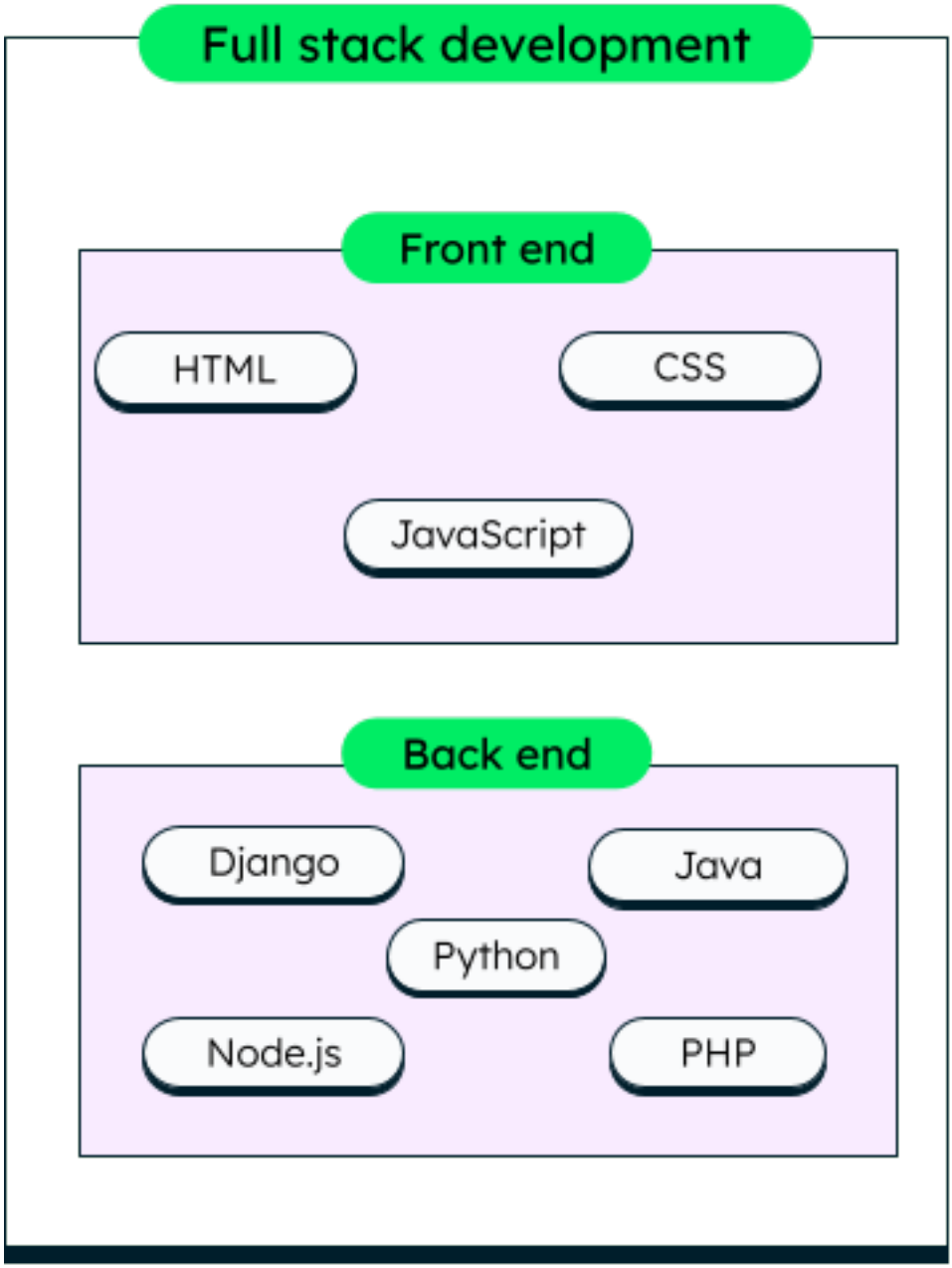
Full stack development refers to the end-to-end application software development, including the front end and back end. The front end consists of the user interface, and the back end takes care of the business logic and application workflows.

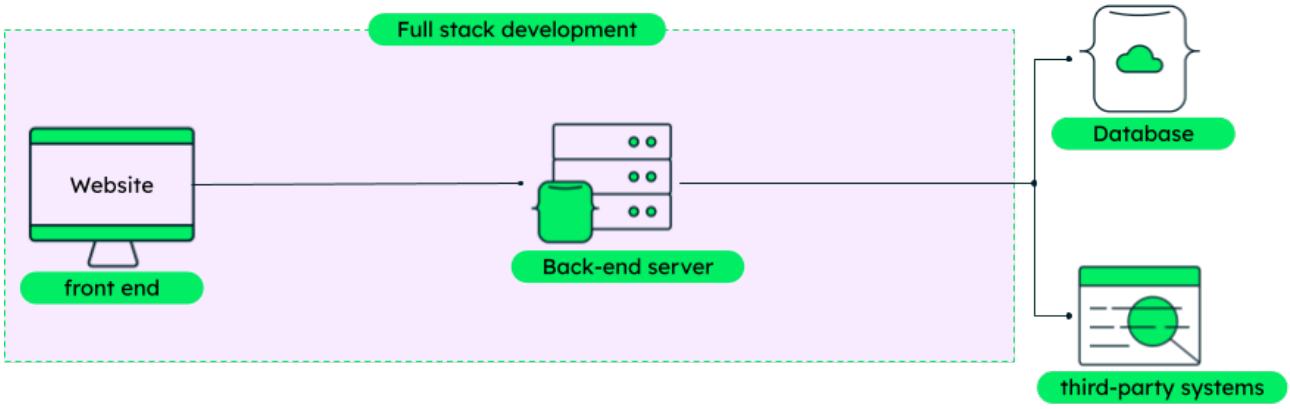


The main components of a full stack development are the front-end, back-end and database

Consider a retail website. Users can browse or purchase specific items, delete or add items in cart, change their profile, and do many other things. All these actions require a front-end user interface (UI), as well as some business logic, written in the back-end.

- The website UI can be built using various, *front-end technologies* like HTML, CSS, Javascript.
- The back end is written in *programming languages* like Java or Python. Further, a good web application would need scalability, event handling, and routing, which are usually handled by *libraries and frameworks* like React or Angular.
- The back end also consists of logic that can connect the application to other services and databases. For example, all the user and transaction data is stored in a database through specific *drivers* handled on the back end.

A full stack developer is one who can single-handedly implement both the front-end and back-end workflows, like placing the order or changing the user profile.



Demonstration of full stack development in an end-to-end workflow

What is a full stack developer and what do they do?

Full stack developers must have knowledge of an entire technology stack, i.e., the set of technologies that are used to build an end-to-end application quickly and efficiently. For example, if they want to build an application using the MERN stack, they should know how to work with MongoDB, Express, React and Node.

Full stack developers should be able to judge whether the selected technologies are the right choice for their project during the early phases. Some responsibilities of a full stack developer are to:

- Help in choosing the right technologies for the project development and testing both on the front end and the back end.
- Write clean code across the stack by following the best practices of the tools used.
- Be up to date with the latest technologies and tools to make the best technology usage decisions.

What languages do full stack developers use?

Full stack developers are free to use any set of languages that are compatible with each other and the overall application framework. JavaScript is a popular language often used by full-stack developers as it's one of the very few languages that can be used both on the front end and back end. Companies will most likely hire a full stack developer for smaller or medium-size projects. Some popular languages are:

- Front end: HTML, CSS, JavaScript.
- Back end: Python, Java, R, Ruby, Node.js, PHP.

It's also a popular and convenient practice to use full technology stacks like MEAN stack, MERN stack, Ruby on Rails, and LAMP for faster and more efficient development, and an easier learning curve.

Front end vs back end vs full stack

Applications that require higher scalability and more complex workflows require broader skill sets and collaboration across teams. For example, the front end may be handled by the UI team, and the back end by another team. In some organizations, individuals will be required to work on both the front-end and back-end implementation of a feature. This is where full stack developers would come into play.

Front-end developers

These developers handle the UI of a web application (or website)—for example, visual effects, frames, navigation, and forms. They focus mainly on user experience and use HTML, CSS, and JavaScript as programming languages.

Back-end developers

They deal with the business logic, security, performance, scalability, and handling request-response of the application. They create or use frameworks to design the core application workflows and use technologies like JavaScript, Python, Java, and .NET.

Full stack developers

They are responsible for coding end-to-end workflows by using both front-end and back-end technologies. MERN stack and MEAN stack are examples of JavaScript-based technology stacks that full stack developers can use to build end-to-end applications.

Full stack development advantages

There are many advantages of hiring full stack developers for web application development:

- Complete ownership and understanding of the project
- Saves both project time and cost, and enhances productivity
- Faster bug fixing due to knowledge of complete system
- Easy knowledge transfer to other team members
- Better division of work amongst team members

The **Model-View-Controller (MVC)** framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. Hence the abbreviation MVC. Each architecture component is built to handle specific development aspect of an application. MVC separates the business logic and presentation layer from each other. It was traditionally used for desktop graphical user interfaces (GUIs). Nowadays, MVC architecture in web technology has become popular for designing web applications as well as mobile apps.

Features of MVC

- Easy and frictionless testability. Highly testable, extensible and pluggable framework
- To design a web application architecture using the MVC pattern, it offers full control over your HTML as well as your URLs

- Leverage existing features provided by ASP.NET, JSP, Django, etc.
- Clear separation of logic: Model, View, Controller. Separation of application tasks viz. business logic, UI logic, and input logic
- URL Routing for SEO Friendly URLs. Powerful URL-mapping for comprehensible and searchable URLs
- Supports for Test Driven Development (TDD)

Three important MVC components are:

- Model: It includes all the data and its related logic
- View: Present data to the user or handles user interaction
- Controller: An interface between Model and View components

View

A View is that part of the application that represents the presentation of data.
Views are created by the data collected from the model data. A view requests the model to give information so that it presents the output presentation to the user.
The view also represents the data from charts, diagrams, and tables. For example, any customer view will include all the UI components like text boxes, drop downs, etc.

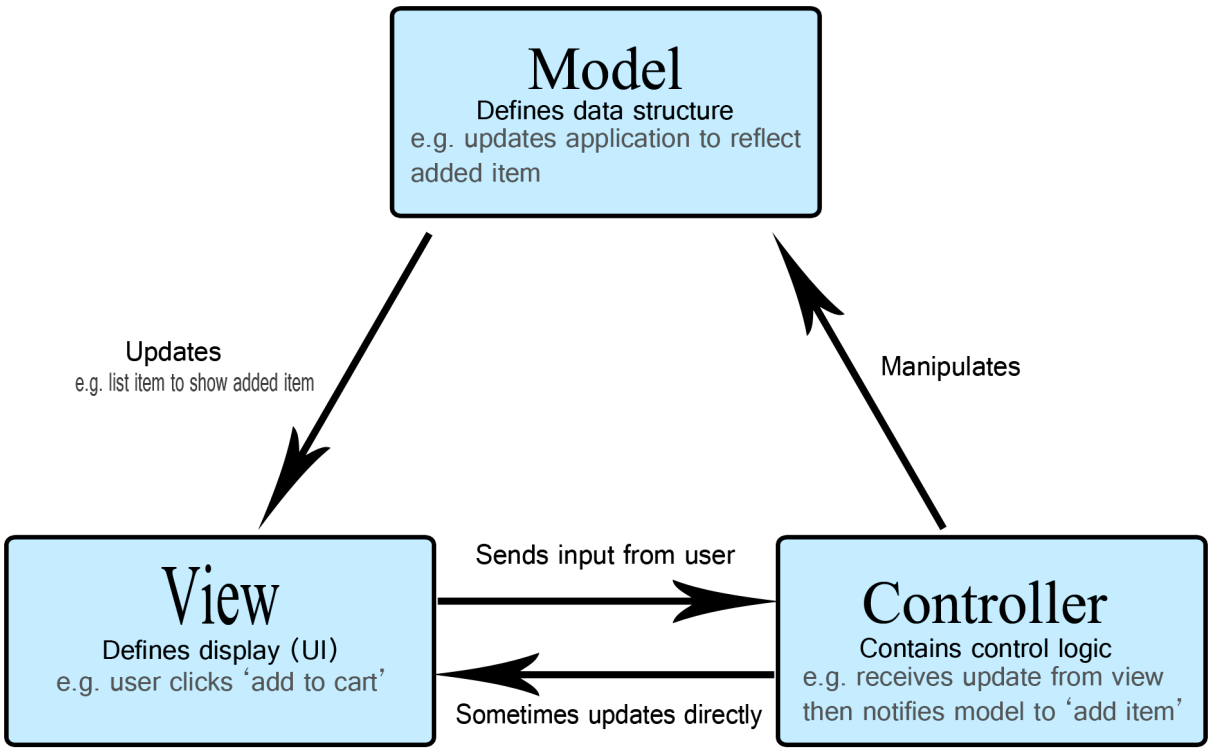
Controller

The Controller is that part of the application that handles the user interaction. The controller interprets the mouse and keyboard inputs from the user, informing model and the view to change as appropriate.
A Controller send's commands to the model to update its state(E.g., Saving a specific document). The controller also sends commands to its associated view to change the view's presentation (For example scrolling a particular document).

Model

The model component stores data and its related logic. It represents data that is being transferred between controller components or any other related business logic. For example, a Controller object will retrieve the customer info from the database. It manipulates data and sends back to the database or uses it to render the same data.
It responds to the request from the views and also responds to instructions from the controller to update itself. It is also the lowest level of the pattern which is responsible for maintaining data.

MVC Architecture



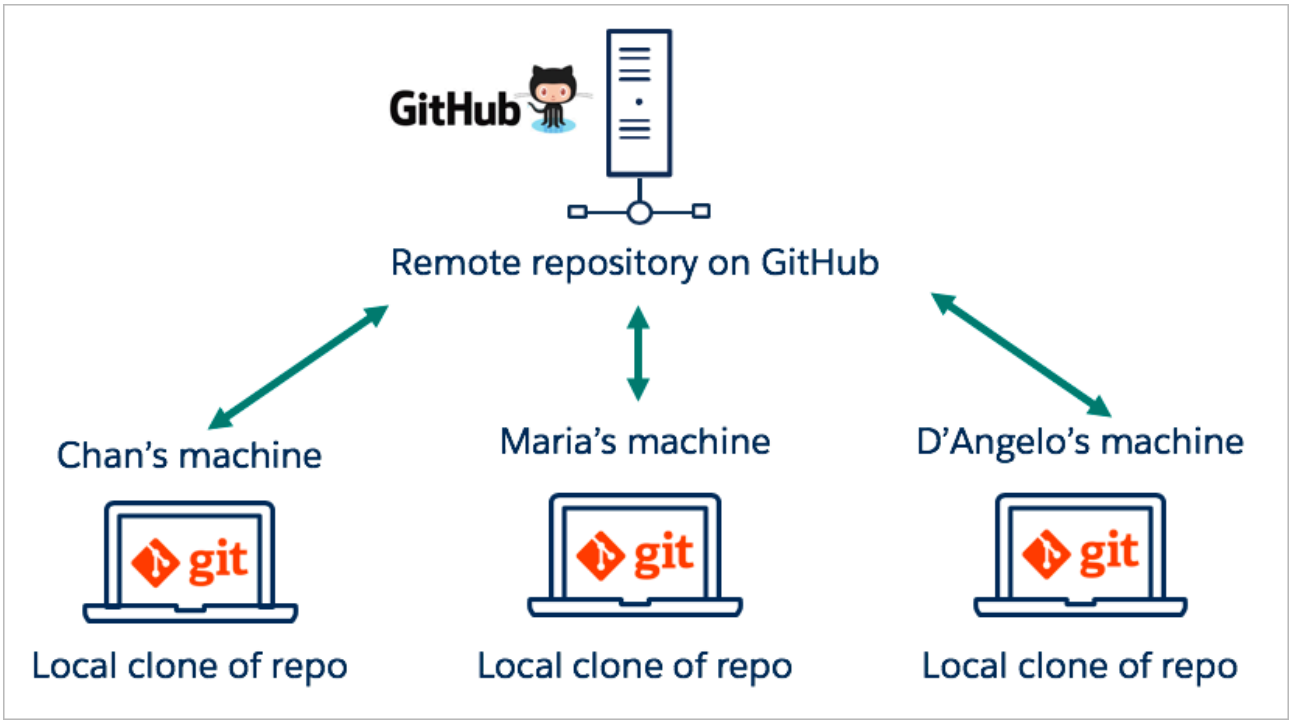
Domain Driven Design (DDD) Architecture:

Domain-Driven Design is a concept introduced by a programmer Eric Evans in 2004 in his book Domain-Driven Design: Tackling Complexity in Heart of Software.

What is Domain ?
The word Domain used in context of software development refers to business. In the process of application development, term domain logic or business logic is commonly used. Basically, business logic is area of knowledge around which application logic revolves. The business logic of an application is a set of rules and guidelines that explain how business object should interact with each other to process modeled data.

Repository:

A repository contains all of your project's files and each file's revision history. You can discuss and manage your project's work within the repository.



Install Required Application

1. Visual Code with Git & Html BoilersPlate as extension
2. GitHub Desktop
3. Create GitHub account and integrate with VS Code and GitHub Desktop

Full Stack Development

Project Structure

FrontEnd	BackEnd	Database
Languages	**Languages**	**Languages**
(HTML I CSS I Java/Type Script).	**(Java/Type Script)**	**(Java/Type Script)**
Frameworks		
(Vue I Angular I Webpack)	**(Express Js)**	**(MangoDB)**
Libraries		
(React I jQuery)	**(Express Js)**	**(Mongose)**

HTML

HTML Hyper Text Markup Language: is the standard markup language for creating Web pages.

HTML Element:

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

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

HTML Paragraphs

HTML paragraphs are defined with the `<p>` tag:

<> </> = Fragment

HTML Links (a=Anchor Tag)

HTML links are defined with the `<a>` tag:

```
Eg. <a href="https://www.softpro9.com">This is a
link</a>
```

HTML Images

HTML images are defined with the `<img>` tag.

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

```
Eg.<img src="softpro9.jpg" alt="softpro9.com" width="
104" height="142">
```

Image with Hyperlink:

```
<a herf="url"><img src="file path" alt="text"
height="150px" width="150px"
```

Images on Another Server/ Website

Some web sites point to an image on another server.

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

```
<img src=“https://softpro9.com/images/
image1.jpg" alt=“softpro9 image1">
```

Image Floating

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

```
<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

View HTML Source Code:

Right-click in an HTML page and select "View Page Source" (in Chrome) or "View Source" (in Edge), or similar in other browsers. This will open a window containing the HTML source code of the page.

HTML Block and 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, and the browsers automatically add some space (a margin) before and after the element.

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

Two commonly used block elements are: `<p>` and `<div>`.

The `<p>` element defines a paragraph in an HTML document.

The `<div>` element defines a division or a section in an HTML document.

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

The element is a block-level element.

Example:

```
<p>Hello World</p>
<div>Hello World</div>
```

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 element inside a paragraph.

Example: `<span>Hello World</span>`

The 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 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>
```

async: Executes the code while downloading and do not block DOM construction during download process

defer: Executes the code once its downloaded & browser has finished DOM construction & rendering process.

Exercise : Create a html page with all the above discussed elements (header tags, style, image with hyperlink, Video and geo map and link the page to index.html

Session 2:

HTML Styles

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

The HTML `style` attribute has the following syntax:

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

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

- Use the `style` attribute for styling HTML elements
- Use `background-color` for background color
- Use `color` for text colors
- Use `font-family` for text fonts
- Use `font-size` for text sizes
- Use `text-align` for text alignment

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

1. <body style="background-color:powderblue;">

```
<h1>Welcome to Html Style element</h1>
```

```
<p>Lets know how to add style to html.</p>

</body>
```

2. Set background color for two different elements:

```
<body>

<h1 style="background-color:powderblue;">Welcome to
Html Style element</h1>
<p style="background-color:tomato;">Lets know how to
add style to html.</p>

</body>
```

Text Color

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

```
<h1 style="color:blue;">Welcome to Html Style
element</h1>
<p style="color:red;">Lets know how to add style to
html.</p>
```

Fonts

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

```
<h1 style="font-family:verdana;">Welcome to Html
Style element</h1>
<p style="font-family:courier;">Lets know how to add
style to html.</p>
```

Text Size

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

```
<h1 style="font-size:300%;">Welcome to Html Style element</h1>
<p style="font-size:160%;">Lets know how to add style to html.</p>
```

Text Alignment

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

```
<h1 style="text-align:center;">Centered Heading</h1>
<p style="text-align:center;">Centered paragraph.</p>
```

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 Elements

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

```
<b>This text is bold</b>
```

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

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

HTML <i> and Elements

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

```
<i>This text is italic</I>
```

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

```
<em>This text is emphasized</em>
```

HTML <small> Element

The HTML `<small>` element defines smaller text:

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

HTML <mark> Element

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

```
<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:

```
<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:

```
<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 H_2O:

```
<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]:

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

HTML Colors

HTML colors are specified with predefined color names, or with RGB, HEX, HSL, RGBA, or HSLA values.
In HTML, a color can be specified by using a color name:
Tomato
Orange
DodgerBlue
MediumSeaGreen
Gray
SlateBlue
Violet
LightGray

Background Color

set the background color for HTML elements:

```
<h1 style="background-color:DodgerBlue;">Hello
World</h1>
<p style="background-color:Tomato;">Welcome</p>
```

Text Color

set the color of text:

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Welcome</p>
```

```
<p style=“color:MediumSeaGreen;">Good Morning</p>
```

Border Color

set the color of borders:

```
<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>
```

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.

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!

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, we will use inline and internal styles.

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:

```
<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:

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

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:

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

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:

```
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```

Sessin 3

CSS Colors, Fonts and Sizes

Some commonly used CSS properties.

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.

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.

```
p {
  border: 2px solid powderblue;
}
```

CSS Padding

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

```
p {
  border: 2px solid powderblue;
  padding: 30px;
}
```

CSS Margin

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

```
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.

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

```
<link rel="stylesheet" href="https://softpro9.com/
html/styles.css">
```

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

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

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

HTML links are hyperlinks.

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

- Use the `<a>` element to define a link
- Use the `href` attribute to define the link address
- Use the `target` attribute to define where to open the linked document
- Use the `<img>` element (inside `<a>`) to use an image as a link
- Use the `mailto:` scheme inside the `href` attribute to create a link that opens the user's email program

HTML Links - Syntax

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

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

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

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

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

```
<a href="https://softpro9.com/">Welcome to SoftPro9.com!</a>
```

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

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

```
<a href="https://
softpro9.com/" target="_blank">Welcome to
SoftPro9.com!</a>
```

HTML Links - Use an Image as a Link

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

```
<a href="default.asp">
<img src="file
path" alt="text" 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):

```
<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:

```
<button onclick="document.location='default.js'">HTML Button</button>
```

Button as a Link to Scroll to Top

Add any element with id tag just below body open tag as shown

 Top Area

This code need to be added into bottom of body but just above body closure tag

 Scroll To Top

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.

```
<a href="https://softpro9.com/html/" title="Welcome to SoftPro9 HTML Learning section">Learn HTML</a>
```

HTML Image Maps

The idea behind an image map is that you should be able to perform different actions depending on where in the image you click.

To create an image map you need an image, and some HTML code that describes the clickable areas.

The Image

The image is inserted using the `<img>` tag. The only difference from other images is that you must add a `usemap` attribute:

```
<img src="workplace.jpg" alt="Workplace" usemap="#workmap">
```

The `usemap` value starts with a hash tag `#` followed by the name of the image map, and is used to create a relationship between the image and the image map.

Create Image Map

Then, add a `<map>` element.

The `<map>` element is used to create an image map, and is linked to the image by using the required `name` attribute:

```
<map name="workmap">
```

The `name` attribute must have the same value as the `<img>`'s `usemap` attribute .

https://www.image-map.net/ use this link to generate image coordinates

The Areas

Then, add the clickable areas.

A clickable area is defined using an `<area>` element.

Shape

You must define the shape of the clickable area, and you can choose one of these values:

- `rect` - defines a rectangular region
- `circle` - defines a circular region
- `poly` - defines a polygonal region
- `default` - defines the entire region

You must also define some coordinates to be able to place the clickable area onto the image.

Shape="rect"

The coordinates for `shape="rect"` come in pairs, one for the x-axis and one for the y-axis.

```
<area shape="rect" coords="34, 44, 270,
350" href="computer.htm">
```

Shape="poly"

The `shape="poly"` contains several coordinate points, which creates a shape formed with straight lines (a polygon).

This can be used to create any shape.

Like maybe a croissant shape!

The coordinates come in pairs, one for the x-axis and one for the y-axis:

Image Map and JavaScript

A clickable area can also trigger a JavaScript function.

Add a `click` event to the `<area>` element to execute a JavaScript function:

Here, we use the onclick attribute to execute a JavaScript function when the area is clicked:

```
<map name="workmap">
  <area shape="circle" coords="337,300,44" href="coff
ee.htm" onclick="myFunction()">
</map>

<script>
function myFunction() {
  alert("You clicked the coffee cup!");
}
</script>
```

Background Image on a HTML element

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

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

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

```
<style>
p {
  background-image: url('img.jpg');
}
</style>
```

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:

Add a background image for the entire page:

```
<style>
body {
  background-image: url('img.jpg');
}
</style>
```

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:

```
<style>
body {
  background-image: url('img.jpg');
}
</style>
```

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

```
<style>
body {
  background-image: url('example_img_girl.jpg');
  background-repeat: no-repeat;
}
</style>
```

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):

```
<style>
body {
  background-image: url('img_girl.jpg');
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: cover;
}
</style>
```

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.

```
<style>
```

```
body {
  background-image: url('img_girl.jpg');
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: 100% 100%;
}
</style>
```

HTML <picture> Element

The HTML `<picture>` element allows you to display different pictures for different devices or screen sizes.

The HTML <picture> Element

The HTML `<picture>` element gives web developers more flexibility in specifying image resources.

The `<picture>` element contains one or more `<source>` elements, each referring to different images through the `srcset` attribute. This way the browser can choose the image that best fits the current view and/or device.

Each `<source>` element has a `media` attribute that defines when the image is the most suitable.

Show different images for different screen sizes:

```
<picture>
  <source media="(min-width:
650px)" srcset="img1.jpg">
  <source media="(min-width:
465px)" srcset="img2.jpg">
  <img src="img3.jpg">
</picture>
```

Always specify an <img> element as the last child element of the <picture> element. The <img> element is used by

browsers that do not support the <picture> element, or if none of the <source> tags match.

When to use the Picture Element

There are two main purposes for the `<picture>` element:

1. Bandwidth

If you have a small screen or device, it is not necessary to load a large image file. The browser will use the first `<source>` element with matching attribute values, and ignore any of the following elements.

2. Format Support

Some browsers or devices may not support all image formats. By using the `<picture>` element, you can add images of all formats, and the browser will use the first format it recognizes, and ignore any of the following elements.

The browser will use the first image format it recognizes:

```
<picture>
  <source srcset="img_avatar.png">
  <source srcset="img_girl.jpg">
  <img src="img_beatles.gif" alt="Beatles" style="wid
th:auto;">
</picture>
```

HTML Image Tags

Tag	Description

<img>	Defines an image
<map>	Defines an image map
<area>	Defines a clickable area inside an image map
<picture>	Defines a container for multiple image resources

HTML Favicon

A favicon is a small image displayed next to the page title in the browser tab.

How To Add a Favicon in HTML

You can use any image you like as your favicon. You can also create your own favicon on sites like https://www.favicon.cc.

To add a favicon to your website, either save your favicon image to the root directory of your webserver, or create a folder in the root directory called images, and save your favicon image in this folder. A common name for a favicon image is "favicon.ico".

Next, add a `<link>` element to your "index.html" file, after the `<title>` element, like this:

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

```
  <link rel="icon" type="image/x-icon" href="/images/
favicon.ico">
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Now, save the "index.html" file and reload it in your browser. Your browser tab should now display your favicon image to the left of the page title.

HTML Tables

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

Define an HTML Table

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

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. A table cell can contain all sorts of HTML elements: text, images, lists, links, other tables, etc.

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

Table Rows

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

`tr` stands for table row.

```
<table>
  <tr>
    <td>Emil</td>
    <td>Tobias</td>
```

```
    <td>Linus</td>
  </tr>
  <tr>
    <td>16</td>
    <td>14</td>
    <td>10</td>
  </tr>
</table>
```

Can have as many rows as we like in a table; just make sure that the number of cells are the same in each row.

Table Headers

Sometimes you want your cells to be table header cells. In those cases use the `<th>` tag instead of the `<td>` tag:

`th` stands for table header.
Let the first row be table header cells:

```
<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 Lists

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

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:

```
<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:

```
<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:

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

HTML List Tags

Tag	Description
	Defines an unordered list

	Defines an ordered list
	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 Unordered Lists

The HTML `<ul>` tag defines an unordered (bulleted) list.

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:

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

Unordered HTML List - Choose List Item Marker

The CSS `list-style-type` property is used to define the style of the list item marker. It can have one of the following values:

Value	Description
disc	Sets the list item marker to a bullet (default)

circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

Example - Disc

```
<ul style="list-style-type:disc;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Circle

```
<ul style="list-style-type:circle;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Square

```
<ul style="list-style-type:square;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - None

```
<ul style="list-style-type:none;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Nested HTML Lists

Lists can be nested (list inside list):

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
```

Note: A list item (`<li>`) can contain a new list, and other HTML elements, like images and links, etc.

HTML Ordered Lists

The HTML `<ol>` tag defines an ordered list. An ordered list can be numerical or alphabetical.

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:

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

Numbers:

```
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Letters:

```
<ol type="A">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Letters:

```
<ol type="a">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Roman Numbers:

```
<ol type="I">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Roman Numbers:

```
<ol type="i">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Control List Counting

By default, an ordered list will start counting from 1. If you want to start counting from a specified number, you can use the `start` attribute:

```
<ol start="50">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Nested HTML Lists

Lists can be nested (list inside list):

```
<ol>
  <li>Coffee</li>
  <li>Tea
    <ol>
      <li>Black tea</li>
      <li>Green tea</li>
    </ol>
  </li>
  <li>Milk</li>
</ol>
```

Note: A list item (`<li>`) can contain a new list, and other HTML elements, like images and links, etc.

Ordered HTML List - The Type Attribute

The `type` attribute of the `<ol>` tag, defines the type of the list item marker:

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

HTML class Attribute

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 1

```
<!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>Bangalore</h2>
  <p>Bangalore is the capital of Karnataka.</p>
</div>

<div class="city">
  <h2>Delhi</h2>
  <p>Delhi is the capital of India.</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:

```
<!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>
```

Note: The class name is case sensitive!, The `class` attribute can be used on **any** HTML element.

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. . 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>` point 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>
```

HTML id Attribute

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

```
<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>

<!-- 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>
```

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 = "Ha
ve a nice day!";
}
</script>
```

HTML Iframes

An HTML iframe is used to display a web page within a web page.

HTML Iframe Syntax

The HTML `<iframe>` tag specifies an inline frame.

An inline frame is used to embed another document within the current HTML document.

Syntax

```
<iframe src="url" title="description"></iframe>
```

Tip: It is a good practice to always include a `title` attribute for the `<iframe>`. This is used by screen readers to read out what the content of the iframe is.

Iframe - Set Height and Width

Use the `height` and `width` attributes to specify the size of the iframe.

The height and width are specified in pixels by default:

Example

```
<iframe src="demo_iframe.htm" height="200" width="300
" title="Iframe Example"></iframe>
```

Or you can add the `style` attribute and use the CSS `height` and `width` properties:

Example

```
<iframe src="demo_iframe.htm" style="height:200px;wid
th:300px;" title="Iframe Example"></iframe>
```

Iframe - Remove the Border

By default, an iframe has a border around it.

To remove the border, add the `style` attribute and use the CSS `border` property:

Example

```
<iframe src="demo_iframe.htm" style="border:none;" ti
tle="Iframe Example"></iframe>
```

With CSS, you can also change the size, style and color of the iframe's border:

Example

```
<iframe src="demo_iframe.htm" style="border:2px solid
red;" title="Iframe Example"></iframe>
```

Iframe - Target for a Link

An iframe can be used as the target frame for a link.

The `target` attribute of the link must refer to the `name` attribute of the iframe:

Example

```
<iframe src="demo_iframe.htm" name="iframe_a" title="
Iframe Example"></iframe>

<p><a href="https://
www.softpro9.com" target="iframe_a">SoftPro9</a></p>
```

HTML iframe Tag

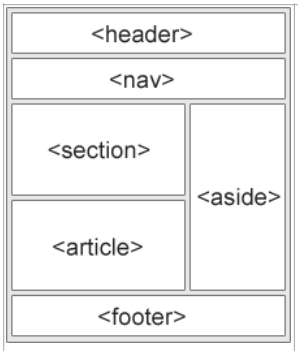
Tag	Description
<iframe>	Defines an inline frame

HTML Layout Elements and Techniques

Websites often display content in multiple columns (like a magazine or a newspaper).

HTML Layout Elements

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



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

HTML Layout Techniques

There are four different techniques to create multicolumn layouts. Each technique has its pros and cons:

- CSS framework
- CSS float property
- CSS flexbox
- CSS grid

CSS Frameworks

If you want to create your layout fast, you can use a CSS framework

CSS Float Layout

It is common to do entire web layouts using the CSS `float` property. Float is easy to learn - you just need to remember how the `float` and `clear` properties

work. **Disadvantages:** Floating elements are tied to the document flow, which may harm the flexibility.

CSS Flexbox Layout

Use of flexbox ensures that elements behave predictably when the page layout must accommodate different screen sizes and different display devices.

CSS Grid Layout

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

HTML Forms

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

The <form> Element

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

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.

A form with input fields for text:

```
<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>
```

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.

A form with radio buttons:

```
<p>Choose your favorite Web language:</p>

<form>
  <input type="radio" id="html" name="fav_language" v
alue="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" va
lue="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_langu
age" value="JavaScript">
```

```
  <label for="javascript">JavaScript</label>
</form>
```

Checkboxes

The `<input type="checkbox">` defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

A form with checkboxes:

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

The Submit Button

The `<input type="submit">` defines a button for submitting the form data to a form-handler.

The form-handler is typically a file on the server with a script for processing input data.

The form-handler is specified in the form's `action` attribute

A form with a submit button:

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

```
  <input type="text" id="fname" name="fname" value="J
ohn"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="D
oe"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTML Form Attributes

The different attributes for the HTML `<form>` element.

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:

On submit, send form data to "action_page.php":

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="J
ohn"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="D
oe"><br><br>
  <input type="submit" value="Submit">
</form>
```

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.

the submitted result will open in a new browser tab:

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

The Method Attribute

The `method` attribute specifies the HTTP method to be used when submitting the form data.

The form-data can be sent as URL variables (with `method="get"`) or as HTTP post transaction (with `method="post"`).

The default HTTP method when submitting form data is GET.

This example uses the GET method when submitting the form data:

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

This example uses the POST method when submitting the form data:

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

Notes on GET:

- Appends the form data to the URL, in name/value pairs
- NEVER use GET to send sensitive data! (the submitted form data is visible in the URL!)
- The length of a URL is limited (2048 characters)
- Useful for form submissions where a user wants to bookmark the result
- GET is good for non-secure data, like query strings in Google

Notes on POST:

- Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL)
- POST has no size limitations, and can be used to send large amounts of data.
- Form submissions with POST cannot be bookmarked

Tip: Always use POST if the form data contains sensitive or personal information!

The Autocomplete Attribute

The `autocomplete` attribute specifies whether a form should have autocomplete on or off.

When autocomplete is on, the browser automatically complete values based on values that the user has entered before.

A form with autocomplete on:

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

The Novalidate Attribute

The `novalidate` attribute is a boolean attribute.

When present, it specifies that the form-data (input) should not be validated when submitted.

A form with a novalidate attribute:

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

List of All <form> Attributes

Attribute	Description
accept-charset	Specifies the character encodings used for form submission
action	Specifies where to send the form-data when a form is submitted
autocomplete	Specifies whether a form should have autocomplete on or off
enctype	Specifies how the form-data should be encoded when submitting it to the server (only for method="post")
method	Specifies the HTTP method to use when sending form-data
name	Specifies the name of the form

novalidate	Specifies that the form should not be validated when submitted
rel	Specifies the relationship between a linked resource and the current document
target	Specifies where to display the response that is received after submitting the form

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

```
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname">
```

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

```
<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>
```

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

```
<option value="fiat" selected>Fiat</option>
```

Visible Values:

Use the `size` attribute to specify the number of visible values:

Example

```
<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>
```

Allow Multiple Selections:

Use the `multiple` attribute to allow the user to select more than one value:

Example

```
<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>
```

The <textarea> Element

The `<textarea>` element defines a multi-line input field (a text area):

Example

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The `rows` attribute specifies the visible number of lines in a text area.

The `cols` attribute specifies the visible width of a text area.

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

```
<button type="button" onclick="alert('Hello
World!')">Click Me!</button>
```

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

```
<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>
```

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

```
<form action="/action_page.php">
  <input list="browsers">
  <datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
  </datalist>
</form>
```

The <output> Element

The `<output>` element represents the result of a calculation

Example

Perform a calculation and show the result in an `<output>` element:

```
<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>
```

HTML Form Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control
<textarea>	Defines a multiline input control (text area)
<label>	Defines a label for an <input> element
<fieldset>	Groups related elements in a form
<legend>	Defines a caption for a <fieldset> element
<select>	Defines a drop-down list
<optgroup>	Defines a group of related options in a drop-down list
<option>	Defines an option in a drop-down list
<button>	Defines a clickable button
<datalist>	Specifies a list of pre-defined options for input controls
<output>	Defines the result of a calculation

HTML Input Types

Describes the different types for the HTML `<input>` element.

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"><b
r>
  <label for="pwd">Password:</label><br>
  <input type="password" id="pwd" name="pwd">
</form>
```

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="J
ohn"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="D
oe"><br><br>
  <input type="submit" value="Submit">
</form>
```

If you omit the submit button's value attribute, the button will get a default text:

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="J
ohn"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="D
oe"><br><br>
  <input type="submit" value="Submit">
  <input type="reset">
</form>
```

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" v
alue="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" va
lue="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_langu
age" 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 Image

The `<input type="image">` defines an image as a submit button.

The path to the image is specified in the `src` attribute.

Example

```
<form>
<input type="image" src="img_submit.gif" alt="Submit"
 width="48" height="48">
</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 lets 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" valu
e="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="bdayzzmonth" 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

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 25:

Example

```
<form>
  <label for="quantity">Quantity:</label>
  <input type="number" id="quantity" name="quantity"
min="0" max="100" step="10" value="25">
</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:

```
<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="J
ohn"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="D
oe">
</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="J
ohn" readonly><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="D
oe">
</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="J
ohn" disabled><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="D
oe">
</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:

```
<form>
  <label for="files">Select files:</label>
  <input type="file" id="files" name="files" multiple
>
</form>
```

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):

```
<form>
  <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">
</form>
```

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}">
</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" re
quired>
</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 intervals:

```
<form>
  <label for="points">Points:</label>
  <input type="number" id="points" name="points" step
="3">
</form>
```

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" autofocu
s><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</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="Submi
t" 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>:

```
<form>
  <input list="browsers">
  <datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
```

```
    <option value="Safari">
  </datalist>
</form>
```

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" autocom
plete="off"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTML Form and Input Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control

HTML Geolocation API

The HTML Geolocation API is used to locate a user's position.

Locate the User's Position

The HTML Geolocation API is used to get the geographical position of a user.

Since this can compromise privacy, the position is not available unless the user approves it.

Note: Geolocation is most accurate for devices with GPS, like smartphones.

Using HTML Geolocation

The `getCurrentPosition()` method is used to return the user's position.

The example below returns the latitude and longitude of the user's position:

Example

```
<script>
var x = document.getElementById("demo");
```

```
function getLocation() {
  if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(showPosition
);
  } else {
    x.innerHTML = "Geolocation is not supported by
this browser.";
  }
}

function showPosition(position) {
  x.innerHTML = "Latitude: " +
position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
```

Example explained:

- Check if Geolocation is supported
- If supported, run the getCurrentPosition() method. If not, display a message to the user
- If the getCurrentPosition() method is successful, it returns a coordinates object to the function specified in the parameter (showPosition)
- The showPosition() function outputs the Latitude and Longitude

The example above is a very basic Geolocation script, with no error handling.

Handling Errors and Rejections

The second parameter of the `getCurrentPosition()` method is used to handle errors. It specifies a function to run if it fails to get the user's location:

Example

```
function showError(error) {
  switch(error.code) {
    case error.PERMISSION_DENIED:
      x.innerHTML = "User denied the request for
Geolocation."
      break;
    case error.POSITION_UNAVAILABLE:
      x.innerHTML = "Location information is
unavailable."
      break;
    case error.TIMEOUT:
      x.innerHTML = "The request to get user location
timed out."
      break;
    case error.UNKNOWN_ERROR:
      x.innerHTML = "An unknown error occurred."
      break;
  }
}
```

Location-specific Information

This page has demonstrated how to show a user's position on a map.

Geolocation is also very useful for location-specific information, like:

- Up-to-date local information
- Showing Points-of-interest near the user
- Turn-by-turn navigation (GPS)

The getCurrentPosition() Method - Return Data

The `getCurrentPosition()` method returns an object on success. The latitude, longitude and accuracy properties are always returned. The other properties are returned if available:

Property	Returns
coords.latitude	The latitude as a decimal number (always returned)
coords.longitude	The longitude as a decimal number (always returned)
coords.accuracy	The accuracy of position (always returned)
coords.altitude	The altitude in meters above the mean sea level (returned if available)
coords.altitudeAccuracy	The altitude accuracy of position (returned if available)
coords.heading	The heading as degrees clockwise from North (returned if available)
coords.speed	The speed in meters per second (returned if available)
timestamp	The date/time of the response (returned if available)

Geolocation Object - Other interesting Methods

The Geolocation object also has other interesting methods:

- `watchPosition()` - Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).
- `clearWatch()` - Stops the `watchPosition()` method.

The example below shows the `watchPosition()` method. You need an accurate GPS device to test this (like smartphone):

Example

```
<script>
var x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {

navigator.geolocation.watchPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by
this browser.";
  }
}
function showPosition(position) {
  x.innerHTML = "Latitude: " +
position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
```

HTML Drag and Drop API

In HTML, any element can be dragged and dropped.

Drag and Drop

Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

HTML Drag and Drop Example

The example below is a simple drag and drop example:

Example

```
<!DOCTYPE HTML>
<html>
<head>
```

```
<script>
function allowDrop(ev) {
  ev.preventDefault();
}

function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data)
);
}
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)" ondragover="allow
Drop(event)"></div>

<img id="drag1" src="img_logo.gif" draggable="true"
ondragstart="drag(event)" width="336" height="69">

</body>
</html>
```

Make an Element Draggable

First of all: To make an element draggable, set the `draggable` attribute to true:

```
<img draggable="true">
```

What to Drag - ondragstart and setData()

Then, specify what should happen when the element is dragged.

In the example above, the `ondragstart` attribute calls a function, drag(event), that specifies what data to be dragged.

The `dataTransfer.setData()` method sets the data type and the value of the dragged data:

```
function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}
```

In this case, the data type is "text" and the value is the id of the draggable element ("drag1").

Where to Drop - ondragover

The `ondragover` event specifies where the dragged data can be dropped.

By default, data/elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element.

This is done by calling the `event.preventDefault()` method for the ondragover event:

```
event.preventDefault()
```

Do the Drop - ondrop

When the dragged data is dropped, a drop event occurs.

In the example above, the ondrop attribute calls a function, drop(event):

```
function drop(ev) {
  ev.preventDefault();
```

```
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data)
);
}
```

Code explained:

- Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop)
- Get the dragged data with the dataTransfer.getData() method. This method will return any data that was set to the same type in the setData() method
- The dragged data is the id of the dragged element ("drag1")
- Append the dragged element into the drop element

HTML Web Storage API

HTML web storage; better than cookies.

What is HTML Web Storage?

With web storage, web applications can store data locally within the user's browser.

Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.

Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

HTML Web Storage Objects

HTML web storage provides two objects for storing data on the client:

- `window.localStorage` - stores data with no expiration date
- `window.sessionStorage` - stores data for one session (data is lost when the browser tab is closed)

Before using web storage, check browser support for localStorage and sessionStorage:

```
if (typeof(Storage) !== "undefined") {
  // Code for localStorage/sessionStorage.
} else {
  // Sorry! No Web Storage support..
}
```

The localStorage Object

The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Example

```
// Store
localStorage.setItem("lastname", "Smith");

// Retrieve
document.getElementById("result").innerHTML =
localStorage.getItem("lastname");
```

Example explained:

- Create a localStorage name/value pair with name="lastname" and value="Smith"

- Retrieve the value of "lastname" and insert it into the element with id="result"

The example above could also be written like this:

```
// Store
localStorage.lastname = "Smith";
// Retrieve
document.getElementById("result").innerHTML =
localStorage.lastname;
```

The syntax for removing the "lastname" localStorage item is as follows:

```
localStorage.removeItem("lastname");
```

Note: Name/value pairs are always stored as strings. Remember to convert them to another format when needed!

The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter:

Example

```
if (localStorage.clickcount) {
  localStorage.clickcount =
Number(localStorage.clickcount) + 1;
} else {
  localStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You
have clicked the button " +
localStorage.clickcount + " time(s).";
```

The sessionStorage Object

The `sessionStorage` object is equal to the localStorage object, **except** that it stores the data for only one session. The data is deleted when the user closes the specific browser tab.

The following example counts the number of times a user has clicked a button, in the current session:

Example

```
if (sessionStorage.clickcount) {
  sessionStorage.clickcount =
Number(sessionStorage.clickcount) + 1;
} else {
  sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You
have clicked the button " +
sessionStorage.clickcount + " time(s) in this
session.";
```

HTML Web Workers API

A web worker is a JavaScript running in the background, without affecting the performance of the page.

What is a Web Worker?

When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

HTML Web Workers Example

The example below creates a simple web worker that count numbers in the background:

Create a Web Worker File

Now, let's create our web worker in an external JavaScript.

Here, we create a script that counts. The script is stored in the "demo_workers.js" file:

```
var i = 0;

function timedCount() {
  i = i + 1;
  postMessage(i);
  setTimeout("timedCount()",500);
}

timedCount();
```

The important part of the code above is the `postMessage()` method - which is used to post a message back to the HTML page.

Note: Normally web workers are not used for such simple scripts, but for more CPU intensive tasks.

Create a Web Worker Object

Now that we have the web worker file, we need to call it from an HTML page.

The following lines checks if the worker already exists, if not - it creates a new web worker object and runs the code in "demo_workers.js":

```
if (typeof(w) == "undefined") {
  w = new Worker("demo_workers.js");
}
```

Then we can send and receive messages from the web worker.

Add an "onmessage" event listener to the web worker.

```
w.onmessage = function(event){
  document.getElementById("result").innerHTML = event
.data;
};
```

When the web worker posts a message, the code within the event listener is executed. The data from the web worker is stored in event.data.

Terminate a Web Worker

When a web worker object is created, it will continue to listen for messages (even after the external script is finished) until it is terminated.

To terminate a web worker, and free browser/computer resources, use the `terminate()` method:

```
w.terminate();
```

Reuse the Web Worker

If you set the worker variable to undefined, after it has been terminated, you can reuse the code:

```
w = undefined;
```

Full Web Worker Example Code

We have already seen the Worker code in the .js file. Below is the code for the HTML page:

Example

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

<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>

<script>
var w;

function startWorker() {
  if (typeof(Worker) !== "undefined") {
    if (typeof(w) == "undefined") {
      w = new Worker("demo_workers.js");
    }
    w.onmessage = function(event) {

document.getElementById("result").innerHTML = event.d
ata;
    };
  } else {

document.getElementById("result").innerHTML = "Sorry!
No Web Worker support.";
  }
}

function stopWorker() {
  w.terminate();
  w = undefined;
}
</script>

</body>
</html>
```

HTML SSE API

Server-Sent Events (SSE) allow a web page to get updates from a server.

Server-Sent Events - One Way Messaging

A server-sent event is when a web page automatically gets updates from a server.

This was also possible before, but the web page would have to ask if any updates were available. With server-sent events, the updates come automatically.

Examples: Facebook/Twitter updates, stock price updates, news feeds, sport results, etc.

Receive Server-Sent Event Notifications

The EventSource object is used to receive server-sent event notifications:

Example

```
var source = new EventSource("demo_sse.php");
source.onmessage = function(event) {
  document.getElementById("result").innerHTML += even
t.data + "<br>";
};
```

Example explained:

- Create a new EventSource object, and specify the URL of the page sending the updates (in this example "demo_sse.php")
- Each time an update is received, the onmessage event occurs
- When an onmessage event occurs, put the received data into the element with id="result"

Check Server-Sent Events Support

In the tryit example above there were some extra lines of code to check browser support for server-sent events:

```
if(typeof(EventSource) !== "undefined") {
  // Yes! Server-sent events support!
  // Some code.....
} else {
  // Sorry! No server-sent events support..
}
```

Server-Side Code Example

For the example above to work, you need a server capable of sending data updates (like PHP or ASP).

The server-side event stream syntax is simple. Set the "Content-Type" header to "text/event-stream". Now you can start sending event streams.

Code in PHP (demo_sse.php):

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

$time = date('r');
echo "data: The server time is: {$time}\n\n";
```

```
flush();
?>
```

Code in ASP (VB) (demo_sse.asp):

```
<%
Response.ContentType = "text/event-stream"
Response.Expires = -1
Response.Write("data: The server time is: " & now())
Response.Flush()
%>
```

Code explained:

- Set the "Content-Type" header to "text/event-stream"
- Specify that the page should not cache
- Output the data to send (**Always** start with "data: ")
- Flush the output data back to the web page

The EventSource Object

In the examples above we used the onmessage event to get messages. But other events are also available:

Events	Description
onopen	When a connection to the server is opened
onmessage	When a message is received
onerror	When an error occurs