every n milliseconds, until the interval is cleared with clearInterval()
* clearInterval(): clears an interval created with setInterval()
* [setTimeout()](https://flaviocopes.com/javascript-timers/): execute a function after ‘n’ milliseconds
* [setImmediate()](https://flaviocopes.com/javascript-timers/): execute a function as soon as the browser is ready
* addEventListener(): add an event listener to the document
* removeEventListener(): remove an event listener from the document

See the full reference of all the properties and methods of the window object at <https://developer.mozilla.org/en-US/docs/Web/API/Window>