**Contents**

[Network Introduction 11](#_Toc94014162)

[Socket programming 11](#_Toc94014163)

[Client programming 11](#_Toc94014164)

[Building web servers 11](#_Toc94014165)

[Web Scrapping 11](#_Toc94014166)

[Web Frame works 11](#_Toc94014167)

[Difference between IP address and Port Number 12](#_Toc94014168)

[Difference between IP address and Port Number: 12](#_Toc94014169)

[Host name 13](#_Toc94014170)

[MAC Address (Media Access Control address) 13](#_Toc94014171)

[Setup 14](#_Toc94014172)

[Configure a local testing server 14](#_Toc94014173)

[Local files vs. remote files 14](#_Toc94014174)

[The problem with testing local files 14](#_Toc94014175)

[Running a simple local HTTP server 14](#_Toc94014176)

[Running server-side languages locally 16](#_Toc94014177)

[Accessing the server over a Network 16](#_Toc94014178)

[First Web Server Program 18](#_Toc94014179)

[Similar Example, with more details 18](#_Toc94014180)

[HTTP Status Codes 21](#_Toc94014181)

[Information responses 21](#_Toc94014182)

[Successful responses 21](#_Toc94014183)

[Redirection messages 22](#_Toc94014184)

[Client error responses 22](#_Toc94014185)

[Server error responses 23](#_Toc94014186)

[HTTP Headers 24](#_Toc94014187)

[HTTP Methods 25](#_Toc94014188)

[GET vs POST 25](#_Toc94014189)

[HTML 5 27](#_Toc94014190)

[What is an HTML Element? 28](#_Toc94014191)

[Web Browsers 28](#_Toc94014192)

[HTML Page Structure 28](#_Toc94014193)

[HTML History 29](#_Toc94014194)

[HTML Basic Examples 30](#_Toc94014195)

[HTML Documents 30](#_Toc94014196)

[The <!DOCTYPE> Declaration 30](#_Toc94014197)

[HTML Headings 30](#_Toc94014198)

[HTML Paragraphs 31](#_Toc94014199)

[HTML Links 31](#_Toc94014200)

[HTML Images 31](#_Toc94014201)

[HTML Elements 31](#_Toc94014202)

[Nested HTML Elements 32](#_Toc94014203)

[Never Skip the End Tag 33](#_Toc94014204)

[Empty HTML Elements 33](#_Toc94014205)

[HTML is Not Case Sensitive 33](#_Toc94014206)

[HTML Attributes 34](#_Toc94014207)

[The href Attribute 34](#_Toc94014208)

[The src Attribute 34](#_Toc94014209)

[The width and height Attributes 34](#_Toc94014210)

[The alt Attribute 35](#_Toc94014211)

[The style Attribute 35](#_Toc94014212)

[The lang Attribute 35](#_Toc94014213)

[The title Attribute 36](#_Toc94014214)

[Always Use Lowercase Attributes 36](#_Toc94014215)

[Always Quote Attribute Values 36](#_Toc94014216)

[Single or Double Quotes? 36](#_Toc94014217)

[HTML Headings 38](#_Toc94014218)

[HTML Headings 38](#_Toc94014219)

[Headings Are Important 38](#_Toc94014220)

[Bigger Headings 39](#_Toc94014221)

[HTML Paragraphs 40](#_Toc94014222)

[HTML Display 40](#_Toc94014223)

[HTML Horizontal Rules 40](#_Toc94014224)

[HTML Line Breaks 41](#_Toc94014225)

[The Poem Problem 41](#_Toc94014226)

[Solution - The HTML <pre> Element 41](#_Toc94014227)

[HTML Styles 43](#_Toc94014228)

[The HTML Style Attribute 43](#_Toc94014229)

[Background Color 43](#_Toc94014230)

[Text Color 43](#_Toc94014231)

[Fonts 44](#_Toc94014232)

[Text Size 44](#_Toc94014233)

[Text Alignment 44](#_Toc94014234)

[HTML Text Formatting 45](#_Toc94014235)

[HTML Formatting Elements 45](#_Toc94014236)

[HTML <b> and <strong> Elements 45](#_Toc94014237)

[HTML <i> and <em> Elements 45](#_Toc94014238)

[HTML <small> Element 46](#_Toc94014239)

[HTML <mark> Element 46](#_Toc94014240)

[HTML <del> Element 46](#_Toc94014241)

[HTML <ins> Element 46](#_Toc94014242)

[HTML <sub> Element 47](#_Toc94014243)

[HTML <sup> Element 47](#_Toc94014244)

[HTML Text Formatting Elements 47](#_Toc94014245)

[HTML Colors 48](#_Toc94014246)

[Color Names 48](#_Toc94014247)

[Background Color 48](#_Toc94014248)

[Text Color 48](#_Toc94014249)

[Border Color 49](#_Toc94014250)

[Color Values 49](#_Toc94014251)

[HTML Styles - CSS 51](#_Toc94014252)

[What is CSS? 51](#_Toc94014253)

[Using CSS 51](#_Toc94014254)

[Inline CSS 51](#_Toc94014255)

[Internal CSS 51](#_Toc94014256)

[External CSS 52](#_Toc94014257)

[CSS Syntax 53](#_Toc94014258)

[CSS Colors, Fonts and Sizes 53](#_Toc94014259)

[CSS Border 54](#_Toc94014260)

[CSS Padding 54](#_Toc94014261)

[CSS Margin 55](#_Toc94014262)

[Link to External CSS 55](#_Toc94014263)

[HTML Links – Hyperlinks 56](#_Toc94014264)

[HTML Links - Syntax 56](#_Toc94014265)

[HTML Links - The target Attribute 56](#_Toc94014266)

[Absolute URLs vs. Relative URLs 57](#_Toc94014267)

[HTML Links - Use an Image as a Link 57](#_Toc94014268)

[Link to an Email Address 57](#_Toc94014269)

[Button as a Link 57](#_Toc94014270)

[Link Titles 58](#_Toc94014271)

[More on Absolute URLs and Relative URLs 58](#_Toc94014272)

[HTML Links – Different Colors 59](#_Toc94014273)

[Link Buttons 59](#_Toc94014274)

[HTML Images 61](#_Toc94014275)

[HTML Images Syntax 61](#_Toc94014276)

[The src Attribute 61](#_Toc94014277)

[The alt Attribute 61](#_Toc94014278)

[Image Size - Width and Height 62](#_Toc94014279)

[Width and Height, or Style? 62](#_Toc94014280)

[Images in Another Folder 63](#_Toc94014281)

[Images on Another Server/Website 63](#_Toc94014282)

[Animated Images 63](#_Toc94014283)

[Image as a Link 63](#_Toc94014284)

[Image Floating 64](#_Toc94014285)

[Common Image Formats 64](#_Toc94014286)

[HTML Background Images 65](#_Toc94014287)

[Background Image on a Page 66](#_Toc94014288)

[Background Repeat 67](#_Toc94014289)

[Background Cover 68](#_Toc94014290)

[Background Stretch 69](#_Toc94014291)

[HTML Tables 71](#_Toc94014292)

[Define an HTML Table 71](#_Toc94014293)

[Table Cells 71](#_Toc94014294)

[Table Rows 72](#_Toc94014295)

[Table Headers 72](#_Toc94014296)

[HTML Table Tags 73](#_Toc94014297)

[HTML Table Borders 75](#_Toc94014298)

[How To Add a Border 75](#_Toc94014299)

[Collapsed Table Borders 76](#_Toc94014300)

[Style Table Borders 77](#_Toc94014301)

[Round Table Borders 78](#_Toc94014302)

[Dotted Table Borders 81](#_Toc94014303)

[Border Color 82](#_Toc94014304)

[HTML Table Sizes 84](#_Toc94014305)

[HTML Table Width 84](#_Toc94014306)

[HTML Table Column Width 84](#_Toc94014307)

[HTML Table Row Height 85](#_Toc94014308)

[HTML Table Headers 87](#_Toc94014309)

[HTML Table Headers 87](#_Toc94014310)

[Vertical Table Headers 87](#_Toc94014311)

[Align Table Headers 88](#_Toc94014312)

[Header for Multiple Columns 88](#_Toc94014313)

[Table Caption 89](#_Toc94014314)

[HTML Table Padding & Spacing 91](#_Toc94014315)

[HTML Table - Cell Padding 91](#_Toc94014316)

[HTML Table - Cell Spacing 93](#_Toc94014317)

[HTML Table Colspan & Rowspan 95](#_Toc94014318)

[HTML Table - Colspan 95](#_Toc94014319)

[HTML Table - Rowspan 96](#_Toc94014320)

[HTML Lists 98](#_Toc94014321)

[Unordered HTML List 98](#_Toc94014322)

[Ordered HTML List 98](#_Toc94014323)

[HTML Description Lists 99](#_Toc94014324)

[HTML List Tags 99](#_Toc94014325)

[HTML Block & Inline Elements 100](#_Toc94014326)

[Block-level Elements 100](#_Toc94014327)

[Inline Elements 100](#_Toc94014328)

[The <div> Element 101](#_Toc94014329)

[The <span> Element 101](#_Toc94014330)

[HTML Classes 102](#_Toc94014331)

[Using The class Attribute 102](#_Toc94014332)

[The Syntax for Class 103](#_Toc94014333)

[Multiple Classes 104](#_Toc94014334)

[Different Elements Can Share Same Class 104](#_Toc94014335)

[Use of the class Attribute in JavaScript 104](#_Toc94014336)

[HTML Id 106](#_Toc94014337)

[Using The id Attribute 106](#_Toc94014338)

[Difference Between Class and ID 106](#_Toc94014339)

[HTML Bookmarks with ID and Links 107](#_Toc94014340)

[Using The id Attribute in JavaScript 108](#_Toc94014341)

[HTML Forms 110](#_Toc94014342)

[The <form> Element 110](#_Toc94014343)

[The <input> Element 110](#_Toc94014344)

[Text Fields 111](#_Toc94014345)

[The <label> Element 112](#_Toc94014346)

[Radio Buttons 112](#_Toc94014347)

[Checkboxes 113](#_Toc94014348)

[The Submit Button 114](#_Toc94014349)

[The Name Attribute for <input> 115](#_Toc94014350)

[HTML Form Attributes 116](#_Toc94014351)

[The Action Attribute 116](#_Toc94014352)

[The Target Attribute 116](#_Toc94014353)

[The Method Attribute 117](#_Toc94014354)

[The Autocomplete Attribute 119](#_Toc94014355)

[The Novalidate Attribute 119](#_Toc94014356)

[HTML Form Elements 121](#_Toc94014357)

[The HTML <form> Elements 121](#_Toc94014358)

[The <input> Element 121](#_Toc94014359)

[The <label> Element 121](#_Toc94014360)

[The <select> Element 122](#_Toc94014361)

[Visible Values: 123](#_Toc94014362)

[Allow Multiple Selections: 123](#_Toc94014363)

[The <textarea> Element 124](#_Toc94014364)

[The <button> Element 125](#_Toc94014365)

[The <fieldset> and <legend> Elements 126](#_Toc94014366)

[The <datalist> Element 126](#_Toc94014367)

[The <output> Element 127](#_Toc94014368)

[HTML Input Types 129](#_Toc94014369)

[Input Type Text 129](#_Toc94014370)

[Input Type Password 129](#_Toc94014371)

[Input Type Submit 130](#_Toc94014372)

[Input Type Reset 130](#_Toc94014373)

[Input Type Radio 131](#_Toc94014374)

[Input Type Checkbox 131](#_Toc94014375)

[Input Type Button 132](#_Toc94014376)

[Input Type Color 132](#_Toc94014377)

[Input Type Date 132](#_Toc94014378)

[Input Type Datetime-local 133](#_Toc94014379)

[Input Type Email 133](#_Toc94014380)

[Input Type File 133](#_Toc94014381)

[Input Type Hidden 133](#_Toc94014382)

[Input Type Month 134](#_Toc94014383)

[Input Type Number 134](#_Toc94014384)

[Input Restrictions 134](#_Toc94014385)

[Input Type Range 135](#_Toc94014386)

[Input Type Search 135](#_Toc94014387)

[Input Type Tel 136](#_Toc94014388)

[Input Type Time 136](#_Toc94014389)

[Input Type Url 136](#_Toc94014390)

[Input Type Week 136](#_Toc94014391)

[HTML Input Attributes 138](#_Toc94014392)

[The value Attribute 138](#_Toc94014393)

[The readonly Attribute 138](#_Toc94014394)

[The disabled Attribute 138](#_Toc94014395)

[The size Attribute 139](#_Toc94014396)

[The maxlength Attribute 139](#_Toc94014397)

[The min and max Attributes 139](#_Toc94014398)

[The multiple Attribute 140](#_Toc94014399)

[The pattern Attribute 141](#_Toc94014400)

[The placeholder Attribute 141](#_Toc94014401)

[The required Attribute 142](#_Toc94014402)

[The step Attribute 142](#_Toc94014403)

[The autofocus Attribute 143](#_Toc94014404)

[The height and width Attributes 143](#_Toc94014405)

[The list Attribute 143](#_Toc94014406)

[The autocomplete Attribute 144](#_Toc94014407)

[HTML Input Form Attributes 146](#_Toc94014408)

[The form Attribute 146](#_Toc94014409)

[The formaction Attribute 146](#_Toc94014410)

[~~The formenctype Attribute~~ 147](#_Toc94014411)

[The formmethod Attribute 147](#_Toc94014412)

[The formtarget Attribute 148](#_Toc94014413)

[The formnovalidate Attribute 148](#_Toc94014414)

[The novalidate Attribute 149](#_Toc94014415)

[HTML Query String 150](#_Toc94014416)

[Query Strings/Parameters 150](#_Toc94014417)

[URL Encoding 152](#_Toc94014418)

[Sample PHP Code to Fetch Query String 152](#_Toc94014419)

[CSS3 154](#_Toc94014420)

[Selectors 154](#_Toc94014421)

[The CSS element Selector 154](#_Toc94014422)

[The CSS id Selector 154](#_Toc94014423)

[The CSS class Selector 155](#_Toc94014424)

[The CSS Universal Selector 156](#_Toc94014425)

[The CSS Grouping Selector 156](#_Toc94014426)

[All CSS Simple Selectors 157](#_Toc94014427)

[The CSS Box Model 158](#_Toc94014428)

[Width and Height of an Element 159](#_Toc94014429)

[CSS Outline 160](#_Toc94014430)

[CSS Outline Style 160](#_Toc94014431)

[Flask 163](#_Toc94014432)

[What is Web Framework? 163](#_Toc94014433)

[WSGI 163](#_Toc94014434)

[Werkzeug 163](#_Toc94014435)

[Jinja2 163](#_Toc94014436)

[What is Flask? 163](#_Toc94014437)

[Virtual Environment Setup (optional) 163](#_Toc94014438)

[Creating virtual environment in Linux 163](#_Toc94014439)

[Creating Python virtualenv in Windows 164](#_Toc94014440)

[Install Flask 165](#_Toc94014441)

[First Flask Application 165](#_Toc94014442)

[Debug mode 166](#_Toc94014443)

[Flask Routing 166](#_Toc94014444)

[Flask Variables 167](#_Toc94014445)

[Flask URL Binding 168](#_Toc94014446)

[Flask HTTP Methods 169](#_Toc94014447)

[Flask Templates 172](#_Toc94014448)

[Flask Static Files 175](#_Toc94014449)

[Flask Request Object 176](#_Toc94014450)

[Flask Sending Form Data to Template 176](#_Toc94014451)

[Flask Cookies 178](#_Toc94014452)

[Flask Sessions 181](#_Toc94014453)

[Flask Redirect & Errors 183](#_Toc94014454)

# Network Introduction

As python’s versatility as a programming language grown over the years, we find that python is very suitable in the world of network programming too. With growth in cloud computing , network programming has become even a more hot topic and python has a big role to play. Below are the few important reasons for python’s use as a preferred language for network Programming.

## Socket programming

Sockets are the links through which the client and servers communicate with each other. For example when a browser is opened a socket is automatically created to connect with the server. Python has a socket module which an be used to implement various socket functionalities like binding an address or starting a listener port. Socket programming is fundamental to computer networking and python handles it well.

## Client programming

The client is the computer which requests for information and waits for the response. Python programs can be written to validate many client-side actions like parsing a URL, sending parameters with the URL while submitting a request, connect to a alternate URL if access to one URL becomes unsuccessful etc. These programs are run in the client programs and handle all the communication needs with the server even without using a browser. For example – you can provide an URL to the python program for downloading a file and it will get done by the program itself without taking help from the browser program.

## Building web servers

It is possible to create simple web servers which are good enough to run websites using python as a programming language. Python already has some inbuilt web servers which can be tweaked to achieve some additional functionalities needed.

The **SimpleHTTPServer** module provides the functionalities of a web server out of the box and you can start running it from the local python installation. In python 3 it is named as **http.serverCherryPy** and **Tornado** are examples of webservers written in python which run as good as non python well known web servers like Apache or Ngnix.

## Web Scrapping

One of the important reasons python became famous is the its dominance among the languages used for scrapping the web. Its data structure and network access abilities makes it ideal for visiting webpages and download their data automatically. And if there is some API connectivity available for the target website, then python will handle it even more easily through its program structures.

## Web Frame works

Web Frame works makes application development easy and fast by offering pre-defined structures and modularity. The developer has to do minimal coding to leverage those existing libraries and customize a little to achieve the goal. **Django** and **Flask** are two famous ones which have seen much commercial use even though they are opensource.

## Difference between IP address and Port Number

1. [IP address](https://www.geeksforgeeks.org/introduction-of-classful-ip-addressing/)

An Internet Protocol address (IP address) is the logical address of our network hardware by which other devices identify it in a network. IP address stands for Internet Protocol address which is an unique number or a numerical representation that uniquely identifies a specific interface on the network. Each device that is connected to internet an IP address is assigned to it for its unique identification.

Addresses in IPv4 are 32-bits long example,

12.244.233.165

And addresses in IPv6 are 128-bits example,

2001:0db8:0000:0000:0000:ff00:0042:7879

Also known as the Logical Address, the IP Address is the network address of the system across the network.

To identify each device in the world-wide-web, the Internet Assigned Numbers Authority (IANA) assigns an IPV4 (Version 4) address as a unique identifier to each device on the Internet.   
The length of an IPv4 address is 32-bits, hence, we have 232 IP addresses available. The length of an IPv6 address is 128-bits.

Type “ipconfig” in the command prompt and press ‘Enter’, this gives us the IP address of the device.

1. [Port Number](https://www.geeksforgeeks.org/tcp-ip-ports-and-its-applications/)

Port number is the part of the addressing information used to identify the senders and receivers of messages in computer networking. Different port numbers are used to determine what protocol incoming traffic should be directed to. Port number identifies a specific process to which an Internet or other network message is to be forwarded when it arrives at a server. Ports are identified for each protocol and it is considered as a communication endpoint.

Ports are represented by 16-bit numbers. 0 to 1023 are restricted port numbers are as they are used by well-known protocol services. 1024 to 49151 are registered port numbers means it can be registered to specific protocols by software corporations and in last 49152 to 65536 are used as private ports means they can be used by anybody.

### Difference between IP address and Port Number:

|  |  |  |
| --- | --- | --- |
| **Serial No** | **IP address** | **Port Number** |
| 01. | Internet Protocol address (IP address) used to identify a host in network. | Port number is used to identify a processes/services on your system |
| 02. | IPv4 is of 32 bits (4 bytes) size and for IPv6 is 128 bits (16 bytes). | The Port number is 16 bits numbers. |
| 03. | IP address is the address of the layer-3 IP protocol. | Port number is the address of the layer-4 protocols. |
| 04. | IP address is provided by admin of system or network administrator. | Port number for application is provided by kernel of Operating System. |
| 05. | ipconfig command can be used to find IP address. | netstat command can be used to find Network Statistics Including Available TCP Ports. |
| 06. | IP address identify a host/computer on a computer network. | Port numbers are logical interfaces used by communication protocols. |
| 07. | 192.168.0.2, 172.16.0.2 are some of IP address examples. | 80 for HTTP, 123 for NTP, 67 and 68 for DHCP traffic, 22 for SSH etc. |

### Host name

Each device in the network is associated with a unique device name known as Hostname. Type “hostname” in the command prompt (Administrator Mode) and press ‘Enter’, this displays the hostname of your machine.

### MAC Address (Media Access Control address)

Also known as physical address, the MAC Address is the unique identifier of each host and is associated with its NIC (Network Interface Card).

A MAC address is assigned to the NIC at the time of manufacturing.

The length of the MAC address is: 12-nibble/ 6 bytes/ 48 bits.

*Type “ipconfig/all” in the command prompt and press ‘Enter’, this gives us the MAC address.*

Socket   
The unique combination of IP address and Port number together are termed as Socket.

# Setup

## Configure a local testing server

<https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server>

### [Local files vs. remote files](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server#local_files_vs._remote_files)

Throughout most of the learning area, we tell you to just open your examples directly in a browser — this can be done by double clicking the HTML file, dragging and dropping it into the browser window, or choosing File > Open... and navigating to the HTML file. There are many ways to achieve this.

If the web address path starts with file:// followed by the path to the file on your local hard drive, a local file is being used. In contrast, if you view one of our examples hosted on GitHub (or an example on some other remote server), the web address will start with http:// or https://, to show that the file has been received via HTTP.

### [The problem with testing local files](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server#the_problem_with_testing_local_files)

Some examples won't run if you open them as local files. This can be due to a variety of reasons, the most likely being:

* They feature asynchronous requests. Some browsers (including Chrome) will not run async requests (see [Fetching data from the server](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data)) if you just run the example from a local file. This is because of security restrictions (for more on web security, read [Website security](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Website_security)).
* They feature a server-side language. Server-side languages (such as PHP or Python) require a special server to interpret the code and deliver the results.
* They include other files. Browsers commonly treat requests to load resources using the file:// schema as cross-origin requests. So if you load a local file that includes other local files, this may trigger a [CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS) error.

### [Running a simple local HTTP server](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server#running_a_simple_local_http_server)

To get around the problem of async requests, we need to test such examples by running them through a local web server. One of the easiest ways to do this for our purposes is to use Python's http.server module.

Note: Older versions of Python (up to version 2.7) provided a similar module named SimpleHTTPServer. If you are using Python 2.x, you can follow this guide by replacing all uses of http.server with SimpleHTTPServer. However, we recommend you use the latest version of Python.

To do this:

1. Install Python. If you are using Linux or macOS, it should be available on your system already. If you are a Windows user, you can get an installer from the Python homepage and follow the instructions to install it:
   * Go to [python.org](https://www.python.org/)
   * Under the Download section, click the link for Python “3.xxx”.
   * At the bottom of the page, click the *Windows Installer* link to download the installer file.
   * When it has downloaded, run it.
   * On the first installer page, make sure you check the “Add Python 3.xxx to PATH” checkbox.
   * Click *Install*, then click *Close* when the installation has finished.
2. Open your command prompt (Windows) / terminal (macOS/ Linux). To check if Python is installed, enter the following command:

python -V

# If the above fails, try:

python3 -V

# Or, if the “py” command is available, try:

py -V

1. This should return a version number. If this is OK, navigate to the directory that your example is inside, using the cd command.

# include the directory name to enter it, for example

cd Desktop

# use two dots to jump up one directory level if you need to

cd ..

1. Enter the command to start up the server in that directory:

# If Python version returned above is 3.X

# On Windows, try “python -m http.server” or “py -3 -m http.server”

python3 -m http.server

# If Python version returned above is 2.X

python -m SimpleHTTPServer

1. By default, this will run the contents of the directory on a local web server, on port 8000. You can go to this server by going to the URL localhost:8000 in your web browser. Here you’ll see the contents of the directory listed — click the HTML file you want to run.

Note: If you already have something running on port 8000, you can choose another port by running the server command followed by an alternative port number, e.g. python3 -m http.server 7800 (Python 3.x) or python -m SimpleHTTPServer 7800 (Python 2.x). You can then access your content at localhost:7800.

### [Running server-side languages locally](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server#running_server-side_languages_locally)

Python’s http.server (or SimpleHTTPServer for Python 2) module is useful, but it is merely a static file server; it doesn’t know how to run code written in languages such as Python, PHP or JavaScript. To handle them, you’ll need something more — exactly what you’ll need depends on the server-side language you are trying to run. Here are a few examples:

* To run Python server-side code, you'll need to use a Python web framework. There are many popular Python web frameworks, such as Django (a [guide](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django) is available), [Flask](https://flask.palletsprojects.com/), and [Pyramid](https://trypyramid.com/).
* To run Node.js (JavaScript) server-side code, you'll need to use raw node or a framework built on top of it. Express is a good choice — see [Express Web Framework (Node.js/JavaScript)](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs).
* To run PHP server-side code, launch [PHP's built-in development server](https://php.net/manual/en/features.commandline.webserver.php):

$ cd path/to/your/php/code

$ php -S localhost:8000

<https://www.geeksforgeeks.org/setting-up-a-simple-http-server-using-python/>

### Accessing the server over a Network

Before going into application make a note that the device on which you have run the above commands is called a Server or a Host and the second device that you will be using to access the Server over the network is called Client. For accessing the server over a Network make sure that both the device(Server and the Client) are connected over the same LAN or WLAN network. To access the Server you need the IP address of the server.

For obtaining the IP address the following steps are to be followed on your Server device:

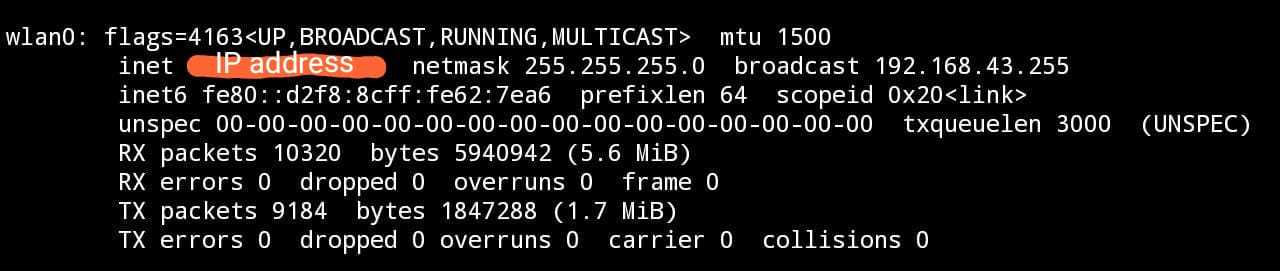
* On the Windows command prompt, execute the following statement:

ipconfig

* On the Linux, Unix or macOS terminal, execute the following statement:

ifconfig

* Note the IP address returned by the above command. We will use this IP address further.



*The result of ‘ifconfig’ command in Linux.*

* Once you know the IP address open any web browser on the Client device and type in the IP address of the first machine (Server device), along with port 8000: http://[ip address]:8000

# First Web Server Program

<https://pythonbasics.org/webserver/>

# ServerEx1.py

# Python 3 server example.

# After running this code, open the browser to http://localhost:8000

from http.server import BaseHTTPRequestHandler, HTTPServer

import time

hostName = "localhost"

serverPort = 8000

class MyServer(BaseHTTPRequestHandler):

    def do\_GET(self):

        self.send\_response(200)

        self.send\_header("Content-type", "text/html")

        self.end\_headers()

        self.wfile.write(bytes("<html><head><title>https://pythonbasics.org</title></head>", "utf-8"))

        self.wfile.write(bytes("<p>Request: %s</p>" % self.path, "utf-8"))

        self.wfile.write(bytes("<body>", "utf-8"))

        self.wfile.write(bytes("<p>This is an example web server.</p>", "utf-8"))

        self.wfile.write(bytes("</body></html>", "utf-8"))

if \_\_name\_\_ == "\_\_main\_\_":

    webServer = HTTPServer((hostName, serverPort), MyServer)

    print("Server started http://%s:%s" % (hostName, serverPort))

    try:

        webServer.serve\_forever()

    except KeyboardInterrupt:

        pass

    webServer.server\_close()

    print("Server stopped.")

## Similar Example, with more details

<https://flaviocopes.com/python-http-server/>

# ServerEx2.py

from http.server import BaseHTTPRequestHandler, HTTPServer

hostName = "localhost"

serverPort = 8000

class handler(BaseHTTPRequestHandler):

    def do\_GET(self):

        self.send\_response(200)

        self.send\_header('Content-type','text/html')

        self.end\_headers()

        message = "Hello, World!"

        self.wfile.write(bytes(message, "utf8"))

if \_\_name\_\_ == "\_\_main\_\_":

    webServer = HTTPServer((hostName, serverPort), handler)

    print("Server started http://%s:%s" % (hostName, serverPort))

    try:

        webServer.serve\_forever()

    except KeyboardInterrupt:

        pass

    webServer.server\_close()

    print("Server stopped.")

After running this program locally, you can try connecting using a Web browser to port [http://localhost:8000](http://localhost:8000/).

This serves the same Hello, World! string to anyone visiting the site on http://localhost:8000, on GET requests to any URL, complete with a 200 response and a Content-type: text/html header.

We write to wfile, which contains the output stream for writing a response back to the client.

It works on GET requests because we implemented the handler do\_GET method.

You can also implement do\_HEAD(), do\_POST() and any other HTTP method:

# BasicServerWithGetPost.py

from http.server import BaseHTTPRequestHandler, HTTPServer

hostName = "localhost"

serverPort = 8000

class handler(BaseHTTPRequestHandler):

    def do\_GET(self):

        self.send\_response(200)

        self.send\_header('Content-type','text/html')

        self.end\_headers()

        message = "Hello, World! Here is a GET response"

        self.wfile.write(bytes(message, "utf8"))

    def do\_POST(self):

        self.send\_response(200)

        self.send\_header('Content-type','text/html')

        self.end\_headers()

        message = "Hello, World! Here is a POST response"

        self.wfile.write(bytes(message, "utf8"))

if \_\_name\_\_ == "\_\_main\_\_":

    webServer = HTTPServer((hostName, serverPort), handler)

    print("Server started http://%s:%s" % (hostName, serverPort))

    try:

        webServer.serve\_forever()

    except KeyboardInterrupt:

        pass

    webServer.server\_close()

    print("Server stopped.")

# HTTP Status Codes

HTTP response status codes indicate whether a specific [HTTP](https://developer.mozilla.org/en-US/docs/Web/HTTP) request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (100–199)
2. Successful responses (200–299)
3. Redirection messages (300–399)
4. Client error responses (400–499)
5. Server error responses (500–599)

Note: If you receive a response that is not in [this list](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#information-responses), it is a non-standard response, possibly custom to the server's software.

## [Information responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#information_responses)

**~~100 Continue~~**

~~This interim response indicates that the client should continue the request or ignore the response if the request is already finished.~~

**~~101 Switching Protocols~~**

~~This code is sent in response to an Upgrade request header from the client and indicates the protocol the server is switching to.~~

**~~102 Processing~~**~~(WebDAV)~~

~~This code indicates that the server has received and is processing the request, but no response is available yet.~~

**~~103 Early Hints~~**

~~This status code is primarily intended to be used with the Link header, letting the user agent start preloading resources while the server prepares a response.~~

## [Successful responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#successful_responses)

**200 OK**

The request succeeded. The result meaning of "success" depends on the HTTP method:

GET: The resource has been fetched and transmitted in the message body.

HEAD: The representation headers are included in the response without any message body.

PUT or POST: The resource describing the result of the action is transmitted in the message body.

TRACE: The message body contains the request message as received by the server.

**201 Created**

The request succeeded, and a new resource was created as a result. This is typically the response sent after POST requests, or some PUT requests.

**202 Accepted**

The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

**203 Non-Authoritative Information**

This response code means the returned metadata is not exactly the same as is available from the origin server, but is collected from a local or a third-party copy. This is mostly used for mirrors or backups of another resource. Except for that specific case, the 200 OK response is preferred to this status.

**204 No Content**

There is no content to send for this request, but the headers may be useful. The user agent may update its cached headers for this resource with the new ones.

## [Redirection messages](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#redirection_messages)

**300 Multiple Choice**

The request has more than one possible response. The user agent or user should choose one of them. (There is no standardized way of choosing one of the responses, but HTML links to the possibilities are recommended so the user can pick.)

**301 Moved Permanently**

The URL of the requested resource has been changed permanently. The new URL is given in the response.

**302 Found**

This response code means that the URI of requested resource has been changed temporarily. Further changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.

## [Client error responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#client_error_responses)

**400 Bad Request**

The server could not understand the request due to invalid syntax.

**401 Unauthorized**

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

**402 Payment Required**

This response code is reserved for future use. The initial aim for creating this code was using it for digital payment systems, however this status code is used very rarely and no standard convention exists.

**403 Forbidden**

The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401 Unauthorized, the client's identity is known to the server.

**404 Not Found**

The server cannot find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 Forbidden to hide the existence of a resource from an unauthorized client. This response code is probably the most well-known due to its frequent occurrence on the web.

## [Server error responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#server_error_responses)

**500 Internal Server Error**

The server has encountered a situation it does not know how to handle.

**501 Not Implemented**

The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.

**502 Bad Gateway**

This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.

**503 Service Unavailable**

The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded. Note that together with this response, a user-friendly page explaining the problem should be sent. This response should be used for temporary conditions and the Retry-After HTTP header should, if possible, contain the estimated time before the recovery of the service. The webmaster must also take care about the caching-related headers that are sent along with this response, as these temporary condition responses should usually not be cached.

[**504 Gateway Timeout**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/504)

This error response is given when the server is acting as a gateway and cannot get a response in time.

**505 HTTP Version Not Supported**

The HTTP version used in the request is not supported by the server.

# HTTP Headers

HTTP headers let the client and the server pass additional information with an HTTP request or response. An HTTP header consists of its case-insensitive name followed by a colon (:), then by its value. Whitespace before the value is ignored.

Custom proprietary headers have historically been used with an X- prefix, but this convention was deprecated in June 2012 because of the inconveniences it caused when nonstandard fields became standard in RFC 6648; others are listed in an IANA registry, whose original content was defined in RFC 4229. IANA also maintains a registry of proposed new HTTP headers.

Headers can be grouped according to their contexts:

* Request headers contain more information about the resource to be fetched, or about the client requesting the resource.
* Response headers hold additional information about the response, like its location or about the server providing it.
* Representation headers contain information about the body of the resource, like its MIME type, or encoding/compression applied.
* Payload headers contain representation-independent information about payload data, including content length and the encoding used for transport.

# HTTP Methods

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable.

**GET**

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

**HEAD**

The HEAD method asks for a response identical to a GET request, but without the response body.

**POST**

The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

[**PUT**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT)

The PUT method replaces all current representations of the target resource with the request payload.

**DELETE**

The DELETE method deletes the specified resource.

**CONNECT**

The CONNECT method establishes a tunnel to the server identified by the target resource.

**OPTIONS**

The OPTIONS method describes the communication options for the target resource.

**TRACE**

The TRACE method performs a message loop-back test along the path to the target resource.

**PATCH**

The PATCH method applies partial modifications to a resource.

## GET vs POST

The following table compares the two HTTP methods: GET and POST.

|  |  |  |
| --- | --- | --- |
|  | **GET** | **POST** |
| BACK button/Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be cached | Not cached |
| Encoding type | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| Restrictions on data type | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL.  Never use GET when sending passwords or other sensitive information! | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

# HTML 5

HTML is the standard markup language for Web pages. With HTML you can create your own website.

**What is HTML?**

* HTML stands for Hyper Text Markup Language
* HTML is the standard markup language for creating Web pages
* HTML describes the structure of a Web page
* HTML consists of a series of elements
* HTML elements tell the browser how to display the content
* HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

**Example:**

<!DOCTYPE**html**>  
<html>  
<head>  
<title>Page Title</title>

</head>  
<body>  
<h1>This is a Heading</h1>

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

</body>  
</html>

**Example Explained:**

* The <!DOCTYPE html> declaration defines that this document is an HTML5 document
* The <html> element is the root element of an HTML page
* The <head> element contains meta information about the HTML page
* The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
* The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
* The <h1> element defines a large heading
* The <p> element defines a paragraph

## What is an HTML Element?

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

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

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

<h1>My First Heading</h1>

<p>My first paragraph.</p>

|  |  |  |
| --- | --- | --- |
| **Start tag** | **Element content** | **End tag** |
| <h1> | My First Heading | </h1> |
| <p> | My first paragraph. | </p> |
| <br> | none | none |

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

## Web Browsers

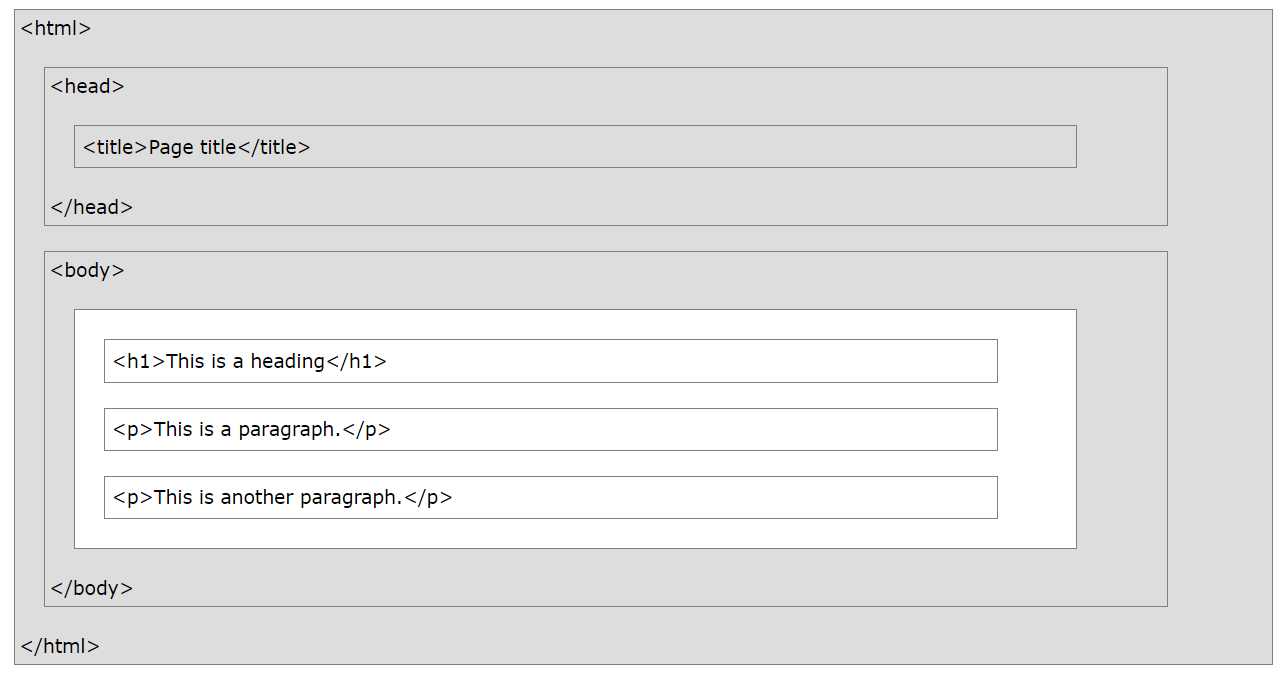
The purpose of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them correctly.

A browser does not display the HTML tags, but uses them to determine how to display the document:



## HTML Page Structure

Below is a visualization of an HTML page structure:



Note: The content inside the <body> section (the white area above) will be displayed in a browser. The content inside the <title> element will be shown in the browser's title bar or in the page's tab.

## HTML History

Since the early days of the World Wide Web, there have been many versions of HTML:

|  |  |
| --- | --- |
| **Year** | **Version** |
| 1989 | Tim Berners-Lee invented www |
| 1991 | Tim Berners-Lee invented HTML |
| 1993 | Dave Raggett drafted HTML+ |
| 1995 | HTML Working Group defined HTML 2.0 |
| 1997 | W3C Recommendation: HTML 3.2 |
| 1999 | W3C Recommendation: HTML 4.01 |
| 2000 | W3C Recommendation: XHTML 1.0 |
| 2008 | WHATWG HTML5 First Public Draft |
| 2012 | WHATWG HTML5 Living Standard |
| 2014 | W3C Recommendation: HTML5 |
| 2016 | W3C Candidate Recommendation: HTML 5.1 |
| 2017 | W3C Recommendation: HTML5.1 2nd Edition |
| 2017 | W3C Recommendation: HTML5.2 |

## HTML Basic Examples

### HTML Documents

All HTML documents must start with a document type declaration: <!DOCTYPE html>.

The HTML document itself begins with <html> and ends with </html>.

The visible part of the HTML document is between <body> and </body>.

**Example**

<!DOCTYPE**html**>  
<html>  
<body>  
<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>  
</html>

### The <!DOCTYPE> Declaration

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

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

The <!DOCTYPE> declaration is not case sensitive.

The <!DOCTYPE> declaration for HTML5 is:

<!DOCTYPE**html**>

### HTML Headings

HTML headings are defined with the <h1> to <h6> tags.

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

**Example**

<h1>This is heading 1</h1>

<h2>This is heading 2</h2>

<h3>This is heading 3</h3>

### HTML Paragraphs

HTML paragraphs are defined with the <p> tag:

**Example**

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

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

### HTML Links

HTML links are defined with the <a> tag:

**Example**

<a**href**="https://www.w3schools.com">This is a link</a>

The link's destination is specified in the href attribute.

Attributes are used to provide additional information about HTML elements.

You will learn more about attributes in a later.

### HTML Images

HTML images are defined with the <img> tag.

The source file (src), alternative text (alt), width, and height are provided as attributes:

**Example**

<img**src**="w3schools.jpg"**alt**="W3Schools.com"**width**="104"**height**="142">

## HTML Elements

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

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

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

Examples of some HTML elements:

<h1>My First Heading</h1>

<p>My first paragraph.</p>

|  |  |  |
| --- | --- | --- |
| **Start tag** | **Element content** | **End tag** |
| <h1> | My First Heading | </h1> |
| <p> | My first paragraph. | </p> |
| <br> | none | none |

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

### Nested HTML Elements

HTML elements can be nested (this means that elements can contain other elements).

All HTML documents consist of nested HTML elements.

The following example contains four HTML elements (<html>, <body>, <h1> and <p>):

**Example**

<!DOCTYPE**html**>  
<html>  
<body>  
<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>  
</html>

**Example Explained**

The <html> element is the root element and it defines the whole HTML document.

It has a start tag <html> and an end tag </html>.

Then, inside the <html> element there is a <body> element:

<body>  
<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>

The <body> element defines the document's body.

It has a start tag <body> and an end tag </body>.

Then, inside the <body> element there are two other elements: <h1> and <p>:

<h1>My First Heading</h1>

<p>My first paragraph.</p>

The <h1> element defines a heading.

It has a start tag <h1> and an end tag </h1>:

<h1>My First Heading</h1>

The <p> element defines a paragraph.

It has a start tag <p> and an end tag </p>:

<p>My first paragraph.</p>

### Never Skip the End Tag

Some HTML elements will display correctly, even if you forget the end tag:

**Example**

<html>  
<body>  
<p>This is a paragraph

<p>This is a paragraph

</body>  
</html>

However, never rely on this! Unexpected results and errors may occur if you forget the end tag!

### Empty HTML Elements

HTML elements with no content are called empty elements.

The <br> tag defines a line break, and is an empty element without a closing tag:

**Example**

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

### HTML is Not Case Sensitive

HTML tags are not case sensitive: <P> means the same as <p>.

The HTML standard does not require lowercase tags, but W3C **recommends** lowercase in HTML, and **demands** lowercase for stricter document types like XHTML.

# HTML Attributes

HTML attributes provide additional information about HTML elements.

* All HTML elements can have **attributes**
* Attributes provide **additional information** about elements
* Attributes are always specified in **the start tag**
* Attributes usually come in name/value pairs like: **name="value"**

### The href Attribute

The <a> tag defines a hyperlink. The href attribute specifies the URL of the page the link goes to:

**Example**

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

### The src Attribute

The <img> tag is used to embed an image in an HTML page. The src attribute specifies the path to the image to be displayed:

**Example**

<img**src**="img\_girl.jpg">

There are two ways to specify the URL in the src attribute:

1. Absolute URL - Links to an external image that is hosted on another website. Example: src="https://www.w3schools.com/images/img\_girl.jpg".

Notes: External images might be under copyright. If you do not get permission to use it, you may be in violation of copyright laws. In addition, you cannot control external images; it can suddenly be removed or changed.

1. Relative URL - Links to an image that is hosted within the website. Here, the URL does not include the domain name. If the URL begins without a slash, it will be relative to the current page. Example: src="img\_girl.jpg". If the URL begins with a slash, it will be relative to the domain. Example: src="/images/img\_girl.jpg".

Tip: It is almost always best to use relative URLs. They will not break if you change domain.

### The width and height Attributes

The <img> tag should also contain the width and height attributes, which specifies the width and height of the image (in pixels):

**Example**

<img**src**="img\_girl.jpg"**width**="500"**height**="600">

### The alt Attribute

The required alt attribute for the <img> tag specifies an alternate text for an image, if the image for some reason cannot be displayed. This can be due to slow connection, or an error in the src attribute, or if the user uses a screen reader.

**Example**

<img**src**="img\_girl.jpg"**alt**="Girl with a jacket">

**Example**

See what happens if we try to display an image that does not exist:

<img**src**="img\_typo.jpg"**alt**="Girl with a jacket">

### The style Attribute

The style attribute is used to add styles to an element, such as color, font, size, and more.

**Example**

<p**style**="color:red;">This is a red paragraph.</p>

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_attributes_style)

### The lang Attribute

You should always include the lang attribute inside the <html> tag, to declare the language of the Web page. This is meant to assist search engines and browsers.

The following example specifies English as the language:

<!DOCTYPE**html**>  
<html**lang**="en">  
<body>  
...  
</body>  
</html>

Country codes can also be added to the language code in the lang attribute. So, the first two characters define the language of the HTML page, and the last two characters define the country.

The following example specifies English as the language and United States as the country:

<!DOCTYPE**html**>  
<html**lang**="en-US">  
<body>  
...  
</body>  
</html>

### The title Attribute

The title attribute defines some extra information about an element.

The value of the title attribute will be displayed as a tooltip when you mouse over the element:

**Example**

<p**title**="I'm a tooltip">This is a paragraph.</p>

### Always Use Lowercase Attributes

The HTML standard does not require lowercase attribute names.

The title attribute (and all other attributes) can be written with uppercase or lowercase like title or TITLE.

However, W3C recommends lowercase attributes in HTML, and demands lowercase attributes for stricter document types like XHTML.

### Always Quote Attribute Values

The HTML standard does not require quotes around attribute values.

However, W3C **recommends** quotes in HTML, and demands quotes for stricter document types like XHTML.

**Good:**

<a**href**="https://www.w3schools.com/html/">Visit our HTML tutorial</a>

**Bad:**

<a**href**=https://www.w3schools.com/html/>Visit our HTML tutorial</a>

Sometimes you have to use quotes. This example will not display the title attribute correctly, because it contains a space:

**Example**

<p**title**=About **W3Schools**>

### Single or Double Quotes?

Double quotes around attribute values are the most common in HTML, but single quotes can also be used.

In some situations, when the attribute value itself contains double quotes, it is necessary to use single quotes:

<p**title**='John "ShotGun" Nelson'>

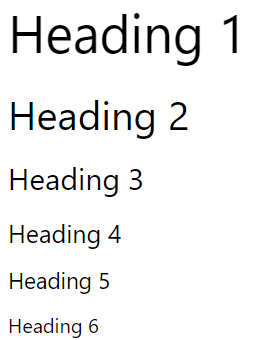
Or vice versa:

<p**title**="John 'ShotGun' Nelson">

# HTML Headings

HTML headings are titles or subtitles that you want to display on a webpage.

**Example:**



## HTML Headings

HTML headings are defined with the <h1> to <h6> tags.

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

**Example**

<h1>Heading 1</h1>

<h2>Heading 2</h2>

<h3>Heading 3</h3>

<h4>Heading 4</h4>

<h5>Heading 5</h5>

<h6>Heading 6</h6>

Note: Browsers automatically add some white space (a margin) before and after a heading.

## Headings Are Important

Search engines use the headings to index the structure and content of your web pages.

Users often skim a page by its headings. It is important to use headings to show the document structure.

<h1> headings should be used for main headings, followed by <h2> headings, then the less important <h3>, and so on.

Note: Use HTML headings for headings only. Don't use headings to make text BIG or bold.

## Bigger Headings

Each HTML heading has a default size. However, you can specify the size for any heading with the style attribute, using the CSS font-size property:

**Example**

<h1**style**="font-size:60px;">Heading 1</h1>

# HTML Paragraphs

A paragraph always starts on a new line, and is usually a block of text.

The HTML <p> element defines a paragraph.

A paragraph always starts on a new line, and browsers automatically add some white space (a margin) before and after a paragraph.

**Example**

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

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

## HTML Display

You cannot be sure how HTML will be displayed.

Large or small screens, and resized windows will create different results.

With HTML, you cannot change the display by adding extra spaces or extra lines in your HTML code.

The browser will automatically remove any extra spaces and lines when the page is displayed:

**Example**

<p>  
This paragraph

contains a lot of lines

in the source code,

but the browser

ignores it.

</p>  
  
<p>  
This paragraph

contains a lot of spaces

in the source code,

but the browser

ignores it.

</p>

## HTML Horizontal Rules

The <hr> tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.

The <hr> element is used to separate content (or define a change) in an HTML page:

**Example**

<h1>This is heading 1</h1>

<p>This is some text.</p>

<hr>  
<h2>This is heading 2</h2>

<p>This is some other text.</p>

<hr>

The <hr> tag is an empty tag, which means that it has no end tag.

## HTML Line Breaks

The HTML <br> element defines a line break.

Use <br> if you want a line break (a new line) without starting a new paragraph:

**Example**

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

The <br> tag is an empty tag, which means that it has no end tag.

## The Poem Problem

This poem will display on a single line:

**Example**

<p>  
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.

</p>

## Solution - The HTML <pre> Element

The HTML <pre> element defines preformatted text.

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

**Example**

<pre>  
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.

</pre>

# HTML Styles

The HTML style attribute is used to add styles to an element, such as color, font, size, and more.

**Example**:



## The HTML Style Attribute

Setting the style of an HTML element, can be done with the style attribute.

The HTML style attribute has the following syntax:

<tagname**style**="property:value;">

The *property* is a CSS property. The *value* is a CSS value.

## Background Color

The CSS background-color property defines the background color for an HTML element.

**Example**

Set the background color for a page to powderblue:

<body**style**="background-color:powderblue;">  
<h1>This is a heading</h1>

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

</body>

**Example**

Set background color for two different elements:

<body>  
<h1**style**="background-color:powderblue;">This is a heading</h1>

<p**style**="background-color:tomato;">This is a paragraph.</p>

</body>

## Text Color

The CSS color property defines the text color for an HTML element:

**Example**

<h1**style**="color:blue;">This is a heading</h1>

<p**style**="color:red;">This is a paragraph.</p>

## Fonts

The CSS font-family property defines the font to be used for an HTML element:

**Example**

<h1**style**="font-family:verdana;">This is a heading</h1>

<p**style**="font-family:courier;">This is a paragraph.</p>

## Text Size

The CSS font-size property defines the text size for an HTML element:

**Example**

<h1**style**="font-size:300%;">This is a heading</h1>

<p**style**="font-size:160%;">This is a paragraph.</p>

## Text Alignment

The CSS text-align property defines the horizontal text alignment for an HTML element:

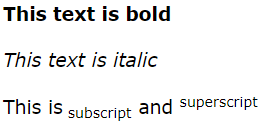
**Example**

<h1**style**="text-align:center;">Centered Heading</h1>

<p**style**="text-align:center;">Centered paragraph.</p>

# HTML Text Formatting

HTML contains several elements for defining text with a special meaning.



## HTML Formatting Elements

Formatting elements were designed to display special types of text:

* <b> - Bold text
* <strong> - Important text
* <i> - Italic text
* <em> - Emphasized text
* <mark> - Marked text
* <small> - Smaller text
* <del> - Deleted text
* <ins> - Inserted text
* <sub> - Subscript text
* <sup> - Superscript text

## HTML <b> and <strong> Elements

The HTML <b> element defines bold text, without any extra importance.

**Example**

<b>This text is bold</b>

The HTML <strong> element defines text with strong importance. The content inside is typically displayed in bold.

**Example**

<strong>This text is important!</strong>

## HTML <i> and <em> Elements

The HTML <i> element defines a part of text in an alternate voice or mood. The content inside is typically displayed in italic.

Tip: The <i> tag is often used to indicate a technical term, a phrase from another language, a thought, a ship name, etc.

**Example**

<i>This text is italic</i>

The HTML <em> element defines emphasized text. The content inside is typically displayed in italic.

Tip: A screen reader will pronounce the words in <em> with an emphasis, using verbal stress.

**Example**

<em>This text is emphasized</em>

## HTML <small> Element

The HTML <small> element defines smaller text:

**Example**

<small>This is some smaller text.</small>

## HTML <mark> Element

The HTML <mark> element defines text that should be marked or highlighted:

**Example**

<p>Do not forget to buy <mark>milk</mark> today.</p>

## HTML <del> Element

The HTML <del> element defines text that has been deleted from a document. Browsers will usually strike a line through deleted text:

**Example**

<p>My favorite color is <del>blue</del> red.</p>

## HTML <ins> Element

The HTML <ins> element defines a text that has been inserted into a document. Browsers will usually underline inserted text:

**Example**

<p>My favorite color is <del>blue</del> <ins>red</ins>.</p>

## HTML <sub> Element

The HTML <sub> element defines subscript text. Subscript text appears half a character below the normal line, and is sometimes rendered in a smaller font. Subscript text can be used for chemical formulas, like H2O:

**Example**

<p>This is <sub>subscripted</sub> text.</p>

## HTML <sup> Element

The HTML <sup> element defines superscript text. Superscript text appears half a character above the normal line, and is sometimes rendered in a smaller font. Superscript text can be used for footnotes, like WWW[1]:

**Example**

<p>This is <sup>superscripted</sup> text.</p>

## HTML Text Formatting Elements

|  |  |
| --- | --- |
| Tag | Description |
| [<b>](https://www.w3schools.com/tags/tag_b.asp) | Defines bold text |
| [<em>](https://www.w3schools.com/tags/tag_em.asp) | Defines emphasized text |
| [<i>](https://www.w3schools.com/tags/tag_i.asp) | Defines a part of text in an alternate voice or mood |
| [<small>](https://www.w3schools.com/tags/tag_small.asp) | Defines smaller text |
| [<strong>](https://www.w3schools.com/tags/tag_strong.asp) | Defines important text |
| [<sub>](https://www.w3schools.com/tags/tag_sub.asp) | Defines subscripted text |
| [<sup>](https://www.w3schools.com/tags/tag_sup.asp) | Defines superscripted text |
| [<ins>](https://www.w3schools.com/tags/tag_ins.asp) | Defines inserted text |
| [<del>](https://www.w3schools.com/tags/tag_del.asp) | Defines deleted text |
| [<mark>](https://www.w3schools.com/tags/tag_mark.asp) | Defines marked/highlighted text |

# HTML Colors

HTML colors are specified with predefined color names, or with RGB, HEX, HSL, RGBA, or HSLA values.

## Color Names

In HTML, a color can be specified by using a color name:



HTML supports 140 standard color names.

## Background Color

You can set the background color for HTML elements:

Hello World

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

**Example**

<h1**style**="background-color:DodgerBlue;">Hello World</h1>

<p**style**="background-color:Tomato;">Lorem ipsum...</p>

## Text Color

You can set the color of text:

**Hello World**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

**Example**

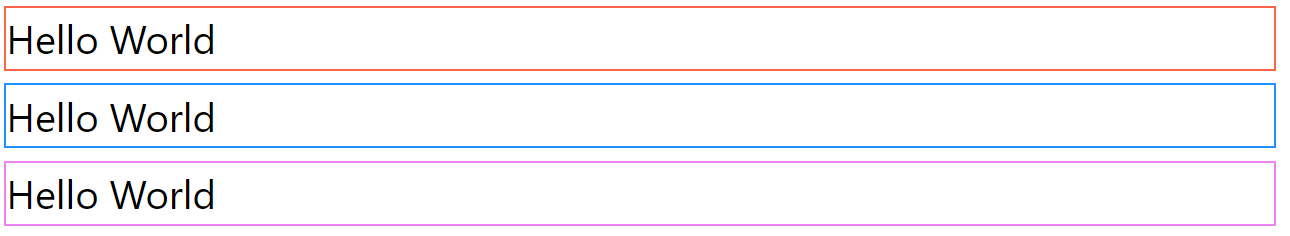
<h1**style**="color:Tomato;">Hello World</h1>

<p**style**="color:DodgerBlue;">Lorem ipsum...</p>

<p**style**="color:MediumSeaGreen;">Ut wisi enim...</p>

## Border Color

You can set the color of borders:



**Example**

<h1**style**="border:2px solid Tomato;">Hello World</h1>

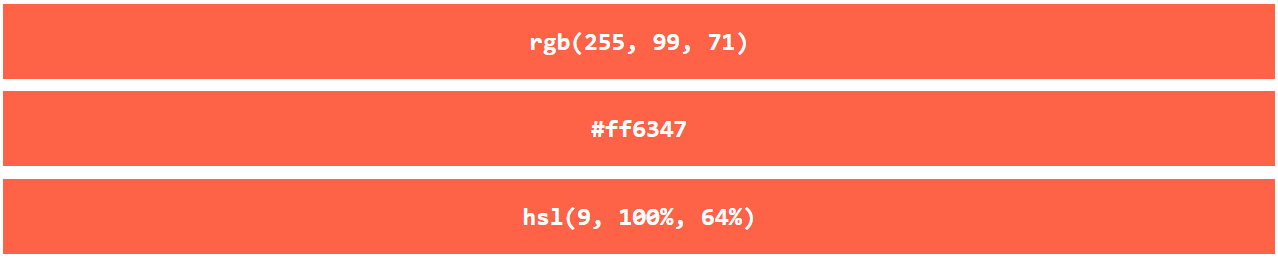
<h1**style**="border:2px solid DodgerBlue;">Hello World</h1>

<h1**style**="border:2px solid Violet;">Hello World</h1>

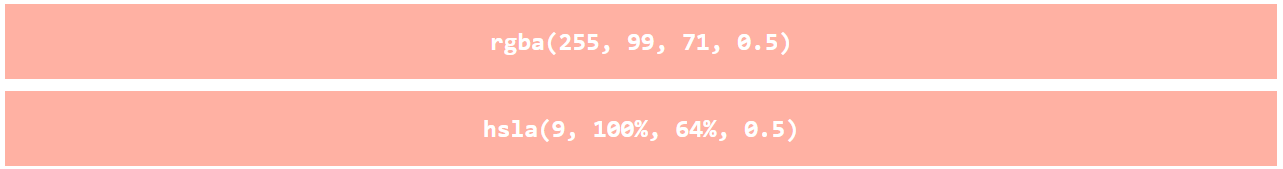
## Color Values

In HTML, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values.

The following three <div> elements have their background color set with RGB, HEX, and HSL values:



The following two <div> elements have their background color set with RGBA and HSLA values, which adds an Alpha channel to the color (here we have 50% transparency):



**Example**

<h1**style**="background-color:rgb(255, 99, 71);">...</h1>

<h1**style**="background-color:#ff6347;">...</h1>  
<h1**style**="background-color:hsl(9, 100%, 64%);">...</h1>

<h1**style**="background-color:rgba(255, 99, 71, 0.5);">...</h1>

<h1**style**="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>

# HTML Styles - CSS

CSS stands for Cascading Style Sheets.

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

## What is CSS?

Cascading Style Sheets (CSS) is used to format the layout of a webpage.

CSS is the language we use to style an HTML document.

CSS describes how HTML elements should be displayed.

With CSS, you can control the color, font, the size of text, the spacing between elements, how elements are positioned and laid out, what background images or background colors are to be used, different displays for different devices and screen sizes, and much more!

Tip: The word cascading means that a style applied to a parent element will also apply to all children elements within the parent. So, if you set the color of the body text to "blue", all headings, paragraphs, and other text elements within the body will also get the same color (unless you specify something else)!

## Using CSS

CSS can be added to HTML documents in 3 ways:

* Inline - by using the style attribute inside HTML elements
* Internal - by using a <style> element in the <head> section
* External - by using a <link> element to link to an external CSS file

The most common way to add CSS, is to keep the styles in external CSS files. However, in this tutorial we will use inline and internal styles, because this is easier to demonstrate, and easier for you to try it yourself.

## Inline CSS

An inline CSS is used to apply a unique style to a single HTML element.

An inline CSS uses the style attribute of an HTML element.

The following example sets the text color of the <h1> element to blue, and the text color of the <p> element to red:

**Example**

<h1**style**="color:blue;">A Blue Heading</h1>

<p**style**="color:red;">A red paragraph.</p>

## Internal CSS

An internal CSS is used to define a style for a single HTML page.

An internal CSS is defined in the <head> section of an HTML page, within a <style> element.

The following example sets the text color of ALL the <h1> elements (on that page) to blue, and the text color of ALL the <p> elements to red. In addition, the page will be displayed with a "powderblue" background color:

**Example**

<!DOCTYPE**html**>  
<html>  
<head>  
<style>  
body {background-color: powderblue;}  
h1   {color: blue;}  
p    {color: red;}  
</style>  
</head>  
<body>  
<h1>This is a heading</h1>

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

</body>  
</html>

## External CSS

An external style sheet is used to define the style for many HTML pages.

To use an external style sheet, add a link to it in the <head> section of each HTML page:

**Example**

<!DOCTYPE**html**>  
<html>  
<head>  
  <link**rel**="stylesheet"**href**="styles.css">  
</head>  
<body>  
<h1>This is a heading</h1>

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

</body>  
</html>

The external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.

Here is what the "styles.css" file looks like:

**"styles.css":**

body {  
  background-color: powderblue;  
}  
h1 {  
  color: blue;

}  
p {  
  color: red;  
}

Tip: With an external style sheet, you can change the look of an entire web site, by changing one file!

## CSS Syntax

A CSS rule consists of a selector and a declaration block.



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

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

Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

**Example**

In this example all <p> elements will be center-aligned, with a red text color:

p {  
  color: red;  
  text-align: center;  
}

## CSS Colors, Fonts and Sizes

Here, we will demonstrate some commonly used CSS properties. You will learn more about them later.

The CSS color property defines the text color to be used.

The CSS font-family property defines the font to be used.

The CSS font-size property defines the text size to be used.

**Example**

Use of CSS color, font-family and font-size properties:

<!DOCTYPE**html**>  
<html>  
<head>  
<style>  
h1 {  
  color: blue;  
  font-family: verdana;  
  font-size: 300%;  
}  
p {  
  color: red;

  font-family: courier;

  font-size: 160%;

}  
</style>  
</head>  
<body>  
<h1>This is a heading</h1>

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

</body>  
</html>

## CSS Border

The CSS border property defines a border around an HTML element.

Tip: You can define a border for nearly all HTML elements.

**Example**

Use of CSS border property:

p {  
  border: 2px solid powderblue;

}

## CSS Padding

The CSS padding property defines a padding (space) between the text and the border.

**Example**

Use of CSS border and padding properties:

p {  
  border: 2px solid powderblue;

  padding: 30px;

}

## CSS Margin

The CSS margin property defines a margin (space) outside the border.

**Example**

Use of CSS border and margin properties:

p {  
  border: 2px solid powderblue;

  margin: 50px;  
}

## Link to External CSS

External style sheets can be referenced with a full URL or with a path relative to the current web page.

**Example**

This example uses a full URL to link to a style sheet:

<link**rel**="stylesheet"**href**="https://www.w3schools.com/html/styles.css">

**Example**

This example links to a style sheet located in the html folder on the current web site:

<link**rel**="stylesheet"**href**="/html/styles.css">

**Example**

This example links to a style sheet located in the same folder as the current page:

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

# HTML Links – Hyperlinks

Links are found in nearly all web pages. Links allow users to click their way from page to page.

HTML links are hyperlinks.

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

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

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

## HTML Links - Syntax

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

<a**href**="*url*">*link text*</a>

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

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

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

**Example**

This example shows how to create a link to W3Schools.com:

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

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

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

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

## HTML Links - The target Attribute

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

The target attribute specifies where to open the linked document.

The target attribute can have one of the following values:

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

**Example**

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

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

## Absolute URLs vs. Relative URLs

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

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

**Example**

<h2>Absolute URLs</h2>

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

<p><a**href**="html\_images.asp">HTML Images</a></p>

<p><a**href**="/css/default.asp">CSS Tutorial</a></p>

## HTML Links - Use an Image as a Link

To use an image as a link, just put the <img> tag inside the <a> tag:

**Example**

<a**href**="default.asp">  
<img**src**="smiley.gif"**alt**="HTML tutorial"**style**="width:42px;height:42px;">  
</a>

## Link to an Email Address

Use mailto: inside the href attribute to create a link that opens the user's email program (to let them send a new email):

**Example**

<a**href**="mailto:someone@example.com">Send email</a>

## Button as a Link

To use an HTML button as a link, you have to add some JavaScript code.

JavaScript allows you to specify what happens at certain events, such as a click of a button:

**Example**

<button**onclick**="document.location='default.asp'">HTML Tutorial</button>

## Link Titles

The title attribute specifies extra information about an element. The information is most often shown as a tooltip text when the mouse moves over the element.

**Example**

<a**href**="https://www.w3schools.com/html/"**title**="Go to W3Schools HTML section">Visit our HTML Tutorial</a>

## More on Absolute URLs and Relative URLs

**Example**

Use a full URL to link to a web page:

<a**href**="https://www.w3schools.com/html/default.asp">HTML tutorial</a>

**Example**

Link to a page located in the html folder on the current web site:

<a**href**="/html/default.asp">HTML tutorial</a>

**Example**

Link to a page located in the same folder as the current page:

<a**href**="default.asp">HTML tutorial</a>

# HTML Links – Different Colors

An HTML link is displayed in a different color depending on whether it has been visited, is unvisited, or is active.

By default, a link will appear like this (in all browsers):

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

You can change the link state colors, by using CSS:

**Example**

Here, an unvisited link will be green with no underline. A visited link will be pink with no underline. An active link will be yellow and underlined. In addition, when mousing over a link (a:hover) it will become red and underlined:

<style>  
a:link {  
  color: green;  
  background-color: transparent;  
  text-decoration: none;  
}  
a:visited {  
  color: pink;

  background-color: transparent;

  text-decoration: none;

}  
a:hover {  
  color: red;

  background-color: transparent;

  text-decoration: underline;

}  
a:active {  
  color: yellow;

  background-color: transparent;

  text-decoration: underline;

}  
</style>

## Link Buttons

A link can also be styled as a button, by using CSS:

[This is a link](javascript:void(0))

**Example**

<style>  
a:link, a:visited {

  background-color: #f44336;

  color: white;

  padding: 15px 25px;

  text-align: center;

  text-decoration: none;

  display: inline-block;

}  
  
a:hover, a:active {

  background-color: red;

}  
</style>

# HTML Images

Images can improve the design and the appearance of a web page.

**Example**

<img**src**="pic\_trulli.jpg"**alt**="Italian Trulli">

**Example**

<img**src**="img\_girl.jpg"**alt**="Girl in a jacket">

**Example**

<img**src**="img\_chania.jpg"**alt**="Flowers in Chania">

## HTML Images Syntax

The HTML <img> tag is used to embed an image in a web page.

Images are not technically inserted into a web page; images are linked to web pages. The <img> tag creates a holding space for the referenced image.

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

The <img> tag has two required attributes:

* src - Specifies the path to the image
* alt - Specifies an alternate text for the image

**Syntax**

<img**src**="*url*"**alt**="alternatetext">

## The src Attribute

The required src attribute specifies the path (URL) to the image.

Note: When a web page loads, it is the browser, at that moment, that gets the image from a web server and inserts it into the page. Therefore, make sure that the image actually stays in the same spot in relation to the web page, otherwise your visitors will get a broken link icon. The broken link icon and the alt text are shown if the browser cannot find the image.

**Example**

<img**src**="img\_chania.jpg"**alt**="Flowers in Chania">

## The alt Attribute

The required alt attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

The value of the alt attribute should describe the image:

**Example**

<img**src**="img\_chania.jpg"**alt**="Flowers in Chania">

If a browser cannot find an image, it will display the value of the alt attribute:

**Example**

<img**src**="wrongname.gif"**alt**="Flowers in Chania">

Tip: A screen reader is a software program that reads the HTML code and allows the user to "listen" to the content. Screen readers are useful for people who are visually impaired or learning disabled.

## Image Size - Width and Height

You can use the style attribute to specify the width and height of an image.

**Example**

<img**src**="img\_girl.jpg"**alt**="Girl in a jacket"

**style**="width:500px;height:600px;">

Alternatively, you can use the width and height attributes:

**Example**

<img**src**="img\_girl.jpg"**alt**="Girl in a jacket"**width**="500"**height**="600">

The width and height attributes always define the width and height of the image in pixels.

Note: Always specify the width and height of an image. If width and height are not specified, the web page might flicker while the image loads.

## Width and Height, or Style?

The width, height, and style attributes are all valid in HTML.

However, we suggest using the style attribute. It prevents styles sheets from changing the size of images:

**Example**

<!DOCTYPE**html**>  
<html>  
<head>  
<style>  
img {  
  width: 100%;  
}  
</style>  
</head>  
<body>  
<img**src**="html5.gif"**alt**="HTML5 Icon"**width**="128"**height**="128">

<img**src**="html5.gif"**alt**="HTML5 Icon"**style**="width:128px;height:128px;">  
</body>  
</html>

## Images in Another Folder

If you have your images in a sub-folder, you must include the folder name in the src attribute:

**Example**

<img**src**="/images/html5.gif"**alt**="HTML5 Icon"

**style**="width:128px;height:128px;">

## Images on Another Server/Website

Some web sites point to an image on another server.

To point to an image on another server, you must specify an absolute (full) URL in the src attribute:

**Example**

<img**src**="https://www.w3schools.com/images/w3schools\_green.jpg"**alt**="W3Schools.com">

Notes on external images: External images might be under copyright. If you do not get permission to use it, you may be in violation of copyright laws. In addition, you cannot control external images; it can suddenly be removed or changed.

## Animated Images

HTML allows animated GIFs:

**Example**

<img**src**="programming.gif"**alt**="Computer Man"

**style**="width:48px;height:48px;">

## Image as a Link

To use an image as a link, put the <img> tag inside the <a> tag:

**Example**

<a**href**="default.asp">  
  <img**src**="smiley.gif"**alt**="HTML tutorial"

**style**="width:42px;height:42px;">  
</a>

## Image Floating

Use the CSS float property to let the image float to the right or to the left of a text:

**Example**

<p><img**src**="smiley.gif"**alt**="Smiley face"**style**="float:right;width:42px;height:42px;">  
The image will float to the right of the text.</p>

<p><img**src**="smiley.gif"**alt**="Smiley face"**style**="float:left;width:42px;height:42px;">  
The image will float to the left of the text.</p>

## Common Image Formats

Here are the most common image file types, which are supported in all browsers (Chrome, Edge, Firefox, Safari, Opera):

|  |  |  |
| --- | --- | --- |
| **Abbreviation** | **File Format** | **File Extension** |
| APNG | Animated Portable Network Graphics | .apng |
| GIF | Graphics Interchange Format | .gif |
| ICO | Microsoft Icon | .ico, .cur |
| JPEG | Joint Photographic Expert Group image | .jpg, .jpeg, .jfif, .pjpeg, .pjp |
| PNG | Portable Network Graphics | .png |
| SVG | Scalable Vector Graphics | .svg |

# HTML Background Images

A background image can be specified for almost any HTML element.

To add a background image on an HTML element, use the HTML style attribute and the CSS background-image property:

**Example**

Add a background image on a HTML element:

<div**style**="background-image: url('img\_girl.jpg');">

<!DOCTYPE html>

<html>

<body>

<h2>Background Image</h2>

<p>A background image for a div element:</p>

<div style="background-image: url('img\_girl.jpg');">

You can specify background images<br>

for any visible HTML element.<br>

In this example, the background image<br>

is specified for a div element.<br>

By default, the background-image<br>

will repeat itself in the direction(s)<br>

where it is smaller than the element<br>

where it is specified. (Try resizing the<br>

browser window to see how the<br>

background image behaves.

</div>

<p>A background image for a p element:</p>

<p style="background-image: url('img\_girl.jpg');">

You can specify background images<br>

for any visible HTML element.<br>

In this example, the background image<br>

is specified for a p element.<br>

By default, the background-image<br>

will repeat itself in the direction(s)<br>

where it is smaller than the element<br>

where it is specified. (Try resizing the<br>

browser window to see how the<br>

background image behaves.

</p>

</body>

</html>

You can also specify the background image in the <style> element, in the <head> section:

**Example**

Specify the background image in the <style> element:

<style>  
div {  
  background-image: url('img\_girl.jpg');

}  
</style>

<!DOCTYPE html>

<html>

<head>

<style>

div {

background-image: url('img\_girl.jpg');

}

</style>

</head>

<body>

<h2>Background Image</h2>

<div>

You can specify background images<br>

for any visible HTML element.<br>

In this example, the background image<br>

is specified for a div element.<br>

By default, the background-image<br>

will repeat itself in the direction(s)<br>

where it is smaller than the element<br>

where it is specified. (Try resizing the<br>

browser window to see how the<br>

background image behaves.

</div>

</body>

</html>

## Background Image on a Page

If you want the entire page to have a background image, you must specify the background image on the <body> element:

**Example**

Add a background image for the entire page:

<style>  
body {  
  background-image: url('img\_girl.jpg');

}  
</style>

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-image: url('img\_girl.jpg');

}

</style>

</head>

<body>

<h2>Background Image</h2>

<p>By default, the background image will repeat itself if it is smaller than the element where it is specified, in this case the body element.</p>

</body>

</html>

## Background Repeat

If the background image is smaller than the element, the image will repeat itself, horizontally and vertically, until it reaches the end of the element:

**Example**

<style>  
body {  
  background-image: url('example\_img\_girl.jpg');

}  
</style>

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-image: url('example\_img\_girl.jpg');

}

</style>

</head>

<body>

<h2>Background Repeat</h2>

<p>By default, the background image will repeat itself if it is smaller than the element where it is specified, in this case the body element.</p>

</body>

</html>

To avoid the background image from repeating itself, set the background-repeat property to no-repeat.

**Example**

<style>  
body {  
  background-image: url('example\_img\_girl.jpg');

  background-repeat: no-repeat;

}  
</style>

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-image: url('example\_img\_girl.jpg');

background-repeat: no-repeat;

}

</style>

</head>

<body>

<h2>Background No Repeat</h2>

<p>You can avoid the image from being repeated by setting the background-repeat property to "no-repeat".</p>

</body>

</html>

## Background Cover

If you want the background image to cover the entire element, you can set the background-size property to cover.

Also, to make sure the entire element is always covered, set the background-attachment property to fixed:

This way, the background image will cover the entire element, with no stretching (the image will keep its original proportions):

**Example**

<style>  
body {  
  background-image: url('img\_girl.jpg');

  background-repeat: no-repeat;

  background-attachment: fixed;

  background-size: cover;

}  
</style>

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-image: url('img\_girl.jpg');

background-repeat: no-repeat;

background-attachment: fixed;

background-size: cover;

}

</style>

</head>

<body>

<h2>Background Cover</h2>

<p>Set the background-size property to "cover" and the background image will cover the entire element, in this case the body element.</p>

</body>

</html>

## Background Stretch

If you want the background image to stretch to fit the entire element, you can set the background-size property to 100% 100%:

Try resizing the browser window, and you will see that the image will stretch, but always cover the entire element.

**Example**

<style>  
body {  
  background-image: url('img\_girl.jpg');

  background-repeat: no-repeat;

  background-attachment: fixed;

  background-size: 100% 100%;

}  
</style>

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-image: url('img\_girl.jpg');

background-repeat: no-repeat;

background-attachment: fixed;

background-size: 100% 100%;

}

</style>

</head>

<body>

<h2>Background Stretch</h2>

<p>Set the background-size property to "100% 100%" and the background image will be stretched to cover the entire element, in this case the body element.</p>

</body>

</html>

# HTML Tables

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

**Example**

|  |  |  |
| --- | --- | --- |
| **Company** | **Contact** | **Country** |
| Alfreds Futterkiste | Maria Anders | Germany |
| Centro comercial Moctezuma | Francisco Chang | Mexico |
| Ernst Handel | Roland Mendel | Austria |
| Island Trading | Helen Bennett | UK |
| Laughing Bacchus Winecellars | Yoshi Tannamuri | Canada |
| Magazzini Alimentari Riuniti | Giovanni Rovelli | Italy |

## Define an HTML Table

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

**Example**

A simple HTML table:

<table>  
  <tr>  
    <th>Company</th>  
    <th>Contact</th>  
    <th>Country</th>  
  </tr>  
  <tr>  
    <td>Alfreds Futterkiste</td>

    <td>Maria Anders</td>

    <td>Germany</td>  
  </tr>  
  <tr>  
    <td>Centro comercial Moctezuma</td>

    <td>Francisco Chang</td>

    <td>Mexico</td>  
  </tr>  
</table>

## Table Cells

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

td stands for table data.

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

**Example**

<table>  
  <tr>  
    <td>Emil</td>

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

Note: table data elements are the data containers of the table.  
They can contain all sorts of HTML elements; text, images, lists, other tables, etc.

## Table Rows

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

tr stands for table row.

**Example**

<table>  
  <tr>  
    <td>Emil</td>

    <td>Tobias</td>  
    <td>Linus</td>  
  </tr>  
  <tr>  
    <td>16</td>

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

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

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

## Table Headers

Sometimes you want your cells to be headers, in those cases use the <th> tag instead of the <td> tag:

**Example**

Let the first row be table headers:

<table>  
  <tr>  
    <th>Person 1</th>

    <th>Person 2</th>

    <th>Person 3</th>

  </tr>  
  <tr>  
    <td>Emil</td>

    <td>Tobias</td>  
    <td>Linus</td>  
  </tr>  
  <tr>  
    <td>16</td>

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

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

## HTML Table Tags

|  |  |
| --- | --- |
| Tag | Description |
| <table> | Defines a table |
| <th> | Defines a header cell in a table |
| <tr> | Defines a row in a table |
| <td> | Defines a cell in a table |
| <caption> | Defines a table caption |
| <colgroup> | Specifies a group of one or more columns in a table for formatting |
| <col> | Specifies column properties for each column within a <colgroup> element |
| <thead> | Groups the header content in a table |
| <tbody> | Groups the body content in a table |
| <tfoot> | Groups the footer content in a table |

# HTML Table Borders

HTML tables can have borders of different styles and shapes.

## How To Add a Border

When you add a border to a table, you also add borders around each table cell:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

To add a border, use the CSS border property on table, th, and td elements:

**Example**

table, th, td {

  border: 1px solid black;

}

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid black;

}

</style>

</head>

<body>

<h2>Table With Border</h2>

<p>Use the CSS border property to add a border to the table.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

</table>

</body>

</html>

## Collapsed Table Borders

To avoid having double borders like in the example above, set the CSS border-collapse property to collapse.

This will make the borders collapse into a single border:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**Example**

table, th, td {

  border: 1px solid black;

  border-collapse: collapse;

}

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid black;

border-collapse: collapse;

}

</style>

</head>

<body>

<h2>Collapsed Borders</h2>

<p>If you want the borders to collapse into one border, add the CSS border-collapse property.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

</table>

</body>

</html>

## Style Table Borders

If you set a background color of each cell, and give the border a white color (the same as the document background), you get the impression of an invisible border:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**Example**

table, th, td {

  border: 1px solid white;

  border-collapse: collapse;

}  
th, td {

  background-color: #96D4D4;

}

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid white;

border-collapse: collapse;

}

th, td {

background-color: #96D4D4;

}

</style>

</head>

<body>

<h2>Table With Invisible Borders</h2>

<p>Style the table with white borders and a background color of the cells to make the impression of invisible borders.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

</table>

</body>

</html>

## Round Table Borders

With the border-radius property, the borders get rounded corners:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**Example**

table, th, td {

  border: 1px solid black;

  border-radius: 10px;

}

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid black;

border-radius: 10px;

}

</style>

</head>

<body>

<h2>Table With Rounded Borders</h2>

<p>Use the CSS border-radius property to add rounded corners to the borders.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

</table>

</body>

</html>

Skip the border around the table by leaving out table from the css selector:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Example

th, td {

  border: 1px solid black;

  border-radius: 10px;

}

<!DOCTYPE html>

<html>

<head>

<style>

th, td {

border: 1px solid black;

border-radius: 10px;

}

</style>

</head>

<body>

<h2>Table With Rounded Borders</h2>

<p>Use the CSS border-radius property to add rounded corners to the table cells.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

</table>

</body>

</html>

## Dotted Table Borders

With the border-style property, you can set the appereance of the border.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

The following values are allowed:

* dotted
* dashed
* solid
* double
* groove
* ridge
* inset
* outset
* none
* hidden

**Example**

 th, td {

  border-style: dotted;

}

<!DOCTYPE html>

<html>

<head>

<style>

th, td {

border-style: dotted;

}

</style>

</head>

<body>

<h2>Table With Dotted Borders</h2>

<p>Use the CSS border-style property to set the style of the borders.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

</table>

</body>

</html>

## Border Color

With the border-color property, you can set the color of the border.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**Example**

 th, td {

  border-color: #96D4D4;

}

<!DOCTYPE html>

<html>

<head>

<style>

th, td {

border-style:solid;

border-color: #96D4D4;

}

</style>

</head>

<body>

<h2>Table With Border Color</h2>

<p>Use the CSS border-color property to set the color of the borders.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

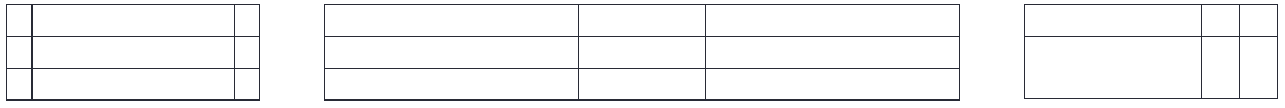
</table>

</body>

</html>

# HTML Table Sizes

HTML tables can have different sizes for each column, row or the entire table.



Use the style attribute with the width or height properties to specify the size of a table, row or column.

## HTML Table Width

To set the width of a table, add the style attribute to the <table> element:

**Example**

Set the width of the table to 100%:

<table**style**="width:100%">  
  <tr>  
    <th>Firstname</th>  
    <th>Lastname</th>  
    <th>Age</th>  
  </tr>  
  <tr>  
    <td>Jill</td>

    <td>Smith</td>  
    <td>50</td>  
  </tr>  
  <tr>  
    <td>Eve</td>  
    <td>Jackson</td>  
    <td>94</td>  
  </tr>  
</table>

Note: Using a percentage as the size unit for a width means how wide will this element be compared to its parent element, which in this case is the <body> element.

## HTML Table Column Width

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

To set the size of a specific column, add the style attribute on a <th> or <td> element:

**Example**

Set the width of the first column to 70%:

<table**style**="width:100%">  
  <tr>  
    <th**style**="width:70%">Firstname</th>  
    <th>Lastname</th>  
    <th>Age</th>  
  </tr>  
  <tr>  
    <td>Jill</td>

    <td>Smith</td>  
    <td>50</td>  
  </tr>  
  <tr>  
    <td>Eve</td>  
    <td>Jackson</td>  
    <td>94</td>  
  </tr>  
</table>

## HTML Table Row Height

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

To set the height of a specific row, add the style attribute on a table row element:

**Example**

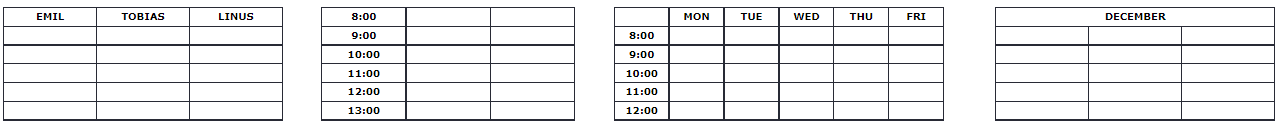
Set the height of the second row to 200 pixels:

<table**style**="width:100%">  
  <tr>  
    <th>Firstname</th>  
    <th>Lastname</th>  
    <th>Age</th>  
  </tr>  
  <tr**style**="height:200px">  
    <td>Jill</td>

    <td>Smith</td>  
    <td>50</td>  
  </tr>  
  <tr>  
    <td>Eve</td>  
    <td>Jackson</td>  
    <td>94</td>  
  </tr>  
</table>

# HTML Table Headers

HTML tables can have headers for each column or row, or for many columns/rows.



## HTML Table Headers

Table headers are defined with th elements, each th element represents a table cell.

**Example**

<table>  
  <tr>  
    <th>Firstname</th>  
    <th>Lastname</th>  
    <th>Age</th>  
  </tr>  
  <tr>  
    <td>Jill</td>  
    <td>Smith</td>  
    <td>50</td>  
  </tr>  
  <tr>  
    <td>Eve</td>  
    <td>Jackson</td>  
    <td>94</td>  
  </tr>  
</table>

## Vertical Table Headers

To use the first column as table headers, define the first cell in each row as a th element:

**Example**

<table>  
  <tr>  
    <th>Firstname</th>  
    <td>Jill</td>  
    <td>Eve</td>  
  </tr>  
  <tr>  
    <th>Lastname</th>

    <td>Smith</td>  
    <td>Jackson</td>  
  </tr>  
  <tr>  
    <th>Age</th>  
    <td>94</td>  
    <td>50</td>  
  </tr>  
</table>

## Align Table Headers

By default, table headers are bold and centered:

|  |  |  |
| --- | --- | --- |
| **Firstname** | **Lastname** | **Age** |
| Jill | Smith | 50 |
| Eve | Jackson | 94 |

To left-align the table headers, use the CSS text-align property:

**Example**

th {  
  text-align: left;  
}

## Header for Multiple Columns

You can have a header that spans over two or more columns.

|  |  |  |
| --- | --- | --- |
| **Name** | | **Age** |
| Jill | Smith | 50 |
| Eve | Jackson | 94 |

To do this, use the colspan attribute on the <th> element:

**Example**

<table>  
  <tr>  
    <th**colspan**="2">Name</th>  
    <th>Age</th>  
  </tr>  
  <tr>  
    <td>Jill</td>  
    <td>Smith</td>  
    <td>50</td>  
  </tr>  
  <tr>  
    <td>Eve</td>  
    <td>Jackson</td>  
    <td>94</td>  
  </tr>  
</table>

## Table Caption

You can add a caption that serves as a heading for the entire table.

|  |  |
| --- | --- |
| Monthly savings | |
| **Month** | **Savings** |
| January | $100 |
| February | $50 |

To add a caption to a table, use the <caption> tag:

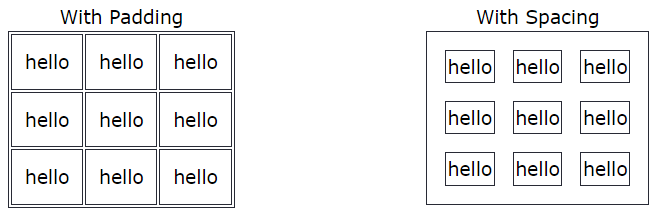
**Example**

<table**style**="width:100%">  
  <caption>Monthly savings</caption>

  <tr>  
    <th>Month</th>  
    <th>Savings</th>  
  </tr>  
  <tr>  
    <td>January</td>  
    <td>$100</td>  
  </tr>  
  <tr>  
    <td>February</td>  
    <td>$50</td>  
  </tr>  
</table>

# HTML Table Padding & Spacing

HTML tables can adjust the padding inside the cells, and also the space between the cells.



## HTML Table - Cell Padding

Cell padding is the space between the cell edges and the cell content.

By default the padding is set to 0.

To add padding on table cells, use the CSS padding property:

**Example**

th, td {

  padding: 15px;  
}

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid black;

border-collapse: collapse;

}

th, td {

padding: 15px;

}

</style>

</head>

<body>

<h2>Cellpadding</h2>

<p>Cell padding specifies the space between the cell content and its borders.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

</table>

<p><strong>Tip:</strong> Try to change the padding to 5px.</p>

</body>

</html>

To add padding only above the content, use the padding-top property.

And the others sides with the padding-bottom, padding-left, and padding-right properties:

**Example**

th, td {

  padding-top: 10px;  
  padding-bottom: 20px;  
  padding-left: 30px;  
  padding-right: 40px;  
}

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid black;

border-collapse: collapse;

}

th, td {

padding-top: 10px;

padding-bottom: 20px;

padding-left: 30px;

padding-right: 40px;

}

</style>

</head>

<body>

<h2>Cellpadding - top - bottom - left - right </h2>

<p>We can specify different padding for all fours sides of the cell content.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

</table>

</body>

</html>

## HTML Table - Cell Spacing

Cell spacing is the space between each cell.

By default the space is set to 2 pixels.

To change the space between table cells, use the CSS border-spacing property on the table element:

**Example**

table {  
  border-spacing: 30px;

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid black;

}

table {

border-spacing: 30px;

}

</style>

</head>

<body>

<h2>Cellspacing</h2>

<p>Change the space between the cells with the border-spacing property.</p>

<table style="width:100%">

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>94</td>

</tr>

<tr>

<td>John</td>

<td>Doe</td>

<td>80</td>

</tr>

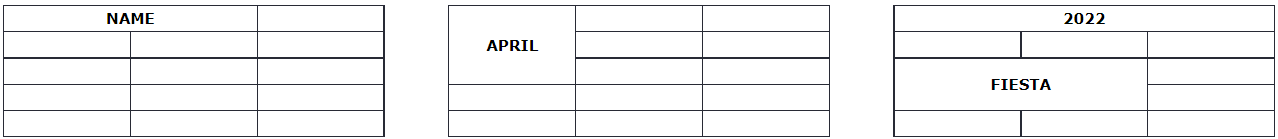
</table>

</body>

</html>

# HTML Table Colspan & Rowspan

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



## HTML Table - Colspan

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

**Example**

<table>  
  <tr>  
    <th**colspan**="2">Name</th>  
    <th>Age</th>  
  </tr>  
  <tr>  
    <td>Jill</td>  
    <td>Smith</td>  
    <td>43</td>  
  </tr>  
  <tr>  
    <td>Eve</td>  
    <td>Jackson</td>  
    <td>57</td>  
  </tr>  
</table>

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

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid black;

border-collapse: collapse;

}

</style>

</head>

<body>

<h2>Cell that spans two columns</h2>

<p>To make a cell span more than one column, use the colspan attribute.</p>

<table style="width:100%">

<tr>

<th colspan="2">Name</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Smith</td>

<td>43</td>

</tr>

<tr>

<td>Eve</td>

<td>Jackson</td>

<td>57</td>

</tr>

</table>

</body>

</html>

## HTML Table - Rowspan

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

**Example**

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

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

<!DOCTYPE html>

<html>

<head>

<style>

table, th, td {

border: 1px solid black;

border-collapse: collapse;

}

</style>

</head>

<body>

<h2>Cell that spans two rows</h2>

<p>To make a cell span more than one row, use the rowspan attribute.</p>

<table style="width:100%">

<tr>

<th>Name</th>

<td>Jill</td>

</tr>

<tr>

<th rowspan="2">Phone</th>

<td>555-1234</td>

</tr>

<tr>

<td>555-8745</td>

</tr>

</table>

</body>

</html>

# HTML Lists

HTML lists allow web developers to group a set of related items in lists.

**Example**

An unordered HTML list:

* Item
* Item
* Item
* Item

An ordered HTML list:

1. First item
2. Second item
3. Third item
4. Fourth item

## Unordered HTML List

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.

The list items will be marked with bullets (small black circles) by default:

**Example**

<ul>  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ul>

## Ordered HTML List

An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.

The list items will be marked with numbers by default:

**Example**

<ol>  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ol>

## HTML Description Lists

HTML also supports description lists.

A description list is a list of terms, with a description of each term.

The <dl> tag defines the description list, the <dt> tag defines the term (name), and the <dd> tag describes each term:

**Example**

<dl>  
  <dt>Coffee</dt>  
  <dd>- black hot drink</dd>

  <dt>Milk</dt>  
  <dd>- white cold drink</dd>

</dl>

## HTML List Tags

|  |  |
| --- | --- |
| **Tag** | **Description** |
| <ul> | Defines an unordered list |
| <ol> | Defines an ordered list |
| <li> | Defines a list item |
| <dl> | Defines a description list |
| <dt> | Defines a term in a description list |
| <dd> | Describes the term in a description list |

# HTML Block & Inline Elements

Every HTML element has a default display value, depending on what type of element it is.

There are two display values: block and inline.

## Block-level Elements

A block-level element always starts on a new line.

A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

A block level element has a top and a bottom margin, whereas an inline element does not.

The <div> element is a block-level element.

**Example**

<div>Hello World</div>

Here are the block-level elements in HTML:

[<address>](https://www.w3schools.com/tags/tag_address.asp) [<article>](https://www.w3schools.com/tags/tag_article.asp) [<aside>](https://www.w3schools.com/tags/tag_aside.asp) [<blockquote>](https://www.w3schools.com/tags/tag_blockquote.asp) [<canvas>](https://www.w3schools.com/tags/tag_canvas.asp) [<dd>](https://www.w3schools.com/tags/tag_dd.asp)

[<div>](https://www.w3schools.com/tags/tag_div.asp) [<dl>](https://www.w3schools.com/tags/tag_dl.asp) [<dt>](https://www.w3schools.com/tags/tag_dt.asp) [<fieldset>](https://www.w3schools.com/tags/tag_fieldset.asp) [<figcaption>](https://www.w3schools.com/tags/tag_figcaption.asp) [<figure>](https://www.w3schools.com/tags/tag_figure.asp) [<footer>](https://www.w3schools.com/tags/tag_footer.asp)

[<form>](https://www.w3schools.com/tags/tag_form.asp) [<h1>-<h6>](https://www.w3schools.com/tags/tag_hn.asp) [<header>](https://www.w3schools.com/tags/tag_header.asp) [<hr>](https://www.w3schools.com/tags/tag_hr.asp) [<li>](https://www.w3schools.com/tags/tag_li.asp) [<main>](https://www.w3schools.com/tags/tag_main.asp) [<nav>](https://www.w3schools.com/tags/tag_nav.asp) [<noscript>](https://www.w3schools.com/tags/tag_noscript.asp)

[<ol>](https://www.w3schools.com/tags/tag_ol.asp) [<p>](https://www.w3schools.com/tags/tag_p.asp) [<pre>](https://www.w3schools.com/tags/tag_pre.asp) [<section>](https://www.w3schools.com/tags/tag_section.asp) [<table>](https://www.w3schools.com/tags/tag_table.asp) [<tfoot>](https://www.w3schools.com/tags/tag_tfoot.asp) [<ul>](https://www.w3schools.com/tags/tag_ul.asp) [<video>](https://www.w3schools.com/tags/tag_video.asp)

## Inline Elements

An inline element does not start on a new line.

An inline element only takes up as much width as necessary.

This is a <span> element inside a paragraph.

**Example**

<span>Hello World</span>

Here are the inline elements in HTML:

[<a>](https://www.w3schools.com/tags/tag_a.asp) [<abbr>](https://www.w3schools.com/tags/tag_abbr.asp) [<acronym>](https://www.w3schools.com/tags/tag_acronym.asp) [<b>](https://www.w3schools.com/tags/tag_b.asp) [<bdo>](https://www.w3schools.com/tags/tag_bdo.asp) [<big>](https://www.w3schools.com/tags/tag_big.asp) [<br>](https://www.w3schools.com/tags/tag_br.asp) [<button>](https://www.w3schools.com/tags/tag_button.asp) [<cite>](https://www.w3schools.com/tags/tag_cite.asp)

[<code>](https://www.w3schools.com/tags/tag_code.asp) [<dfn>](https://www.w3schools.com/tags/tag_dfn.asp) [<em>](https://www.w3schools.com/tags/tag_em.asp) [<i>](https://www.w3schools.com/tags/tag_i.asp) [<img>](https://www.w3schools.com/tags/tag_img.asp) [<input>](https://www.w3schools.com/tags/tag_input.asp) [<kbd>](https://www.w3schools.com/tags/tag_kbd.asp) [<label>](https://www.w3schools.com/tags/tag_label.asp) [<map>](https://www.w3schools.com/tags/tag_map.asp)

[<object>](https://www.w3schools.com/tags/tag_object.asp) [<output>](https://www.w3schools.com/tags/tag_output.asp) [<q>](https://www.w3schools.com/tags/tag_q.asp) [<samp>](https://www.w3schools.com/tags/tag_samp.asp) [<script>](https://www.w3schools.com/tags/tag_script.asp) [<select>](https://www.w3schools.com/tags/tag_select.asp) [<small>](https://www.w3schools.com/tags/tag_small.asp)

[<span>](https://www.w3schools.com/tags/tag_span.asp) [<strong>](https://www.w3schools.com/tags/tag_strong.asp) [<sub>](https://www.w3schools.com/tags/tag_sub.asp)[<sup>](https://www.w3schools.com/tags/tag_sup.asp) [<textarea>](https://www.w3schools.com/tags/tag_textarea.asp) [<time>](https://www.w3schools.com/tags/tag_time.asp) [<tt>](https://www.w3schools.com/tags/tag_tt.asp) [<var>](https://www.w3schools.com/tags/tag_var.asp)

Note: An inline element cannot contain a block-level element!

## The <div> Element

The <div> element is often used as a container for other HTML elements.

The <div> element has no required attributes, but style, class and id are common.

When used together with CSS, the <div> element can be used to style blocks of content:

**Example**

<div**style**="background-color:black;color:white;padding:20px;">  
  <h2>London</h2>  
  <p>London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>  
</div>

## The <span> Element

The <span> element is an inline container used to mark up a part of a text, or a part of a document.

The <span> element has no required attributes, but style, class and id are common.

When used together with CSS, the <span> element can be used to style parts of the text:

**Example**

<p>My mother has <span**style**="color:blue;font-weight:bold">blue</span> eyes and my father has <span**style**="color:darkolivegreen;font-weight:bold">dark green</span> eyes.</p>

# HTML Classes

The HTML class attribute is used to specify a class for an HTML element.

Multiple HTML elements can share the same class.

## Using The class Attribute

The class attribute is often used to point to a class name in a style sheet. It can also be used by a JavaScript to access and manipulate elements with the specific class name.

In the following example we have three <div> elements with a class attribute with the value of "city". All of the three <div> elements will be styled equally according to the .city style definition in the head section:

**Example**

<!DOCTYPE**html**>  
<html>  
<head>  
<style>  
.city {  
  background-color: tomato;  
  color: white;

  border: 2px solid black;

  margin: 20px;

  padding: 20px;

}  
</style>  
</head>  
<body>  
<div**class**="city">  
  <h2>London</h2>  
  <p>London is the capital of England.</p>

</div>  
<div**class**="city">  
  <h2>Paris</h2>  
  <p>Paris is the capital of France.</p>

</div>  
<div**class**="city">  
  <h2>Tokyo</h2>  
  <p>Tokyo is the capital of Japan.</p>

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

In the following example we have two <span> elements with a class attribute with the value of "note". Both <span> elements will be styled equally according to the .note style definition in the head section:

**Example**

<!DOCTYPE**html**>  
<html>  
<head>  
<style>  
.note {  
  font-size: 120%;

  color: red;  
}  
</style>  
</head>  
<body>  
<h1>My <span**class**="note">Important</span> Heading</h1>  
<p>This is some <span**class**="note">important</span> text.</p>

</body>  
</html>

Tip: The class attribute can be used on any HTML element.

Note: The class name is case sensitive!

Tip: You can learn much more about CSS in our [CSS Tutorial](https://www.w3schools.com/css/default.asp).

## The Syntax for Class

To create a class; write a period (.) character, followed by a class name. Then, define the CSS properties within curly braces {}:

**Example**

Create a class named "city":

<!DOCTYPE**html**>  
<html>  
<head>  
<style>  
.city {  
  background-color: tomato;

  color: white;

  padding: 10px;

}  
</style>  
</head>  
<body>  
<h2**class**="city">London</h2>  
<p>London is the capital of England.</p>

<h2**class**="city">Paris</h2>  
<p>Paris is the capital of France.</p>

<h2**class**="city">Tokyo</h2>  
<p>Tokyo is the capital of Japan.</p>

</body>  
</html>

## Multiple Classes

HTML elements can belong to more than one class.

To define multiple classes, separate the class names with a space, e.g. <div class="city main">. The element will be styled according to all the classes specified.

In the following example, the first <h2> element belongs to both the city class and also to the main class, and will get the CSS styles from both of the classes:

**Example**

<h2**class**="city main">London</h2>

<h2**class**="city">Paris</h2>  
<h2**class**="city">Tokyo</h2>

## Different Elements Can Share Same Class

Different HTML elements can point to the same class name.

In the following example, both <h2> and <p> points to the "city" class and will share the same style:

**Example**

<h2**class**="city">Paris</h2>  
<p**class**="city">Paris is the capital of France</p>

## Use of the class Attribute in JavaScript

The class name can also be used by JavaScript to perform certain tasks for specific elements.

JavaScript can access elements with a specific class name with the getElementsByClassName() method:

**Example**

Click on a button to hide all elements with the class name "city":

<script>  
function myFunction() {

  var x = document.**getElementsByClassName**(**"city"**);

  for (var i = 0; i < x.length; i++) {

    x[i].style.display = "none";

  }

}  
</script>

<!DOCTYPE html>

<html>

<body>

<h2>Use of The class Attribute in JavaScript</h2>

<p>Click the button to hide all elements with class name "city":</p>

<button onclick="myFunction()">Hide elements</button>

<h2 class="city">London</h2>

<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>

<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>

<p>Tokyo is the capital of Japan.</p>

<script>

function myFunction() {

var x = document.getElementsByClassName("city");

for (var i = 0; i < x.length; i++) {

x[i].style.display = "none";

}

}

</script>

</body>

</html>

# HTML Id

The HTML id attribute is used to specify a unique id for an HTML element.

You cannot have more than one element with the same id in an HTML document.

## Using The id Attribute

The id attribute specifies a unique id for an HTML element. The value of the id attribute must be unique within the HTML document.

The id attribute is used to point to a specific style declaration in a style sheet. It is also used by JavaScript to access and manipulate the element with the specific id.

The syntax for id is: write a hash character (#), followed by an id name. Then, define the CSS properties within curly braces {}.

In the following example we have an <h1> element that points to the id name "myHeader". This <h1> element will be styled according to the #myHeader style definition in the head section:

**Example**

<!DOCTYPE html>  
<html>  
<head>  
<style>  
#myHeader {  
  background-color: lightblue;

  color: black;

  padding: 40px;

  text-align: center;

}  
</style>  
</head>  
<body>  
<h1 id="myHeader">My Header</h1>

</body>  
</html>

**Note:** The id name is case sensitive!

**Note:** The id name must contain at least one character, cannot start with a number, and must not contain whitespaces (spaces, tabs, etc.).

## Difference Between Class and ID

A class name can be used by multiple HTML elements, while an id name must only be used by one HTML element within the page:

**Example**

<!DOCTYPE html>

<html>

<head>

<style>  
/\* Style the element with the id "myHeader" \*/

**#myHeader** {  
  background-color: lightblue;

  color: black;

  padding: 40px;  
  text-align: center;  
}  
  
/\* Style all elements with the class name "city" \*/

**.city**{  
  background-color: tomato;

  color: white;  
  padding: 10px;  
}  
</style>

</head>

<body>

<h2>Difference Between Class and ID</h2>

<p>A class name can be used by multiple HTML elements, while an id name must only be used by one HTML element within the page:</p>  
<!-- An element with a unique id -->

<h1 id="myHeader">My Cities</h1>

<!-- Multiple elements with same class -->

<h2 class="city">London</h2>  
<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>  
<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>  
<p>Tokyo is the capital of Japan.</p>

</body>

</html>

## HTML Bookmarks with ID and Links

HTML bookmarks are used to allow readers to jump to specific parts of a webpage.

Bookmarks can be useful if your page is very long.

To use a bookmark, you must first create it, and then add a link to it.

Then, when the link is clicked, the page will scroll to the location with the bookmark.

**Example**

First, create a bookmark with the id attribute:

<h2 id="C4">Chapter 4</h2>

Then, add a link to the bookmark ("Jump to Chapter 4"), from within the same page:

**Example**

<a href="#C4">Jump to Chapter 4</a>

Or, add a link to the bookmark ("Jump to Chapter 4"), from another page:

<a href="html\_demo.html#C4">Jump to Chapter 4</a>

## Using The id Attribute in JavaScript

The id attribute can also be used by JavaScript to perform some tasks for that specific element.

JavaScript can access an element with a specific id with the getElementById() method:

**Example**

Use the id attribute to manipulate text with JavaScript:

<script>  
function displayResult() {

  document.getElementById("myHeader").innerHTML = "Have a nice day!";

}  
</script>

<!DOCTYPE html>

<html>

<body>

<h2>Using The id Attribute in JavaScript</h2>

<p>JavaScript can access an element with a specified id by using the getElementById() method:</p>

<h1 id="myHeader">Hello World!</h1>

<button onclick="displayResult()">Change text</button>

<script>

function displayResult() {

document.getElementById("myHeader").innerHTML = "Have a nice day!";

}

</script>

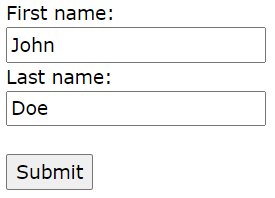
</body>

</html>

# HTML Forms

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

**Example**



<!DOCTYPE html>

<html>

<body>

<h2>HTML Forms</h2>

<form action="/action\_page.php">

<label for="fname">First name:</label><br>

<input type="text" id="fname" name="fname" value="John"><br>

<label for="lname">Last name:</label><br>

<input type="text" id="lname" name="lname" value="Doe"><br><br>

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

</form>

<p>If you click the "Submit" button, the form-data will be sent to a page called "/action\_page.php".</p>

</body>

</html>

## The <form> Element

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

<form>  
….  
*form elements*

….

</form>

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

## The <input> Element

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

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

Here are some examples:

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

## Text Fields

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

**Example**

A form with input fields for text:

<!DOCTYPE html>

<html>

<body>

<h2>Text input fields</h2>

<form>  
  <label for="fname">First name:</label><br>

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

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

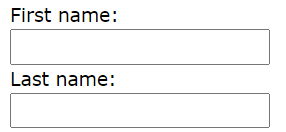
<p>Note that the form itself is not visible.</p>

<p>Also note that the default width of text input fields is 20 characters.</p>

</body>

</html>

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



Top of Form

Bottom of Form

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

## The <label> Element

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

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

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

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

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

## Radio Buttons

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

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

**Example**

A form with radio buttons:

<!DOCTYPE html>

<html>

<body>

<h2>Radio Buttons</h2>

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

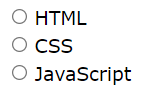
<form>  
  <input type="radio" id="html" name="fav\_language" value="HTML">  
  <label for="html">HTML</label><br>  
  <input type="radio" id="css" name="fav\_language" value="CSS">  
  <label for="css">CSS</label><br>  
  <input type="radio" id="javascript" name="fav\_language" value="JavaScript"> 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

</body>

</html>

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

Choose your favorite Web language:



## Checkboxes

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

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

**Example**

A form with checkboxes:

<!DOCTYPE html>

<html>

<body>

<h2>Checkboxes</h2>

<p>The <strong>input type="checkbox"</strong> defines a checkbox:</p>

<form>  
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">  
  <label for="vehicle1"> I have a bike</label><br>

  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">  
  <label for="vehicle2"> I have a car</label><br>

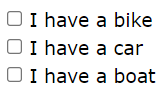
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">  
  <label for="vehicle3"> I have a boat</label>

</form>

</body>

</html>

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



## The Submit Button

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

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

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

**Example**

A form with a submit button:

<!DOCTYPE html>

<html>

<body>

<h2>HTML Forms</h2>

<form action="/action\_page.php">  
  <label for="fname">First name:</label><br>

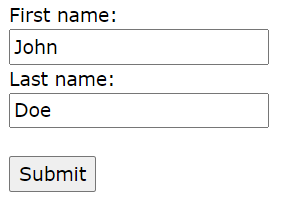
  <input type="text" id="fname" name="fname" value="John"><br>  
  <label for="lname">Last name:</label><br>

  <input type="text" id="lname" name="lname" value="Doe"><br><br>  
  <input type="submit" value="Submit">  
</form>

</body>

</html>

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



**action\_page.php:**

<html>  
<body>  
Welcome <?php echo $\_POST["fname"]; ?><br>

Lastname: <?php echo $\_POST["lname"]; ?><br>

</body>  
</html>

## The Name Attribute for <input>

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

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

**Example**

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

<!DOCTYPE html>

<html>

<body>

<h2>The name Attribute</h2>

<form action="/action\_page.php">  
  <label for="fname">First name:</label><br>

  <input type="text" id="fname" value="John"><br><br>  
  <input type="submit" value="Submit">  
</form>

<p>If you click the "Submit" button, the form-data will be sent to a page called "/action\_page.php".</p>

<p>Notice that the value of the "First name" field will not be submitted, because the input element does not have a name attribute.</p>

</body>

</html>

# HTML Form Attributes

## The Action Attribute

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

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

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

**Example**

On submit, send form data to "action\_page.php":

<!DOCTYPE html>

<html>

<body>

<h2>HTML Forms</h2>

<form action="/action\_page.php">  
  <label for="fname">First name:</label><br>

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

  <input type="text" id="lname" name="lname" value="Doe"><br><br>  
  <input type="submit" value="Submit">  
</form>

<p>If you click the "Submit" button, the form-data will be sent to a page called "/action\_page.php".</p>

</body>

</html>

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

## The Target Attribute

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

The target attribute can have one of the following values:

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

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

**Example**

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

<!DOCTYPE html>

<html>

<body>

<h2>The form target attribute</h2>

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

<form action="/action\_page.php" target="\_blank">

<label for="fname">First name:</label><br>

<input type="text" id="fname" name="fname" value="John"><br>

<label for="lname">Last name:</label><br>

<input type="text" id="lname" name="lname" value="Doe"><br><br>

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

</form>

</body>

</html>

## The Method Attribute

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

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

The default HTTP method when submitting form data is GET.

**Example**

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

<!DOCTYPE html>

<html>

<body>

<h2>The method Attribute</h2>

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

<form action="/action\_page.php" target="\_blank" method="get">

<label for="fname">First name:</label><br>

<input type="text" id="fname" name="fname" value="John"><br>

<label for="lname">Last name:</label><br>

<input type="text" id="lname" name="lname" value="Doe"><br><br>

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

</form>

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

</body>

</html>

**Example**

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

<!DOCTYPE html>

<html>

<body>

<h2>The method Attribute</h2>

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

<form action="/action\_page.php" target="\_blank" method="post">

<label for="fname">First name:</label><br>

<input type="text" id="fname" name="fname" value="John"><br>

<label for="lname">Last name:</label><br>

<input type="text" id="lname" name="lname" value="Doe"><br><br>

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

</form>

<p>After you submit, notice that, unlike the GET method, the form values is NOT visible in the address bar of the new browser tab.</p>

</body>

</html>

Notes on GET:

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

Notes on POST:

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

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

## The Autocomplete Attribute

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

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

**Example**

A form with autocomplete on:

<!DOCTYPE html>

<html>

<body>

<h1>The form autocomplete attribute</h1>

<p>Fill in and submit the form, then reload the page, start to fill in the form again - and see how autocomplete works.</p>

<p>Then, try to set autocomplete to "off".</p>

<form action="/action\_page.php" autocomplete="on">

<label for="fname">First name:</label>

<input type="text" id="fname" name="fname"><br><br>

<label for="email">Email:</label>

<input type="text" id="email" name="email"><br><br>

<input type="submit">

</form>

</body>

</html>

## The Novalidate Attribute

The novalidate attribute is a boolean attribute.

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

**Example**

A form with a novalidate attribute:

<!DOCTYPE html>

<html>

<body>

<h1>The form novalidate attribute</h1>

<p>The novalidate attribute indicates that the form input is not to be validated on submit:</p>

<form action="/action\_page.php" novalidate>

<label for="email">Enter your email:</label>

<input type="email" id="email" name="email"><br><br>

<input type="submit">

</form>

<p><strong>Note:</strong> The novalidate attribute of the form tag is not supported in Safari 10 (or earlier).</p>

</body>

</html>

# HTML Form Elements

## The HTML <form> Elements

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

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

## The <input> Element

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

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

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>The input Element</h2>

<form action="/action\_page.php">

<label for="fname">First name:</label><br>

<input type="text" id="fname" name="fname"><br><br>

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

</form>

</body>

</html>

## The <label> Element

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

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

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

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

## The <select> Element

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

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>The select Element</h2>

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

<form action="/action\_page.php">

<label for="cars">Choose a car:</label>

<select id="cars" name="cars">

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

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

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

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

</select>

<input type="submit">

</form>

</body>

</html>

The <option> elements defines an option that can be selected.

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

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

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>Pre-selected Option</h2>

<p>You can preselect an option with the selected attribute:</p>

<form action="/action\_page.php">

<label for="cars">Choose a car:</label>

<select id="cars" name="cars">

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

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

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

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

</select>

<input type="submit">

</form>

</body>

## Visible Values:

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

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>Visible Option Values</h2>

<p>Use the size attribute to specify the number of visible values.</p>

<form action="/action\_page.php">

<label for="cars">Choose a car:</label>

<select id="cars" name="cars" size="3">

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

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

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

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

</select><br><br>

<input type="submit">

</form>

</body>

</html>

## Allow Multiple Selections:

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

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>Allow Multiple Selections</h2>

<p>Use the multiple attribute to allow the user to select more than one value.</p>

<form action="/action\_page.php">

<label for="cars">Choose a car:</label>

<select id="cars" name="cars" size="4" multiple>

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

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

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

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

</select><br><br>

<input type="submit">

</form>

<p>Hold down the Ctrl (windows) / Command (Mac) button to select multiple options.</p>

</body>

</html>

## The <textarea> Element

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

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>Textarea</h2>

<p>The textarea element defines a multi-line input field.</p>

<form action="/action\_page.php">

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

<br><br>

<input type="submit">

</form>

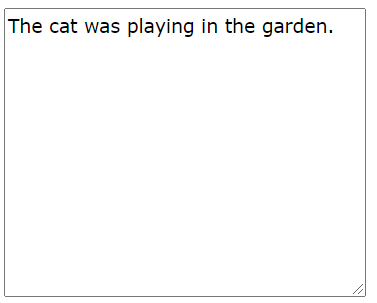
</body>

</html>

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

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

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



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

**Example**

<textarea name="message" style="width:200px; height:600px;">

The cat was playing in the garden.

</textarea>

## The <button> Element

The <button> element defines a clickable button:

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>The button Element</h2>

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

</body>

</html>

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

## The <fieldset> and <legend> Elements

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

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

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>Grouping Form Data with Fieldset</h2>

<p>The fieldset element is used to group related data in a form, and the legend element defines a caption for the fieldset element.</p>

<form action="/action\_page.php">

<fieldset>

<legend>Personalia:</legend>

<label for="fname">First name:</label><br>

<input type="text" id="fname" name="fname" value="John"><br>

<label for="lname">Last name:</label><br>

<input type="text" id="lname" name="lname" value="Doe"><br><br>

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

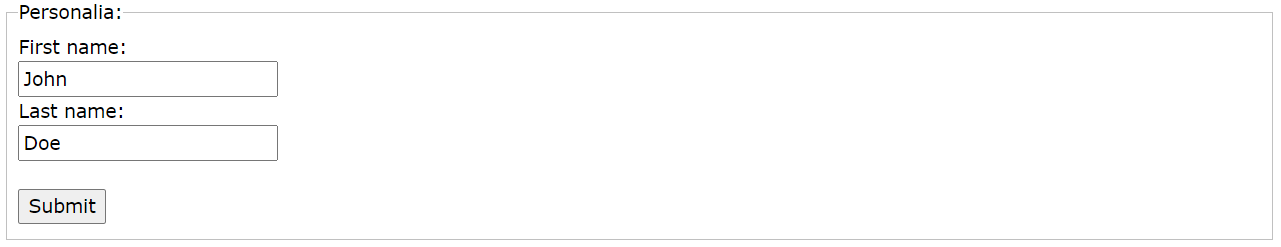
</fieldset>

</form>

</body>

</html>

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



## The <datalist> Element

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

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

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

**Example**

<!DOCTYPE html>

<html>

<body>

<h2>The datalist Element</h2>

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

<form action="/action\_page.php">

<input list="browsers" name="browser">

<datalist id="browsers">

<option value="Internet Explorer">

<option value="Firefox">

<option value="Chrome">

<option value="Opera">

<option value="Safari">

</datalist>

<input type="submit">

</form>

<p><b>Note:</b> The datalist tag is not supported in Safari prior version 12.1.</p>

</body>

</html>

## The <output> Element

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

**Example**

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

<!DOCTYPE html>

<html>

<body>

<h2>The output Element</h2>

<p>The output element represents the result of a calculation.</p>

<form action="/action\_page.php"

oninput="x.value=parseInt(a.value)+parseInt(b.value)">

0

<input type="range" id="a" name="a" value="50">

100 +

<input type="number" id="b" name="b" value="50">

=

<output name="x" for="a b"></output>

<br><br>

<input type="submit">

</form>

<p><strong>Note:</strong> The output element is not supported in Edge prior version 13.</p>

</body>

</html>

# HTML Input Types

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

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

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

## Input Type Text

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

**Example**

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

## Input Type Password

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

**Example**

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

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

## Input Type Submit

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

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

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

**Example**

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

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

**Example**

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

## Input Type Reset

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

**Example**

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

Bottom of Form

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

## Input Type Radio

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

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

**Example**

<p>Choose your favorite Web language:</p>  
  
<form>  
  <input type="radio" id="html" name="fav\_language" value="HTML">  
  <label for="html">HTML</label><br>  
  <input type="radio" id="css" name="fav\_language" value="CSS">  
  <label for="css">CSS</label><br>  
  <input type="radio" id="javascript" name="fav\_language" value="JavaScript">  
  <label for="javascript">JavaScript</label>  
</form>

## Input Type Checkbox

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

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

**Example**

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

## Input Type Button

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

**Example**

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

## Input Type Color

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

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

**Example**

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

## Input Type Date

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

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

**Example**

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

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

**Example**

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

## Input Type Datetime-local

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

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

**Example**

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

## Input Type Email

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

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

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

**Example**

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

## Input Type File

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

**Example**

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

## Input Type Hidden

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

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

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

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

**Example**

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

## Input Type Month

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

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

**Example**

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

## Input Type Number

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

You can also set restrictions on what numbers are accepted.

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

**Example**

<form>  
  <label for="quantity">Quantity (between 1 and 5):</label>  
  <input type="number" id="quantity" name="quantity" min="1" max="5">  
</form>

## Input Restrictions

Here is a list of some common input restrictions:

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

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

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

**Example**

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

## Input Type Range

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

**Example**

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

## Input Type Search

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

**Example**

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

## Input Type Tel

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

**Example**

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

## Input Type Time

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

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

**Example**

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

## Input Type Url

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

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

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

**Example**

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

## Input Type Week

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

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

**Example**

<form>  
  <label for="week">Select a week:</label>

  <input type="week" id="week" name="week">  
</form>

# HTML Input Attributes

## The value Attribute

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

**Example**

Input fields with initial (default) values:

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

## The readonly Attribute

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

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

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

**Example**

A read-only input field:

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

## The disabled Attribute

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

A disabled input field is unusable and un-clickable.

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

**Example**

A disabled input field:

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

## The size Attribute

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

The default value for size is 20.

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

**Example**

Set a width for an input field:

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

## The maxlength Attribute

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

**Note:** When a maxlength is set, the input field will not accept more than the specified number of characters. However, this attribute does not provide any feedback. So, if you want to alert the user, you must write JavaScript code.

**Example**

Set a maximum length for an input field:

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

## The min and max Attributes

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

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

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

**Example**

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

<form>  
  <label for="datemax">Enter a date before 1980-01-01:</label>  
  <input type="date" id="datemax" name="datemax" max="1979-12-31"><br><br>  
  
  <label for="datemin">Enter a date after 2000-01-01:</label>  
  <input type="date" id="datemin" name="datemin" min="2000-01-02"><br><br>  
  
  <label for="quantity">Quantity (between 1 and 5):</label>  
  <input type="number" id="quantity" name="quantity" min="1" max="5">  
</form>

## The multiple Attribute

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

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

**Example**

A file upload field that accepts multiple values:

<!DOCTYPE html>

<html>

<body>

<h1>The input multiple attributes</h1>

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

<form action="/action\_page.php">

<label for="files">Select files:</label>

<input type="file" id="files" name="files" multiple><br><br>

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

</form>

<p>To select multiple files, hold down the CTRL or SHIFT key while selecting.</p>

</body>

</html>

## The pattern Attribute

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

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

**Tip:** Use the global title attribute to describe the pattern to help the user.

**Tip:** Learn more about regular expressions in our JavaScript tutorial.

**Example**

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

<!DOCTYPE html>

<html>

<body>

<h1>The input pattern attribute</h1>

<p>The pattern attribute specifies a regular expression that the input element's value is checked against.</p>

<form action="/action\_page.php">

<label for="country\_code">Country code:</label>

<input type="text" id="country\_code" name="country\_code" pattern="[A-Za-z]{3}" title="Three letter country code"><br><br>

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

</form>

<p><strong>Note:</strong> The pattern attribute of the input tag is not supported in Safari 10 (or earlier).</p>

</body>

</html

## The placeholder Attribute

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

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

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

**Example**

An input field with a placeholder text:

<form>  
  <label for="phone">Enter a phone number:</label>

  <input type="tel" id="phone" name="phone" placeholder="123-45-678"

pattern="[0-9]{3}-[0-9]{2}-[0-9]{3}">  
 <input type="submit" value="Submit">

</form>

## The required Attribute

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

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

**Example**

A required input field:

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

</form>

## The step Attribute

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

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

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

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

**Example**

An input field with a specified legal number of intervals:

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

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

</form>

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

## The autofocus Attribute

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

**Example**

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

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

</form>

## The height and width Attributes

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

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

**Example**

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

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

## The list Attribute

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

**Example**

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

<!DOCTYPE html>

<html>

<body>

<h1>The input list attribute</h1>

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

<form action="/action\_page.php">

<input list="browsers" name="browser">

<datalist id="browsers">

<option value="Internet Explorer">

<option value="Firefox">

<option value="Chrome">

<option value="Opera">

<option value="Safari">

</datalist>

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

</form>

<p><b>Note:</b> The datalist tag is not supported in Safari 12.0 (or earlier).</p>

</body>

</html>

## The autocomplete Attribute

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

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

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

**Example**

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

<form action="/action\_page.php" autocomplete="on">  
  <label for="fname">First name:</label>  
  <input type="text" id="fname" name="fname"><br><br>  
  <label for="lname">Last name:</label>

  <input type="text" id="lname" name="lname"><br><br>  
  <label for="email">Email:</label>  
  <input type="email" id="email" name="email" autocomplete="off"><br><br>  
  <input type="submit" value="Submit">  
</form>

# HTML Input Form Attributes

## The form Attribute

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

The value of this attribute must be equal to the id attribute of the <form> element it belongs to.

**Example**

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

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

## The formaction Attribute

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

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

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

**Example**

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

<p>The formaction attribute specifies the URL of a file that will process the input when the form is submitted.</p>

<form action="/action\_page.php">  
  <label for="fname">First name:</label>  
  <input type="text" id="fname" name="fname"><br><br>  
  <label for="lname">Last name:</label>  
  <input type="text" id="lname" name="lname"><br><br>  
  <input type="submit" value="Submit">  
  <input type="submit" formaction="/action\_page2.php" value="Submit as Admin">  
</form>

## ~~The formenctype Attribute~~

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

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

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

**~~Example~~**

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

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

## The formmethod Attribute

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

**Note:** This attribute overrides the method attribute of the <form> element.

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

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

**Notes on the "get" method:**

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

**Notes on the "post" method:**

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

**Example**

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

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

## The formtarget Attribute

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

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

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

**Example**

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

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

## The formnovalidate Attribute

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

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

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

**Example**

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

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

</form>

## The novalidate Attribute

The novalidate attribute is a <form> attribute.

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

**Example**

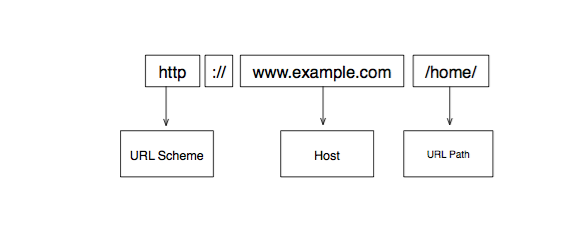
Specify that no form-data should be validated on submit:

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

# HTML Query String

When you see a URL, such as "<http://www.example.com:88/home?item=book>", it is comprised of several components. We can break this URL into 5 parts:

* http: The **scheme**. It always comes before the colon and two forward slashes and tells the web client how to access the resource. In this case it tells the web client to use the Hypertext Transfer Protocol or HTTP to make a request. Other popular URL schemes are ftp, mailto or git. You may sometimes see this part of the URL referred to as the protocol, and there is a connection between the two things in that the scheme can indicate which protocol (or system of rules) should be used to access the resource; in the context of of a URL however, the correct term for this component is the scheme.
* www.example.com: The **host**. It tells the client where the resource is hosted or located.
* :88 : The **port** or port number. It is only required if you want to use a port other than the default.
* /home/: The **path**. It shows what local resource is being requested. This part of the URL is optional.
* ?item=book : The **query string**, which is made up of **query parameters**. It is used to send data to the server. This part of the URL is also optional.



Sometimes, the path can point to a specific resource on the host. For instance, [www.example.com/home/index.html](http://www.example.com/home/index.html) points to an HTML file located on the example.com server.

Sometimes, we may want to include a port number which the host uses to listen to HTTP requests. A URL in the form of: http://localhost:3000/profile is using the port number 3000 to listen to HTTP requests. The default port number for HTTP is port 80. Even though this port number is not always specified, it's assumed to be part of every URL. **Unless a different port number is specified, port 80 will be used by default in normal HTTP requests.** To use anything other than the default, one has to specify it in the URL.

## [Query Strings/Parameters](https://launchschool.com/books/http/read/what_is_a_url" \l "querystringsparameters)

A simple URL with a query string might look like:

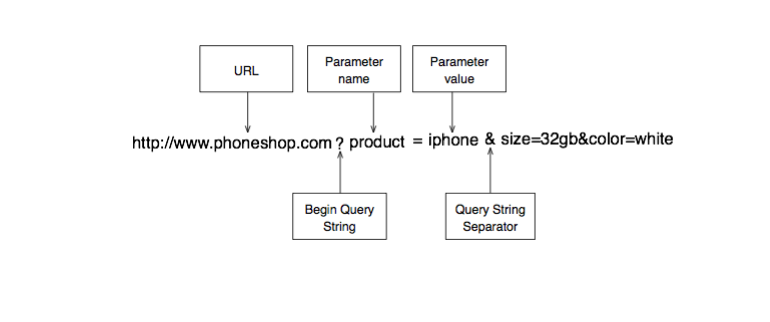
**http://www.example.com?search=ruby&results=10**

Let's take that apart:

|  |  |
| --- | --- |
| **Query String Component** | **Description** |
| ? | This is a reserved character that marks the start of the query string |
| search=ruby | This is a parameter name/value pair. |
| & | This is a reserved character, used when adding more parameters to the query string. |
| results=10 | This is also a parameter name/value pair. |

Now let's take a look at an example. Suppose we had the following URL:

**http://www.phoneshop.com?product=iphone&size=32gb&color=white**



In the above example, name/value pairs in the form of product=iphone, size=32gb and color=white are passed to the server from the URL. This is asking the www.phoneshop.com server to narrow down on a product iphone, size 32gb and color white. How the server uses these parameters is up to the server side application.

**Because query strings are passed in through the URL, they are only used in HTTP GET requests.** We'll talk about the different HTTP requests later in the book, but for now just know that whenever you type in a URL into the address bar of your browser, you're issuing HTTP GET requests. Most links also issue HTTP GET requests, though there are some minor exceptions.

Query strings are great to pass in additional information to the server, however, there are some limits to the use of query strings:

* Query strings have a maximum length. Therefore, if you have a lot of data to pass on, you will not be able to do so with query strings.
* The name/value pairs used in query strings are visible in the URL. For this reason, passing sensitive information like username or password to the server in this manner is not recommended.
* Space and special characters like & cannot be used with query strings. They must be URL encoded, which we'll talk about next.

## [URL Encoding](https://launchschool.com/books/http/read/what_is_a_url" \l "urlencoding)

URLs are designed to accept only certain characters in the standard 128-character [ASCII character set](http://en.wikipedia.org/wiki/ASCII). Reserved or unsafe ASCII characters which are not being used for their intended purpose, as well as characters not in this set, have to be encoded. URL encoding serves the purpose of replacing these non-conforming characters with a % symbol followed by two hexadecimal digits that represent the [ASCII code](http://www.asciitable.com/) of the character.

Below are some popular encoded characters and example URLs:

|  |  |  |
| --- | --- | --- |
| **Character** | **ASCII code** | **URL** |
| Space | 20 | <http://www.thedesignshop.com/shops/tommy%20hilfiger.html> |
| ! | 21 | <http://www.thedesignshop.com/moredesigns%21.html> |
| + | 2B | <http://www.thedesignshop.com/shops/spencer%2B.html> |
| # | 23 | <http://www.thedesignshop.com/%23somequotes%23.html> |

Characters must be encoded if:

1. They have no corresponding character within the standard [ASCII character set](http://www.asciitable.com/).
2. The use of the character is unsafe because it may be misinterpreted, or even possibly modified by some systems. For example % is unsafe because it can be used for encoding other characters. Other unsafe characters include spaces, quotation marks, the # character, < and >, { and }, [ and ], and ~, among others.
3. The character is reserved for special use within the URL scheme. Some characters are reserved for a special meaning; their presence in a URL serve a specific purpose. Characters such as /, ?, :, @, and & are all reserved and must be encoded. For example & is reserved for use as a query string delimiter. : is also reserved to delimit host/port components and user/password.

So what characters can be used safely within a URL? Only alphanumeric and special characters $-\_.+!'()", and reserved characters when used for their reserved purposes can be used unencoded within a URL. As long as a character is not being used for its reserved purpose, it has to be encoded.

## Sample PHP Code to Fetch Query String

|  |
| --- |
| <?php  // Initialize URL to the variable  $url = '[https://www.geeksforgeeks.org?name=Tonny](https://www.geeksforgeeks.org/?name=Tonny)';    // Use parse\_url() function to parse the URL  // and return an associative array which  // contains its various components  $url\_components = parse\_url($url);    // Use parse\_str() function to parse the  // string passed via URL  parse\_str($url\_components['query'], $params);    // Display result  echo ' Hi '.$params['name'];    ?> |

# CSS3

## Selectors

A CSS selector selects the HTML element(s) you want to style.

CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

* Simple selectors (select elements based on name, id, class)
* Combinator selectors (select elements based on a specific relationship between them)
* Pseudo-class selectors (select elements based on a certain state)
* Pseudo-elements selectors (select and style a part of an element)
* Attribute selectors (select elements based on an attribute or attribute value)

## The CSS element Selector

The element selector selects HTML elements based on the element name.

**Example**

Here, all <p> elements on the page will be center-aligned, with a red text color:

p {  
  text-align: center;  
  color: red;  
}

## The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

**Example**

The CSS rule below will be applied to the HTML element with id="para1":

<!DOCTYPE html>

<html>

<head>

<style>

#para1 {

text-align: center;

color: red;

}

</style>

</head>

<body>

<p id="para1">Hello World!</p>

<p>This paragraph is not affected by the style.</p>

</body>

</html>

**Note:** An id name cannot start with a number!

## The CSS class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

**Example**

In this example all HTML elements with class="center" will be red and center-aligned:

<!DOCTYPE html>

<html>

<head>

<style>

.center {

text-align: center;

color: red;

}

</style>

</head>

<body>

<h1 class="center">Red and center-aligned heading</h1>

<p class="center">Red and center-aligned paragraph.</p>

</body>

</html>

You can also specify that only specific HTML elements should be affected by a class.

**Example**

In this example only <p> elements with class="center" will be red and center-aligned:

p.center {  
  text-align: center;

  color: red;  
}

HTML elements can also refer to more than one class.

**Example**

In this example the <p> element will be styled according to class="center" and to class="large":

<!DOCTYPE html>

<html>

<head>

<style>

p.center {

text-align: center;

color: red;

}

p.large {

font-size: 300%;

}

</style>

</head>

<body>

<h1 class="center">This heading will not be affected</h1>

<p class="center">This paragraph will be red and center-aligned.</p>

<p class="center large">This paragraph will be red, center-aligned, and in a large font-size.</p>

</body>

</html>

**Note:** A class name cannot start with a number!

## The CSS Universal Selector

The universal selector (\*) selects all HTML elements on the page.

**Example**

The CSS rule below will affect every HTML element on the page:

\* {  
  text-align: center;  
  color: blue;  
}

## The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

h1 {  
  text-align: center;  
  color: red;  
}  
  
h2 {  
  text-align: center;

  color: red;

}  
  
p {  
  text-align: center;  
  color: red;  
}

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

**Example**

In this example we have grouped the selectors from the code above:

h1, h2, p {

  text-align: center;

  color: red;  
}

## All CSS Simple Selectors

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Example description** |
| [#*id*](https://www.w3schools.com/cssref/sel_id.asp) | #firstname | Selects the element with id="firstname" |
| [.*class*](https://www.w3schools.com/cssref/sel_class.asp) | .intro | Selects all elements with class="intro" |
| [*element.class*](https://www.w3schools.com/cssref/sel_element_class.asp) | p.intro | Selects only <p> elements with class="intro" |
| [\*](https://www.w3schools.com/cssref/sel_all.asp) | \* | Selects all elements |
| [*element*](https://www.w3schools.com/cssref/sel_element.asp) | p | Selects all <p> elements |
| [*element,element,..*](https://www.w3schools.com/cssref/sel_element_comma.asp) | div, p | Selects all <div> elements and all <p> elements |

# The CSS Box Model

All HTML elements can be considered as boxes.

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

Explanation of the different parts:

* **Content** - The content of the box, where text and images appear
* **Padding** - Clears an area around the content. The padding is transparent
* **Border** - A border that goes around the padding and content
* **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

**Example**

Demonstration of the box model:

<!DOCTYPE html>

<html>

<head>

<style>

div {

background-color: lightgrey;

width: 300px;

border: 15px solid green;

padding: 50px;

margin: 20px;

}

</style>

</head>

<body>

<h2>Demonstrating the Box Model</h2>

<p>The CSS box model is essentially a box that wraps around every HTML element. It consists of: borders, padding, margins, and the actual content.</p>

<div>This text is the content of the box. We have added a 50px padding, 20px margin and a 15px green border. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</div>

</body>

</html>

## Width and Height of an Element

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

**Important:** When you set the width and height properties of an element with CSS, you just set the width and height of the **content area**. To calculate the full size of an element, you must also add padding, borders and margins.

**Example:** This <div> element will have a total width of 350px:

<!DOCTYPE html>

<html>

<head>

<style>

div {

width: 320px;

padding: 10px;

border: 5px solid gray;

margin: 0;

}

</style>

</head>

<body>

<h2>Calculate the total width:</h2>

<img src="klematis4\_big.jpg" width="350" height="263" alt="Klematis">

<div>The picture above is 350px wide. The total width of this element is also 350px.</div>

</body>

</html>

Here is the calculation:

320px (width)

+ 20px (left + right padding)

+ 10px (left + right border)

+ 0px (left + right margin)

**= 350px**

The total width of an element should be calculated like this:

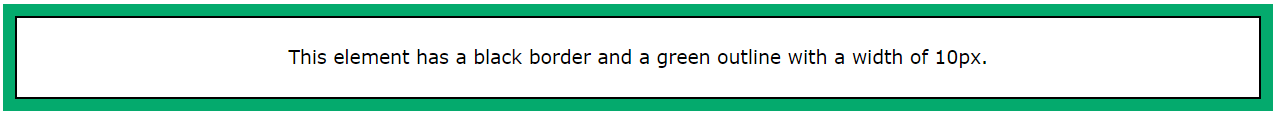
Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin.

# CSS Outline

An outline is a line drawn outside the element's border.



<!DOCTYPE html>

<html>

<head>

<style>

p {

border: 2px solid black;

outline: #4CAF50 solid 10px;

margin: auto;

padding: 20px;

text-align: center;

}

</style>

</head>

<body>

<h2>CSS Outline</h2>

<p>This element has a 2px black border and a green outline with a width of 10px.</p>

</body>

</html>

CSS has the following outline properties:

* outline-style
* outline-color
* outline-width
* outline-offset
* outline

**Note:** Outline differs from [borders](https://www.w3schools.com/css/css_border.asp)! Unlike border, the outline is drawn outside the element's border, and may overlap other content. Also, the outline is NOT a part of the element's dimensions; the element's total width and height is not affected by the width of the outline.

## CSS Outline Style

The outline-style property specifies the style of the outline, and can have one of the following values:

* dotted - Defines a dotted outline
* dashed - Defines a dashed outline
* solid - Defines a solid outline
* double - Defines a double outline
* groove - Defines a 3D grooved outline
* ridge - Defines a 3D ridged outline
* inset - Defines a 3D inset outline
* outset - Defines a 3D outset outline
* none - Defines no outline
* hidden - Defines a hidden outline

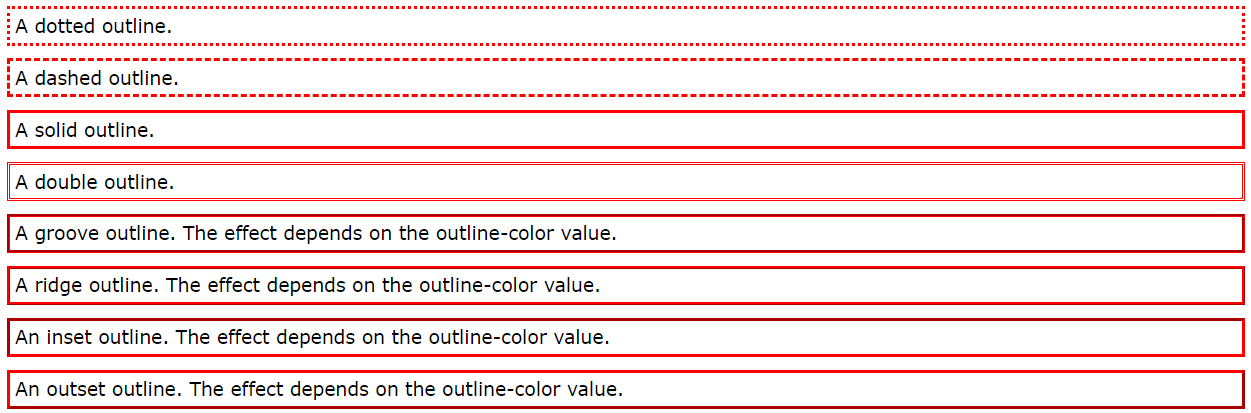
The following example shows the different outline-style values:

**Example**

Demonstration of the different outline styles:

p.dotted {outline-style: dotted;}  
p.dashed {outline-style: dashed;}  
p.solid {outline-style: solid;}  
p.double {outline-style: double;}  
p.groove {outline-style: groove;}  
p.ridge {outline-style: ridge;}  
p.inset {outline-style: inset;}  
p.outset {outline-style: outset;}

**Result:**



<!DOCTYPE html>

<html>

<head>

<style>

p {outline-color:red;}

p.dotted {outline-style: dotted;}

p.dashed {outline-style: dashed;}

p.solid {outline-style: solid;}

p.double {outline-style: double;}

p.groove {outline-style: groove;}

p.ridge {outline-style: ridge;}

p.inset {outline-style: inset;}

p.outset {outline-style: outset;}

</style>

</head>

<body>

<h2>The outline-style Property</h2>

<p class="dotted">A dotted outline</p>

<p class="dashed">A dashed outline</p>

<p class="solid">A solid outline</p>

<p class="double">A double outline</p>

<p class="groove">A groove outline. The effect depends on the outline-color value.</p>

<p class="ridge">A ridge outline. The effect depends on the outline-color value.</p>

<p class="inset">An inset outline. The effect depends on the outline-color value.</p>

<p class="outset">An outset outline. The effect depends on the outline-color value.</p>

</body>

</html>

# Flask

## What is Web Framework?

Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

## WSGI

Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

## Werkzeug

It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

## Jinja2

Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.

## What is Flask?

Flask is a web application framework written in Python. It is developed by **Armin Ronacher**, who leads an international group of Python enthusiasts named Pocco. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

**Flask is a web development framework.** It is a framework with a built-in development server and a debugger.

Flask framework in itself is different from the other archetypes as it allows web developers to be flexible and to comfortably accommodate the frequently released changes in the software development community.

Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the extensions to add such functionality to the application.

## Virtual Environment Setup (optional)

<https://www.geeksforgeeks.org/creating-python-virtual-environment-windows-linux/>

A Virtual Environment is a python environment, that is an isolated working copy of Python which allows you to work on a specific project without affecting other projects  
So basically it is a tool that enables multiple side-by-side installations of Python, one for each project.

### **Creating virtual environment in Linux**

If pip is not in your system

$ sudo apt-get install python-pip

Then install virtualenv

$ pip install virtualenv

Now check your installation

$ virtualenv --version

Create a virtual environment now,

$ virtualenv virtualenv\_name

After this command, a folder named **virtualenv\_name** will be created. You can name anything to it. If you want to create a virtualenv for specific python version, type

$ virtualenv -p /usr/bin/python3 virtualenv\_name

or

$ virtualenv -p /usr/bin/python2.7 virtualenv\_name

Now at last we just need to activate it, using command

$ source virtualenv\_name/bin/activate

Now you are in a Python virtual environment

You can deactivate using

$ deactivate

### **Creating Python virtualenv in Windows**

If python is installed in your system, then pip comes in handy.

So simple steps are:

1) Install virtualenv using

> pip install virtualenv

2)Now in which ever directory you are, this line below will create a virtualenv there

> virtualenv myenv

And here also you can name it anything.

3) Now if you are same directory then type,

> myenv\Scripts\activate

You can explicitly specify your path too.

Similarly like Linux you can deactivate it like

$ deactivate

## Install Flask

pip install Flask

## First Flask Application

# Hello.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

return 'Hello World’

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

Importing flask module in the project is mandatory. An object of Flask class is our **WSGI** application.

Flask constructor takes the name of **current module (\_\_name\_\_)** as argument.

The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function.

app.route(rule, options)

* The **rule** parameter represents URL binding with the function.
* The **options** is a list of parameters to be forwarded to the underlying Rule object.

In the above example, **‘/’** URL is bound with **hello\_world()** function. Hence, when the home page of web server is opened in browser, the output of this function will be rendered.

Finally the **run()** method of Flask class runs the application on the local development server.

app.run(host, port, debug, options)

All parameters are optional

|  |  |
| --- | --- |
| **Sr.No.** | **Parameters & Description** |
| 1 | host  Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to ‘0.0.0.0’ to have server available externally |
| 2 | port  Defaults to 5000 |
| 3 | debug  Defaults to false. If set to true, provides a debug information |
| 4 | options  To be forwarded to underlying Werkzeug server. |

The above given **Python** script is executed from Python shell.

Python Hello.py

A message in Python shell informs you that

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Open the above URL **(localhost:5000)** in the browser. **‘Hello World’** message will be displayed on it.

### Debug mode

A **Flask** application is started by calling the **run()** method. However, while the application is under development, it should be restarted manually for each change in the code. To avoid this inconvenience, enable **debug support**. The server will then reload itself if the code changes. It will also provide a useful debugger to track the errors if any, in the application.

The **Debug** mode is enabled by setting the **debug** property of the **application** object to **True** before running or passing the debug parameter to the **run()** method.

app.debug = True

app.run()

OR

app.run(debug = True)

## Flask Routing

Modern web frameworks use the routing technique to help a user remember application URLs. It is useful to access the desired page directly without having to navigate from the home page.

The **route()** decorator in Flask is used to bind URL to a function. For example:

# HelloRoute.py

@app.route(‘/hello’)

def hello\_world():

return ‘hello world’

Here, URL **‘/hello’** rule is bound to the **hello\_world()** function. As a result, if a user visits **http://localhost:5000/hello** URL, the output of the **hello\_world()** function will be rendered in the browser.

The **add\_url\_rule()** function of an application object is also available to bind a URL with a function as in the above example, **route()** is used.

A decorator’s purpose is also served by the following representation:

def hello\_world():

return ‘hello world’

app.add\_url\_rule(‘/’, ‘hello’, hello\_world)

## Flask Variables

It is possible to build a URL dynamically, by adding variable parts to the rule parameter. This variable part is marked as **<variable-name>**. It is passed as a keyword argument to the function with which the rule is associated.

In the following example, the rule parameter of **route()** decorator contains **<name>** variable part attached to URL **‘/variables’**. Hence, if the **http://localhost:5000/variables/AjaySingala** is entered as a **URL** in the browser, **‘AjaySingala** will be supplied to **variables()** function as argument.

# variables.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/variables/<name>')

def variables(name):

return 'Hello %s!' % name

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

Save the above script as **variables.py** and run it from Python shell. Next, open the browser and enter URL **http://localhost:5000/hello/AjaySingala.**

The following output will be displayed in the browser.

Hello AjaySingala!

In addition to the default string variable part, rules can be constructed using the following converters:

|  |  |
| --- | --- |
| **Sr.No.** | **Converters & Description** |
| 1 | int – accepts integer |
| 2 | float – For floating point value |
| 3 | path – accepts slashes used as directory separator character |

In the following code, all these constructors are used.

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/blog/<int:postID>')

def show\_blog(postID):

return 'Blog Number %d' % postID

@app.route('/rev/<float:revNo>')

def revision(revNo):

return 'Revision Number %f' % revNo

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

Run the above code from Python Shell. Visit the URL **http://localhost:5000/blog/11** in the browser.

The given number is used as argument to the **show\_blog()** function. The browser displays the following output −

Blog Number 11

Enter this URL in the browser − **http://localhost:5000/rev/1.1**

The **revision()** function takes up the floating point number as argument. The following result appears in the browser window −

Revision Number 1.100000

The URL rules of Flask are based on **Werkzeug’s** routing module. This ensures that the URLs formed are unique and based on precedents laid down by Apache.

Consider the rules defined in the following script:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/flask')

def hello\_flask():

return 'Hello Flask'

@app.route('/python/')

def hello\_python():

return 'Hello Python'

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

Both the rules appear similar but in the second rule, trailing slash **(/)** is used. As a result, it becomes a canonical URL. Hence, using **/python** or **/python/** returns the same output. However, in case of the first rule, **/flask/** URL results in **404 Not Found** page.

## Flask URL Binding

The **url\_for()** function is very useful for dynamically building a URL for a specific function. The function accepts the name of a function as first argument, and one or more keyword arguments, each corresponding to the variable part of URL.

The following script demonstrates use of **url\_for()** function.

# urlbinding.py

from flask import Flask, redirect, url\_for

app = Flask(\_\_name\_\_)

@app.route('/admin')

def hello\_admin():

return 'Hello Admin'

@app.route('/guest/<guest>')

def hello\_guest(guest):

return 'Hello %s as Guest' % guest

@app.route('/user/<name>')

def hello\_user(name):

if name =='admin':

return redirect(url\_for('hello\_admin'))

else:

return redirect(url\_for('hello\_guest',guest = name))

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

The above script has a function **user(name)** which accepts a value to its argument from the URL.

The **User()** function checks if an argument received matches **‘admin’** or not. If it matches, the application is redirected to the **hello\_admin()** function using **url\_for()**, otherwise to the **hello\_guest()** function passing the received argument as guest parameter to it.

Save the above code and run from Python shell.

Open the browser and enter URL as − **http://localhost:5000/user/admin**

The application response in browser is −

Hello Admin

Enter the following URL in the browser − **http://localhost:5000/user/mvl**

The application response now changes to −

Hello mvl as Guest

## Flask HTTP Methods

Http protocol is the foundation of data communication in world wide web. Different methods of data retrieval from specified URL are defined in this protocol.

The following table summarizes different http methods:

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | GET  Sends data in unencrypted form to the server. Most common method. |
| 2 | HEAD  Same as GET, but without response body |
| 3 | POST  Used to send HTML form data to server. Data received by POST method is not cached by server. |
| 4 | PUT  Replaces all current representations of the target resource with the uploaded content. |
| 5 | DELETE  Removes all current representations of the target resource given by a URL |

By default, the Flask route responds to the **GET** requests. However, this preference can be altered by providing methods argument to **route()** decorator.

In order to demonstrate the use of **POST** method in URL routing, first let us create an HTML form and use the **POST** method to send form data to a URL.

Save the following script as **login.html**

<html>

<body>

<form action = "http://localhost:5000/login" method = "post">

<p>Enter Name:</p>

<p><input type = "text" name = "nm" /></p>

<p><input type = "submit" value = "submit" /></p>

</form>

</body>

</html>

Now enter the following script in login.py:

# login.py

from flask import Flask, redirect, url\_for, request

app = Flask(\_\_name\_\_)

@app.route('/success/<name>')

def success(name):

return 'welcome %s' % name

@app.route('/login',methods = ['POST', 'GET'])

def login():

if request.method == 'POST':

user = request.form['nm']

return redirect(url\_for('success',name = user))

else:

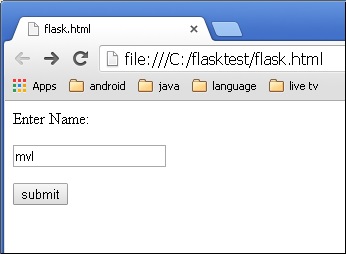
user = request.args.get('nm')

return redirect(url\_for('success',name = user))

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

After the development server starts running, open **login.html** in the browser (*directly from explorer*), enter name in the text field and click **Submit**.

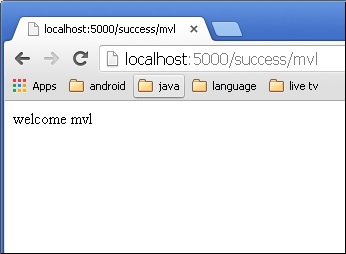


Form data is POSTed to the URL in action clause of form tag.

**http://localhost/login** is mapped to the **login()** function. Since the server has received data by **POST** method, value of ‘nm’ parameter obtained from the form data is obtained by:

user = request.form['nm']

It is passed to **‘/success’** URL as variable part. The browser displays a **welcome** message in the window.



Change the method parameter to **‘GET’** in **login.html** and open it again in the browser. The data received on server is by the **GET** method. The value of ‘nm’ parameter is now obtained by:

User = request.args.get(‘nm’)

Here, **args** is dictionary object containing a list of pairs of form parameter and its corresponding value. The value corresponding to ‘nm’ parameter is passed on to ‘/success’ URL as before.

## Flask Templates

It is possible to return the output of a function bound to a certain URL in the form of HTML. For instance, in the following script, **hello()** function will render **‘Hello World’** with **<h1>** tag attached to it.

# template\_inline.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return '<html><body><h1>Hello World</h1></body></html>'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

However, generating HTML content from Python code is cumbersome, especially when variable data and Python language elements like conditionals or loops need to be put. This would require frequent escaping from HTML.

This is where one can take advantage of **Jinja2** template engine, on which Flask is based. Instead of returning hardcode HTML from the function, a HTML file can be rendered by the render\_template() function.

# template\_simple.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template(‘hello.html’)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

Flask will try to find the HTML file in the templates folder, in the same folder in which this script is present.

* Application folder
  + template\_simple.py
  + templates
    - template\_simple.html

The term **‘web templating system’** refers to designing an HTML script in which the variable data can be inserted dynamically. A web template system comprises of a template engine, some kind of data source and a template processor.

Flask uses **jinja2** template engine. A web template contains HTML syntax interspersed placeholders for variables and expressions (in these case Python expressions) which are replaced values when the template is rendered.

The following code is saved as **template\_simple.html** in the templates folder.

<!doctype html>

<html>

<body>

<h1>Hello {{ name }}!</h1>

</body>

</html>

Next, run the following script **template\_simple.py**.

# template\_simple.py

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/hello/<user>')

def hello\_name(user):

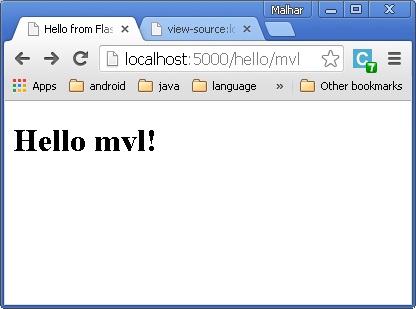
return render\_template('hello.html', name = user)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

As the development server starts running, open the browser and enter URL as − **http://localhost:5000/hello/mvl**

The **variable** part of URL is inserted at **{{ name }}** place holder.



The **jinja2** template engine uses the following delimiters for escaping from HTML.

* {% ... %} for Statements
* {{ ... }} for Expressions to print to the template output
* {# ... #} for Comments not included in the template output
* # ... ## for Line Statements

In the following example, use of conditional statement in the template is demonstrated. The URL rule to the **score()** function accepts the integer parameter. It is passed to the **template\_score.html** template. Inside it, the value of number received (marks) is compared (greater or less than 50) and accordingly HTML is conditionally rendered.

The Python Script is as follows:

# template\_score.py

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/score/<int:score>')

def hello\_name(score):

return render\_template(template\_score.html', marks = score)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

HTML template script of **template\_score.html** is as follows −

<!doctype html>

<html>

<body>

{% if marks>50 %}

<h1> Your result is pass!</h1>

{% else %}

<h1>Your result is fail</h1>

{% endif %}

</body>

</html>

Note that the conditional statements **if-else** and **endif** are enclosed in delimiter **{%..%}**.

Run the Python script and visit URL **http://localhost/score/60** and then **http://localhost/score/30** to see the output of HTML changing conditionally.

The Python loop constructs can also be employed inside the template. In the following script, the **result()** function sends a dictionary object to template **template\_result.html** when URL **http://localhost:5000/template\_result** is opened in the browser.

The Template part of **result.html** employs a **for loop** to render key and value pairs of dictionary object **result{}** as cells of an HTML table.

Run the following code from **template\_result.py**.

# template\_result.py

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/result')

def result():

dict = {'phy':50,'che':60,'maths':70}

return render\_template('result.html', result = dict)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

Save the following HTML script as **template\_result.html** in the templates folder.

<!doctype html>

<html>

<body>

<table border = 1>

{% for key, value in result.items() %}

<tr>

<th> {{ key }} </th>

<td> {{ value }} </td>

</tr>

{% endfor %}

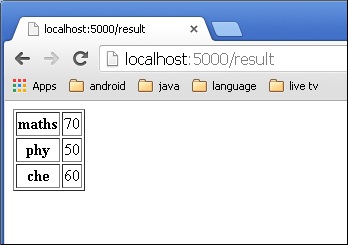
</table>

</body>

</html>

Here, again the Python statements corresponding to the **For** loop are enclosed in {%..%} whereas, the expressions **key and value** are put inside **{{ }}**.

After the development starts running, open **http://localhost:5000/result** in the browser to get the following output.



## Flask Static Files

A web application often requires a static file such as a **javascript** file or a **CSS** file supporting the display of a web page. Usually, the web server is configured to serve them for you, but during the development, these files are served from *static* folder in your package or next to your module and it will be available at ***/static*** on the application.

A special endpoint ‘static’ is used to generate URL for static files.

In the following example, a **javascript** function defined in **hello.js** is called on **OnClick** event of HTML button in **static\_index.html**, which is rendered on **‘/’** URL of the Flask application.

# static\_index.py

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

return render\_template("static\_index.html")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

The HTML script of **static\_index.html** is given below.

<html>

<head>

<script type = "text/javascript"

src = "{{ url\_for('static', filename = 'hello.js') }}" ></script>

</head>

<body>

<input type = "button" onclick = "sayHello()" value = "Say Hello" />

</body>

</html>

**<root>/static/hello.js** contains **sayHello()** function.

function sayHello() {

alert("Hello World")

}

## Flask Request Object

The data from a client’s web page is sent to the server as a global request object. In order to process the request data, it should be imported from the Flask module.

Important attributes of request object are listed below:

* **Form** − It is a dictionary object containing key and value pairs of form parameters and their values.
* **args** − parsed contents of query string which is part of URL after question mark (?).
* **Cookies** − dictionary object holding Cookie names and values.
* **files** − data pertaining to uploaded file.
* **method** − current request method.

## Flask Sending Form Data to Template

We have already seen that the http method can be specified in URL rule. The **Form** data received by the triggered function can collect it in the form of a dictionary object and forward it to a template to render it on a corresponding web page.

In the following example, **‘/’** URL renders a web page (student.html) which has a form. The data filled in it is posted to the **‘/result’** URL which triggers the **result()** function.

The **results()** function collects form data present in **request.form** in a dictionary object and sends it for rendering to **formdata\_result.html**.

The template dynamically renders an HTML table of **form** data.

Given below is the Python code of application −

# formdata.py

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route('/')

def student():

return render\_template('student.html')

@app.route('/result',methods = ['POST', 'GET'])

def result():

if request.method == 'POST':

result = request.form

return render\_template("formdata\_result.html",result = result)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

Given below is the HTML script of **student.html**.

<html>

<body>

<form action = "http://localhost:5000/result" method = "POST">

<p>Name <input type = "text" name = "Name" /></p>

<p>Physics <input type = "text" name = "Physics" /></p>

<p>Chemistry <input type = "text" name = "chemistry" /></p>

<p>Maths <input type ="text" name = "Mathematics" /></p>

<p><input type = "submit" value = "submit" /></p>

</form>

</body>

</html>

Code of template **(formdata\_result.html)** is given below −

<!doctype html>

<html>

<body>

<table border = 1>

{% for key, value in result.items() %}

<tr>

<th> {{ key }} </th>

<td> {{ value }} </td>

</tr>

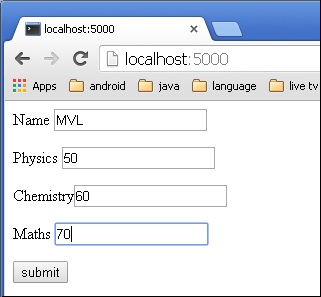
{% endfor %}

</table>

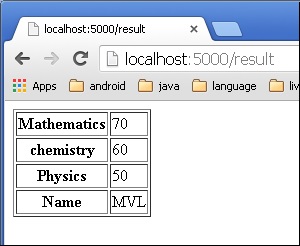
</body>

</html>

Run the Python script and enter the URL **http://localhost:5000/** in the browser.



When the **Submit** button is clicked, form data is rendered on **formdata\_result.html** in the form of HTML table.



## Flask Cookies

A cookie is stored on a client’s computer in the form of a text file. Its purpose is to remember and track data pertaining to a client’s usage for better visitor experience and site statistics.

A **Request object** contains a cookie’s attribute. It is a dictionary object of all the cookie variables and their corresponding values, a client has transmitted. In addition to it, a cookie also stores its expiry time, path and domain name of the site.

In Flask, cookies are set on response object. Use **make\_response()** function to get response object from return value of a view function. After that, use the **set\_cookie()** function of response object to store a cookie.

Reading back a cookie is easy. The **get()** method of **request.cookies** attribute is used to read a cookie.

In the following Flask application, a simple form opens up as you visit **‘/’** URL.

@app.route('/')

def index():

return render\_template('index.html')

This HTML page contains one text input (**index.html**):

<html>

<body>

<form action = "/setcookie" method = "POST">

<p><h3>Enter userID</h3></p>

<p><input type = 'text' name = 'nm'/></p>

<p><input type = 'submit' value = 'Login'/></p>

</form>

</body>

</html>

The Form is posted to **‘/setcookie’** URL. The associated view function sets a Cookie name **userID** and renders another page.

@app.route('/setcookie', methods = ['POST', 'GET'])

def setcookie():

if request.method == 'POST':

user = request.form['nm']

resp = make\_response(render\_template('readcookie.html'))

resp.set\_cookie('userID', user)

return resp

**‘readcookie.html’** contains a hyperlink to another view function **getcookie()**, which reads back and displays the cookie value in browser.

@app.route('/getcookie')

def getcookie():

name = request.cookies.get('userID')

return '<h1>welcome '+name+'</h1>'

**readcookie.html**:

<html>

   <body>

    <!-- <form action = "/getcookie" method = "GET">

        <p><input type = 'submit' value = 'Get Cookie'/></p>

     </form> -->

     <h2>Cookie 'userID' is set</h2>

     <a href="getcookie">Click here to read cookie</a>

   </body>

</html>

**Full Python code**:

# cookie.py

from flask import Flask, render\_template, request, make\_response

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

   return render\_template('cookie\_index.html')

@app.route('/setcookie', methods = ['POST', 'GET'])

def setcookie():

   if request.method == 'POST':

       user = request.form['nm']

   resp = make\_response(render\_template('readcookie.html'))

   resp.set\_cookie('userID', user)

   return resp

@app.route('/getcookie')

def getcookie():

   name = request.cookies.get('userID')

   return '<h1>welcome '+name+'</h1>'

if \_\_name\_\_ == '\_\_main\_\_':

   app.run(debug = True)

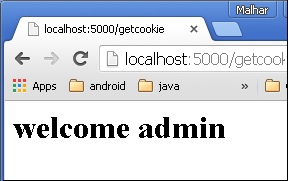
Run the application and visit **http://localhost:5000/**



The result of setting a cookie is displayed like this:



The output of read back cookie is shown below.



## Flask Sessions

Like Cookie, Session data is stored on client. Session is the time interval when a client logs into a server and logs out of it. The data, which is needed to be held across this session, is stored in the client browser.

A session with each client is assigned a **Session ID**. The Session data is stored on top of cookies and the server signs them cryptographically. For this encryption, a Flask application needs a defined **SECRET\_KEY**.

Session object is also a dictionary object containing key-value pairs of session variables and associated values.

For example, to set a **‘username’** session variable use the statement −

Session[‘username’] = ’admin’

To release a session variable use **pop()** method.

session.pop('username', None)

The following code is a simple demonstration of session works in Flask. URL **‘/’** simply prompts user to log in, as session variable **‘username’** is not set.

# session\_login.py

@app.route('/')

def index():

if 'username' in session:

username = session['username']

return 'Logged in as ' + username + '<br>' + \

"<b><a href = '/logout'>click here to log out</a></b>"

return "You are not logged in <br><a href = '/login'></b>" + \

"click here to log in</b></a>"

As user browses to ‘/login’ the login() view function, because it is called through GET method, opens up a login form.

A Form is posted back to **‘/login’** and now session variable is set. Application is redirected to **‘/’**. This time session variable **‘username’** is found.

@app.route('/login', methods = ['GET', 'POST'])

def login():

if request.method == 'POST':

session['username'] = request.form['username']

return redirect(url\_for('index'))

return '''

<form action = "" method = "post">

<p><input type=’text’ name=’username’/></p>

<p<input type=‘submit’ value=‘Login’/></p>

</form>

'''

The application also contains a **logout()** view function, which pops out **‘username’** session variable. Hence, **‘/’** URL again shows the opening page.

@app.route('/logout')

def logout():

# remove the username from the session if it is there

session.pop('username', None)

return redirect(url\_for('index'))

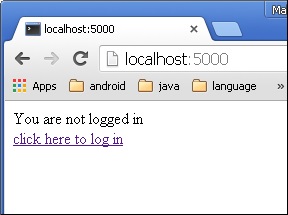
Run the application and visit the homepage. (Ensure to set **secret\_key** of the application)

from flask import Flask, session, redirect, url\_for, escape, request

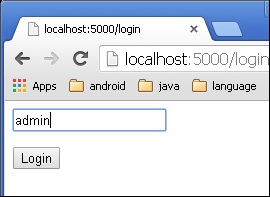
app = Flask(\_\_name\_\_)

app.secret\_key = 'any random string’

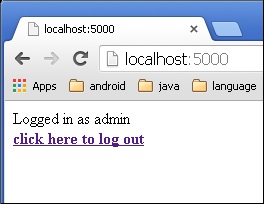
The output will be displayed as shown below. Click the link **“click here to log in”**.



The link will be directed to another screen. Type ‘admin’.



The screen will show you the message, **‘Logged in as admin’**.



## Flask Redirect & Errors

Flask class has a **redirect()** function. When called, it returns a response object and redirects the user to another target location with specified status code.

Prototype of **redirect()** function is as below −

Flask.redirect(location, statuscode, response)

In the above function −

* **location** parameter is the URL where response should be redirected.
* **statuscode** sent to browser’s header, defaults to 302.
* **response** parameter is used to instantiate response.

The following status codes are standardized −

* HTTP\_300\_MULTIPLE\_CHOICES
* HTTP\_301\_MOVED\_PERMANENTLY
* HTTP\_302\_FOUND
* HTTP\_303\_SEE\_OTHER
* HTTP\_304\_NOT\_MODIFIED
* HTTP\_305\_USE\_PROXY
* HTTP\_306\_RESERVED
* HTTP\_307\_TEMPORARY\_REDIRECT

The **default status** code is **302**, which is for **‘found’**.

In the following example, the **redirect()** function is used to display the login page again when a login attempt fails.

# redirect.py

from flask import Flask, redirect, url\_for, render\_template, request

# Initialize the Flask application

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('redirect\_login.html')

@app.route('/login',methods = ['POST', 'GET'])

def login():

if request.method == 'POST' and request.form['username'] == 'admin' :

return redirect(url\_for('success'))

else:

return redirect(url\_for('index'))

@app.route('/success')

def success():

return 'logged in successfully'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

Flask class has **abort()** function with an error code.

Flask.abort(code)

The **Code** parameter takes one of following values −

* **400** − for Bad Request
* **401** − for Unauthenticated
* **403** − for Forbidden
* **404** − for Not Found
* **406** − for Not Acceptable
* **415** − for Unsupported Media Type
* **429** − Too Many Requests

Let us make a slight change in the **login()** function in the above code. Instead of re-displaying the login page, if **‘Unauthourized’** page is to be displayed, replace it with call to **abort(401)**.

# abort.py

from flask import Flask, redirect, url\_for, render\_template, request, abort

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('log\_in.html')

@app.route('/login',methods = ['POST', 'GET'])

def login():

if request.method == 'POST':

if request.form['username'] == 'admin' :

return redirect(url\_for('success'))

else:

abort(401)

else:

return redirect(url\_for('index'))

@app.route('/success')

def success():

return 'logged in successfully'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)