HTML

(Hypertext Markup Language) is not a programming language; it is a *markup language* used to tell your browser how to structure the web pages you visit. It can be as complicated or as simple as the web developer wishes it to be.

HTML 4 vs HTML5

If HTML was fine for over a decade, why was it updated in 2014? The most significant difference between older versions of HTML vs HTML5 is the integration of **video** and **audio** into the language's specifications. Additionally, HTML5 includes the following updates:

- Deprecated elements like center, font, and strike have been dropped
- Improved parsing rules allow for more flexible parsing and compatibility
- New elements including video, time, nav, section, progress, meter, aside and canvas
- New input attributes including email, URL, dates and times
- New attributes including charset, async and ping
- New APIs that offer offline caching, drag-and-drop support and more
- Support for vector graphics without the aid of programs like Silverlight or Flash
- Support for MathML to allow better display of mathematical notations
- JavaScript can now run in the background thanks to the JS Web worker API
- Global attributes such as tabindex, repeat and id can be can be applied for all elements

Other Notable Improvements

Now that a few years have passed since the launch of HTML5, several major companies have converted their websites, and many developers are sharing their

opinions on HTML vs HTML5. Features that have been commonly cited as favorites include:

- The **DOCTYPE** declaration for HTML5 is very simple:

`<!DOCTYPE html>`

- The **character encoding (charset)** declaration is also very simple:

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

The default character encoding in HTML5 is UTF-8.

- **HTML5 Semantic elements:**

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of non-semantic elements: `<div>` and `<span>` - Tells nothing about its content.

Examples of semantic elements: `<form>`, `<table>`, and `<article>` - Clearly defines its content.

Many web sites contain HTML code like: to indicate navigation, header, and footer.

HTML5 offers new semantic elements to define different parts of a web page:

- <article>
- <aside>
- <details>
- <figcaption>
- <figure>
- <footer>
- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>

- <time>

Ex:

```
<figure>
  <img src="pic_trulli.jpg" alt="Trulli">
  <figcaption>Fig1. - Trulli, Puglia, Italy.</figcaption>
</figure>
```

- **Add new elements to Html**

You can also add new elements to an HTML page with a browser trick.

This example adds a new element called <myHero> to an HTML page, and defines a style for it:

```
<head>
<script>document.createElement("myHero")</script>
<style>
myHero {
        display: block;
        background-color: #dddddd;
        padding: 50px;
        font-size: 30px;
}
</style>
</head>
<body>

<h1>A Heading</h1>
<myHero>My Hero Element</myHero>

</body>
```

- **No More Cookies Thanks to Local Storage**

Although its already been mentioned, support for local storage has been a real game changer. Before HTML5, cookies were the only reliable way to store state information. Of course, cookies hold a very limited amount of data, and some web users disable

cookies by default. HTML5's localStorage object provides developers a way to work around the stateless nature of the HTTP protocol.

Because it is part of the global window namespace, localStorage may be accessed from any point within scripts.

- **New attributes**

```
<input type="text" name="lname" autofocus />
 <input type="email" name="email" />
<input type="text" placeholder="Say something" />
<input type="text" name="someInput" required>
<input type="text" name="username" pattern="[A-Za-z]{4,10}" >
<input type="range">
<div contenteditable="true" />
```

- **Script and Link Tags No Longer Require Type Attribute**

Since it's now implied that script and link tags refer to scripts and stylesheets respectively, the need for the type attribute has been eliminated.

For more , https://code.tutsplus.com/tutorials/28-html5-features-tips-and-techniques-you-must-know--net-13520

Comparison of browser engines:

https://en.wikipedia.org/wiki/Comparison_of_browser_engines

How web pages are loaded by web browser:

Ref : https://varvy.com/pagespeed/display.html

Responsive Web Design

https://docs.google.com/document/d/1mnnzAsdhRaeDb4GHXxmNlvq1vyr529fFtdSuufHbQmY/edit

---

Local storage:

The read-only localStorage property allows you to access a Storage object for the Document's origin; the stored data is saved across browser sessions. localStorage is similar to sessionStorage, except that while **data stored in localStorage has no expiration time, data stored in sessionStorage gets cleared when the page session ends — that is, when the page is closed.**

```
localStorage.setItem('myCat', 'Tom');
var cat = localStorage.getItem('myCat');
localStorage.removeItem('myCat');
localStorage.clear(); // to clear all items
```

Session storage

The sessionStorage property allows you to access a session Storage object for the current origin. sessionStorage is similar to Window.localStorage; the only difference is while data stored in localStorage has no expiration set, data stored in sessionStorage gets cleared when the page session ends. A page session lasts for as long as the browser is open and survives over page reloads and restores. **Opening a page in a new tab or window will cause a new session to be initiated with the value of the top-level browsing context, which differs from how session cookies work.**

```
// Save data to sessionStorage
sessionStorage.setItem('key', 'value');

// Get saved data from sessionStorage
var data = sessionStorage.getItem('key');
```

```
// Remove saved data from sessionStorage
sessionStorage.removeItem('key');
// Remove all saved data from sessionStorage
sessionStorage.clear();
```

Block versus inline elements :

There are two important categories of elements in HTML, which you should know about — block-level elements and inline elements.

- Block-level elements form a visible block on a page — they will appear on a new line from whatever content went before it, and any content that goes after it will also appear on a new line. Block-level elements tend to be structural elements on the page that represent, for example, paragraphs, lists, navigation menus, footers, etc. A block-level element wouldn't be nested inside an inline element, but it might be nested inside another block-level element.
- Inline elements are those that are contained within block-level elements and surround only small parts of the document's content, not entire paragraphs and groupings of content. An inline element will not cause a new line to appear in the document; they would normally appear inside a paragraph of text, for example an <a> element (hyperlink) or emphasis elements such as or .

Now take the following examples,

```
<em>first</em><em>second</em><em>third</em>
<p>fourth</p><p>fifth</p><p>sixth</p>
```

 is an inline element, so as you can see below, the first three elements sit on the same line as one another with no space in between. On the other hand, <p> is a block-level element, so each element appears on a new line, with space above and below each (the spacing is due to default CSS styling that the browser applies to paragraphs).

Empty elements

Not all elements follow the above pattern of opening tag, content, closing tag. Some elements consist only of a single tag, which is usually used to insert/embed something in the document at the place it is included. For example, the <img> element embeds an image file onto a page in the position it is included in:

```
<img src="https://raw.githubusercontent.com/mdn/beginner-html-site/gh-pages/images/firefox-icon.png">
```

Attributes:

Attributes contain extra information about the element which you don't want to appear in the actual content. In this case, the class attribute allows you to give the element an identifying name that can be later used to target the element with style information and other things.

An attribute should have:

1. A space between it and the element name (or the previous attribute, if the element already has one or more attributes.)
2. The attribute name, followed by an equals sign.
3. An attribute value, with opening and closing quote marks wrapped around it.

Boolean attributes:

You'll sometimes see attributes written without values — this is perfectly allowed. These are called boolean attributes, and they can only have one value, which is generally the same as the attribute name. As an example, take the disabled attribute, which you can assign to form input elements if you want them to be disabled (greyed out) so the user can't enter any data in them.

```
<input type="text" disabled>
```

Anatomy of an HTML document:

That wraps up the basics of individual HTML elements, but they aren't very useful on their own. Now we'll look at how individual elements are combined to form an entire HTML page:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My test page</title>
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

1. <!DOCTYPE html>

<!DOCTYPE html> is the shortest string of characters that counts as a valid doctype; that's all you really need to know.

1. <html></html>: The <html> element. This element wraps all the content on the entire page, and is sometimes known as the root element.
2. <head></head>: The <head> element. This element acts as a container for all the stuff you want to include on the HTML page that *isn't* the content you are showing to your page's viewers. This includes things like keywords and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more. You'll learn more about this in the next article in the series.
3. <meta charset="utf-8">: This element sets the character set your document should use to UTF-8, which includes most characters from the vast majority of human written languages. Essentially it can now handle any textual content you might put on it. There is no reason not to set this, and it can help avoid some problems later on.
4. <title></title>: The <title> element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in, and is used to describe the page when you bookmark/favourite it.
5. <body></body>: The <body> element. This contains *all* the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

Whitespace in HTML :

The following code snippets are equivalent:

```
<p>Dogs are silly.</p>
<p>Dogs       are
        silly.</p>
```

No matter how much whitespace you use (which can include space characters, but also line breaks), the HTML parser reduces each one down to a single space when rendering the code. So why use so much whitespace? The answer is readability — it is so much easier to understand what is going on in your code if you have it nicely formatted, and not just bunched up together in a big mess.

Entity references: including special characters in HTML:

In HTML, the characters <, >,",' and & are special characters. They are parts of the HTML syntax itself, so how do you include one of these characters in your text, for example if you really want to use an ampersand or less than sign, and not have it interpreted as code as some browsers may do?

We have to use character references — special codes that represent characters, and can be used in these exact circumstances. Each character reference is started with an ampersand (&), and ended by a semi-colon (;).

Literal character	Character reference equivalent
<	<
>	>
“	"
‘	'

& &

```
<p>In HTML, you define a paragraph using the &lt;p&gt; element.</p>
```

Html comments:

```
<!-- <p>I am!</p> →
```

Head element:

The head of an HTML document is the part that is not displayed in the web browser when the page is loaded. It contains information such as the page <title>, links to CSS (if you want to style your HTML content with CSS), links to custom favicons, and other metadata (data about the HTML, such as who wrote it, and important keywords that describe the document.)

Metadata: <meta> element:

There are a lot of different types of <meta> element that can be included in your page's <head>

Specifying your document's character encoding:

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

This element simply specifies the document's character encoding — the character set that the document is permitted to use. utf-8 is a universal character set that includes pretty much any character from any human language. This means that your web page will be able to handle displaying any language; it's therefore a good idea to set this on every web page you create! For example, your page could handle English and Japanese just fine:

If you set your character encoding to ISO-8859-1, for example (the character set for the Latin alphabet), your page rendering would be all messed up:

Adding author and description:

Many <meta> elements include name and content attributes:

- name specifies the type of meta element it is; what type of information it contains.
- content specifies the actual meta content.

Two such meta elements that are useful to include on your page define the author of the page, and provide a concise description of the page. Let's look at an example:

```
<meta name="author" content="Chris Mills">
<meta name="description" content="The MDN Learning Area aims to provide
complete beginners to the Web with all they need to know to get
started with developing web sites and applications.">
```

Specifying an author is useful in a few ways: it is useful to be able to work out who wrote the page, if you want to contact them with questions about the content. Some content management systems have facilities to automatically extract page author information and make it available for such purposes.

Specifying a description that includes keywords relating to the content of your page is useful as it has the potential to make your page appear higher in relevant searches performed in search engines (such activities are termed Search Engine Optimization, or SEO.)

Setting the primary language of the document:

Finally, it's worth mentioning that you can (and really should) set the language of your page. This can be done by adding the lang attribute to the opening HTML tag (as seen in the meta-example.html and shown below.)

```
<html lang="en-US">
```

This is useful in many ways. Your HTML document will be indexed more effectively by search engines if its language is set (allowing it to appear correctly in language-specific results, for example), and it is useful to people with visual impairments using screen readers (for example, the word "six" exists in both French and English, but is pronounced differently.)

You can also set subsections of your document to be recognised as different languages. For example, we could set our Japanese language section to be recognised as Japanese, like so:

```
<p>Japanese example: <span lang="jp">ご飯が熱い。</span>.</p>
```

!important

There is one thing that is even more specific than IDs: !important. !important can be applied to specific attributes instead of full rules. It will override *any* style no matter how specific it is. As a result, it should almost never be used. Once !important is used, it is very hard to override.

Syntax:

```
P {
Color:blue !important;
}
```

Block level elements in HTML:

Block level elements in HTML:

<address>	<article>	<aside>	<blockquote>	<canvas>	<dd>	
<dl>	<dt>	<fieldset>	<figcaption>	<figure>	<footer>	<form>
<h1>-<h6>	<header>	<hr>		<main>	<nav>	<noscript>
	<output>	<p>	<pre>	<section>	<table>	<tfoot>
	<video>					

Position: absolute

The element is placed at that position within its last ancestor element which has a position attribute of anything other than static (page elements default to static when no position attribute specified), or the document body (browser viewport) if no such ancestor exists.

For example, if I had code like this

```
<body> <div style="position:absolute; left: 20px; top: 20px;"></div> </body>
```

...the would be positioned 20px from the top of the browser viewport, and 20px from the left edge of same.

However, if I did something like this:

```
<div id="outer" style="position:relative">
        <div id="inner" style="position:absolute; left: 20px; top: 20px;"> </div>
</div>
```
...then the inner div would be positioned 20px from the top of the outer div, and 20px from the left edge of same, because the outer div isn't positioned with position:static because we've explicitly set it to use position:relative.

https://css-tricks.com/absolute-relative-fixed-positioining-how-do-they-differ/

Z-index:

When boxes on a web page have a combination of different positions, the boxes (and therefore, their content) can overlap with each other, making the content difficult to read or consume.

The z-index property controls how far "back" or how far "forward" an element should appear on the web page.

The z-index property accepts integer values. Depending on their values, the integers instruct the browser on the order in which elements should be displayed on the web page.

```
.box-top {
  background-color: Aquamarine;
  position: relative;
  z-index: 2;
}

.box-bottom {
  background-color: DeepSkyBlue;
  position: absolute;
  top: 20px;
  left: 50px;
  z-index: 1;
}
```

In the example above, we set the .box-top position to relative and the z-index to 2. We changed position to relative, because the z-index property does *not* work on static elements.

The z-index of 2 moves the .box-top element forward, because it is greater than the .box-bottom z-index, 1.

Try out this

Ex:1

```
<div style="position:relative; width:100px; height: 100px; border:1px solid red; "> </div>
<div style="position:absolute; width:100px; height: 100px; border:1px solid green; "> </div>
```

Ex:2

```
<div style="position:relative; width:100px; height: 100px; border:1px solid red; "> </div>
<div style="position:absolute; width:100px; height: 100px; border:1px solid green; top:0px;">
</div>
```

Ex:3

```
<div style="position:relative; width:100px; height: 100px; border:1px solid red; "> </div>
<div style="position:absolute; width:100px; height: 100px; border:1px solid green;"> </div>
<div style="position:fixed; width:100px; height: 100px; border:1px solid blue; "> </div>
```

Ex:4

```
<div style="position:relative; width:100px; height: 100px; border:1px solid red; "> </div>
<div style="position:fixed; width:100px; height: 100px; border:1px solid green;"> </div>
<div style="position:absolute; width:100px; height: 100px; border:1px solid blue; "> </div>
```

Ex:5

```
<div style="position:absolute; width:100px; height: 100px; border:1px solid red; "> </div>
<div style="position:fixed; width:100px; height: 100px; border:1px solid green;"> </div>
<div style="position:relative; width:10px; height: 10px; border:1px solid blue; "> </div>
<div style="position:relative; width:200px; height: 200px; border:1px solid black; "> </div>
```

Ex:6:

```
<style>

#main {
width:200px;
height: 200px;
border:1px solid black;
}


.fixed_block {
position:fixed;
width:50px;
height:50px;
border:1px solid green;
}
```

```
.relative_block {
position:relative;
width:50px;
height:50px;
border:1px solid red;
}

.absolute_block {
position:absolute;
width:50px;
height:50px;
border:1px solid blue;
}

</style>
<div id="main">
        <div class="fixed_block"> </div>
        <div class="relative_block"> </div>
        <div class="relative_block"> </div>
        <div class="absolute_block"> </div>
</div>
```

Ex:7

```
<style>

#main {
width:200px;
height: 200px;
border:1px solid black;
}


.fixed_block {
position:fixed;
width:50px;
height:50px;
border:1px solid green;
}

.relative_block {
position:relative;
width:50px;
height:50px;
border:1px solid red;
```

```
}

.absolute_block {
position:absolute;
width:50px;
height:50px;
border:1px solid blue;
z-index:3;
}

</style>
<div id="main">
        <div class="fixed_block"> </div>
        <div class="relative_block"> </div>
        <div class="absolute_block"> </div>
        <div class="relative_block"> </div>
</div>
```

Ex:8

Position: sticky

An element with `position: sticky;` is positioned based on the user's scroll position.

A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

Note: Internet Explorer, Edge 15 and earlier versions do not support sticky positioning. Safari requires a -webkit- prefix (see example below). You must also specify at least one of `top`, `right`, `bottom` or `left` for sticky positioning to work.

In this example, the sticky element sticks to the top of the page (`top: 0`), when you reach its scroll position.

```
<style>
div.sticky {
  position: -webkit-sticky;
  position: sticky;
  top: -10px;
  padding: 5px;
  background-color: #cae8ca;
```

```
  border: 2px solid #4CAF50;
}
</style>
<p>Try to <b>scroll</b> inside this frame to understand how sticky positioning works.</p>
<p>Note: IE/Edge 15 and earlier versions do not support sticky position.</p>

<div class="sticky">I am sticky!</div>

<div style="padding-bottom:2000px">
  <p>In this example, the sticky element sticks to the top of the page (top: 0), when you reach its scroll position.</p>
  <p>Scroll back up to remove the stickyness.</p>
  <p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p>
  <p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p>
</div>
```

Inline Display:

Every HTML element has a default display value that dictates if it can share horizontal space with other elements. Some elements fill the entire browser from left to right regardless of the size of their content. Other elements only take up as much horizontal space as their content requires and can be directly next to other elements.

In this lesson, we'll cover three values for the display property: inline, block, and inline-block. The default display for some tags, such as , , and <a>, is called *inline*. Inline elements have a box that wraps tightly around their content, only taking up the amount of space necessary to display their content and not requiring a new line after each element.

Elements that are block-level by default include all levels of heading elements (<h1> through <h6>), <p>, and <footer>.

Html commands:

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

------------------------------------------------------------------

<meta name="description" content="Free Web tutorials">

<meta charset="UTF-8">

<meta name="keywords" content="HTML, CSS, XML, JavaScript">

<meta name="author" content="John Doe">

<meta http-equiv="refresh" content="30">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

------------------------------------------------------------------

<style> </style>

<link rel="stylesheet" type="text/css" href="styles.css"> --for linking css sheet

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

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

<span> </span> -- defines a section in a document (inline)

<hr> -- horizontal line

<br>  -- line break

<pre> </pre> -- preserves line space

<b> </b>

<strong> </strong>

<q> </q>  -- quotes

<div> </div>

<!-- Write a comment here -->
```

```
<kbd>Ctrl + S</kbd> -- represent input as in keyboard.

<code> </code>

<var>E</var> = <var>mc</var><sup>2</sup>

<samp>Error!</samp> -- represent output from the program

<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
----------------------------------------------------------------

<table style="width:100%">
<caption>Monthly savings</caption>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
</table>

----------------------------------------------------------------

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

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

----------------------------------------------------------------
```

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

<p><a href="https://www.w3schools.com" target="iframe_a">W3Schools.com</a></p>

----------------------------------------------------------------------

<form action="/action_page.php" method="get/post"  target="_blank">
  <fieldset>
          <legend>Personal information:</legend>
           First name:<br>
           <input type="text" name="firstname" value=”Gautham” readonly>
           <br>
           Last name:<br>
           <input type="text" name="lastname" value=”Sundar” disabled>

          User password:<br>
           <input type="password" name="psw">

           <input type="radio" name="gender" value="male" checked> Male<br>
           <input type="radio" name="gender" value="female"> Female<br>
           <input type="radio" name="gender" value="other"> Other

          <input type="checkbox" name="vehicle1" value="Bike"> I have a bike<br>
          <input type="checkbox" name="vehicle2" value="Car"> I have a car

          <select name="cars">
           <option value="volvo">Volvo</option>
           <option value="saab" selected>Saab</option>
           <option value="fiat">Fiat</option>
           <option value="audi">Audi</option>
          </select>

          <textarea name="message" style="width:200px; height:600px">
          The cat was playing in the garden.
          </textarea>

           <input type="submit" value="Submit" formmethod="post">
           <input type="button" onclick="alert('Hello World!')" value="Click Me!">

          <input type="reset">

 </fieldset>

</form>
```

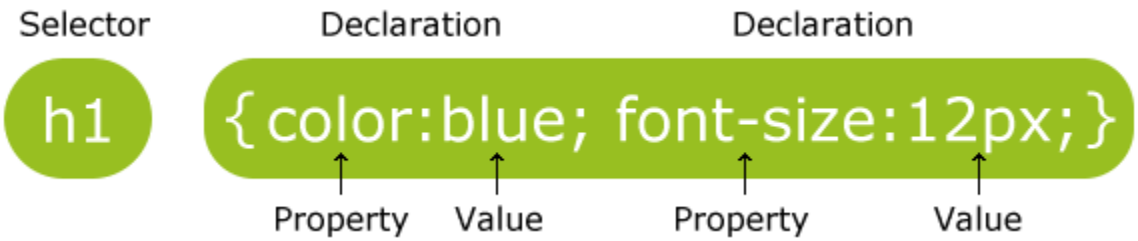
Note:

If the name attribute is omitted, the data of that input field will not be sent at all.

The **<fieldset>** element is used to group related data in a form
<input type="reset"> defines a **reset button** that will reset all form values to their default values

The **formmethod** attribute defines the HTTP method for sending form-data to the action URL.
The formmethod attribute overrides the method attribute of the <form> element.

Css:



Css selectors:

1. **Element selector**

The element selector selects elements based on the element name. You can select all <p> elements on a page . For e.g.,

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

2. Id selector

The id of an element should be unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

```
#para1 {
    text-align: center;
    color: red;
}
```

3. Class selector

The class selector selects elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the name of the class.

```
.center {
    text-align: center;
    color: red;
}
```

You can also specify that only specific HTML elements should be affected by a class.

In the example below, only <p> elements with class="center" will be center-aligned:

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

HTML elements can also refer to more than one class.

In the example below, the <p> element will be styled according to class="center" and to class="large":

```
<style>
p.center {
  text-align: center;
  color: red;
}

p.large {
  font-size: 300%;
}
</style>
<body>

<h1 class="center">This heading will not be affected</h1>
<p class="center">This paragraph will be red and center-aligned.</p>
```

```
<p class="center large">This paragraph will be red, center-aligned, and in a large font-size.</p>

</body>
```

4. Grouping selectors

```
<style>
h1 {
        text-align: center;
        color: red;
}

h2 {
        text-align: center;
        color: red;
}

p {
        text-align: center;
        color: red;
}
</style>
```

Is equivalent to

```
h1, h2, p {
        text-align: center;
        color: red;
}
```

Pseudo - classes:

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

```
a : link {}
a : visited {}
a : hover {}
```

a : active {}

Note : a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective! a:active MUST come after a:hover in the CSS definition in order to be effective! Pseudo-class names are not case-sensitive.

To match the first <p> element,

```
p:first-child {
        color: blue;
}
```

To match the first <i> element in all <p> element,

```
p i:first-child {
        color: blue;
}
```

To match all <i> elements in first <p> element

```
p:first-child i {
        color: blue;
}
```

Ref : https://www.w3schools.com/css/css_pseudo_classes.asp

Pseudo-elements:

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

The ::first-line pseudo-element is used to add a special style to the first line of a text.

```
p::first-line {
        color: #ff0000;
        font-variant: small-caps;
}

p::first-letter {
        color: #ff0000;
```

```
    font-size: xx-large;
}
```

Note: The ::first-line and ::first-letter pseudo-elements can only be applied to block-level elements

::before , ::after

The ::before , ::after pseudo-element can be used to insert some content before or after the content of an element.

The following example inserts an image before the content of each <h1> element:

```
h1::before {
    content: url(smiley.gif);
}
```

:: selection

The ::selection pseudo-element matches the portion of an element that is selected by a user.

```
::selection {
    color: red;
    background: yellow;
}
```

::placeholder

The ::placeholder CSS pseudo-element represents the placeholder text of a formelement.

```
<input placeholder="Type something here!">
```

```
input::placeholder {
  color: red;
  font-size: 1.2em;
  font-style: italic;
}
```

Ref : https://www.w3schools.com/css/css_pseudo_elements.asp

All major web browsers have a default stylesheet they use in the absence of an external stylesheet. These default stylesheets are known as *user agent stylesheets*. In this case, the term "user agent" is a technical term for the browser.

User agent stylesheets often have default CSS rules that set default values for padding and margin. This affects how the browser displays HTML elements, which can make it difficult for a developer to design or style a web page.

Many developers choose to reset these default values so that they can truly work with a clean slate.

```
* {
Padding:0px;
Margin : 0px;
}
```

The code in the example above resets the default margin and padding values of all HTML elements.

Note: What's the difference between `display: none` and `visibility: hidden`? An element with `display: none` will be completely removed from the web page. An element with `visibility: hidden`, however, will not be visible on the web page, but the space reserved for it will.

```
style="color:blue;margin-left:30px;"
```

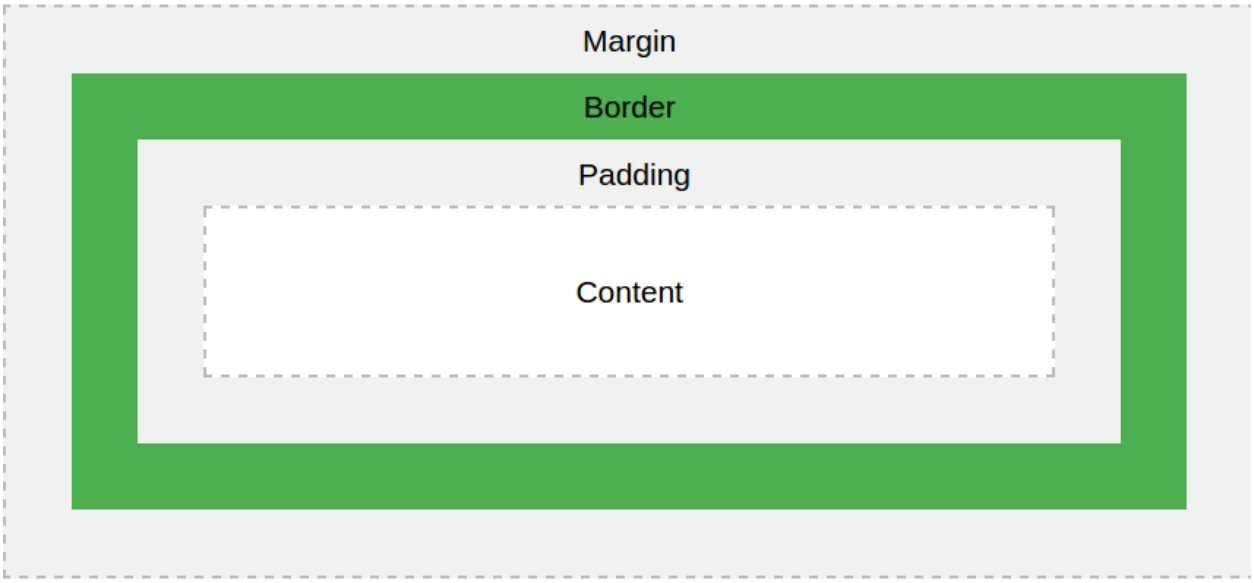
```
background-color: lightblue / #ff6347 / rgb(255,99,71) / rgba(255,9,71,0.5) / hsl(240, 400%, 80%) / hsl(240, 400%, 80%,0.1);
background-image: url("paper.gif");
background-repeat: repeat-x / repeat-y / no-repeat;
background-attachment: fixed;
background-position: right top;
```

When using the shorthand property the order of the property values is:

- background-color
- background-image
- background-repeat
- background-attachment
- Background-position

```
background: #ffffff url("img_tree.png") no-repeat right top;
```

```
border-style: dotted;
border : 1px solid blue;
border-collapse: collapse;
```



In CSS, the box-sizing property controls the type of box model the browser should use when interpreting a web page.

The default value of this property is content-box

Box model : Context-Box

```
h1 {
border: 1px solid black;
height : 200px;
width : 300px;
padding: 10px;
}
```

In the example above, a heading element's box has solid, black, 1 pixel thick borders. The height of the box is 200 pixels, while the width of the box is 300 pixels. A padding of 10 pixels has also been set on all four sides of the box's content.

Unfortunately, under the current box model, the border thickness and the padding will affect the dimensions of the box.

The 10 pixels of padding increases the height of the box to 220 pixels the width to 320 pixels. Next, the 1-pixel thick border increases the height to 221 pixels and the width to 321 pixels.

Under this box model, the border thickness and padding are added to the overall dimensions of the box. This makes it difficult to accurately size a box. Over time, this can also make all of a web page's content difficult to position and manage.

Box model: Border-Box

Fortunately, we can reset the entire box model and specify a new one: border-box.

```
* {
      box-sizing : border-box;
  }
```

The code in the example above resets the box model to `border-box` for all HTML elements. This new box model avoids the dimensional issues that exist in the former box model you learned about.

In this box model, the height and width of the box will remain fixed. The border thickness and padding will be included inside of the box, which means the overall dimensions of the box do not change.

```
h1 {
border: 1px solid black;
height : 200px;
width : 300px;
padding: 10px;
}
```

In the example above, the height of the box would remain at 200 pixels and the width would remain at 300 pixels. The border thickness and padding would remain entirely *inside* of the box.

Text Overflow:

The CSS `text-overflow` property specifies how overflowed content that is not displayed should be signaled to the user.

```
text-overflow: clip;
text-overflow: ellipsis;
text-overflow: inherit;
```

Word-wrapping:

The CSS word-wrap property allows long words to be able to be broken and wrap onto the next line.

If a word is too long to fit within an area, it expands outside. The word-wrap property allows you to force the text to wrap - even if it means splitting it in the middle of a word:

```
word-wrap: break-word;
```

Word breaking:

The CSS word-break property specifies line breaking rules.

```
word-break: keep-all;
word-break: break-all;
```

Word-wrapping vs word-breaking:

```
<style>
p {
   width: 140px;
   border: 1px solid #000000;
}

p.c {
   word-wrap: break-word;
}
p.d {
   word-break: break-all;
}
</style>

<p class="c">Thisissomeveryveryverylong word. This text will break at any character.</p>
<p class="d">Thisissomeveryveryverylong word. This text will break at any character.</p>
```

HSL color:

Hue is the first number. It refers to an angle on a color wheel. Red is 0 degrees, Green is 120 degrees, Blue is 240 degrees, and then back to Red at 360. You can see an example of this color wheel here.

Saturation refers to the intensity or purity of the color. If you imagine a line segment drawn from the center of the color wheel to the perimeter, the saturation is a point on that line segment. If you spin that line segment to different angles, you'll see how that saturation looks for different hues. The saturation increases towards 100% as the point gets closer to the edge (the color becomes more rich). The saturation decreases towards 0% as the point gets closer to the center (the color becomes more gray).

Lightness refers to how light or dark the color is. Halfway, or 50%, is normal lightness. Imagine a sliding dimmer on a light switch that starts halfway. Sliding the dimmer up towards 100% makes the color lighter, closer to white. Sliding the dimmer down towards 0% makes the color darker, closer to black.

HSL is convenient for adjusting colors. In RGB, making the color a little darker may affect all three color components. In HSL, that's as easy as changing the lightness value. HSL is also useful for making a set of colors that work well together by selecting various colors that have the same lightness and saturation but different hues.

```
background-color: hsl(240, 400%, 80%);
```

Alpha:

Alpha is a decimal number from zero to one. If alpha is zero, the color will be completely transparent. If alpha is one, the color will be opaque. The value for half transparent would be 0.5.

```
background-color: hsla(240, 400%, 80%, 0.1);
```

Fallback Fonts

What happens when a stylesheet requires a font that is not installed on a user's computer? Most computers have a small set of typefaces pre-installed. This small set includes serif fonts like Times New Roman and sans-serif fonts like Arial.

These pre-installed fonts serve as *fallback fonts* if the stylesheet specifies a font which is not installed on a user's computer.

To use fallback fonts, the following syntax is required:

```
h1 {
font-family: “Garamond”, “Times”, serif;
}
```

The CSS rule above says:

1. Use the Garamond font for all <h1>elements on the web page.
2. If Garamond is not available, use the Times font.
3. If Garamond and Times are not available, use any serif font pre-installed on the user's computer.

Linking Fonts:

1. When we have the link to the font of our choice, we can add the font to the <head> section of the HTML document, using the <link> tag and the href.

   ```
   <link href="https://fonts.googleapis.com/css?family=Space+Mono:400,700" rel="stylesheet" type="text/css" href="styles/style.css">
   ```

2. There are other ways to link non-user fonts that don't require the use of the <link> tag in the HTML document. CSS offers a way to import fonts directly into stylesheets with the @font-face property.

Go to https://fonts.googleapis.com/css?family=Space+Mono:400,700, and copy @font-face rules labeled latin.

```
@font-face {
  font-family: 'Space Mono';
  font-style: normal;
  font-weight: 400;
  src: local('Space Mono'), local('SpaceMono-Regular'), url(https://fonts.gstatic.com/s/spacemono/v2/adVweg3BJhE6r8jYmXseHfk_vArhqVIZ0nv9q090hN8.woff2) format('woff2');
  unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02C6, U+02DA, U+02DC, U+2000-
```

```
206F, U+2074, U+20AC, U+2212, U+2215;
}
```

---

float property:

```
<style>
img {
float:left;
}
</style>
<p>
<img src="pineapple.jpg" alt="Pineapple" style="width:170px;height:170px;">
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet.
</p>
```

clear property:

The clear property can have one of the following values:

- none - Allows floating elements on both sides. This is default
- left - No floating elements allowed on the left side
- right- No floating elements allowed on the right side
- both - No floating elements allowed on either the left or the right side
- inherit - The element inherits the clear value of its parent

Ex:1

```
<style>
.div1 {
   float: left;
   width: 100px;
   height: 50px;
```

```
    margin: 10px;
    border: 3px solid #73AD21;
}

.div2 {
    border: 1px solid red;
}

.div3 {
    float: left;
    width: 100px;
    height: 50px;
    margin: 10px;
    border: 3px solid #73AD21;
}

.div4 {
    border: 1px solid red;
    clear: left;
}
</style>

<h2>Without clear</h2>
<div class="div1">div1</div>
<div class="div2">div2 - Notice that div2 is after div1 in the HTML code. However, since div1 floats to the left, the text in div2 flows around div1.</div>
<br><br>

<h2>With clear</h2>
<div class="div3">div3</div>
<div class="div4">div4 - Here, clear: left; moves div4 down below the floating div3. The value "left" clears elements floated to the left. You can also clear "right" and "both".</div>
```

Ex:2

```
<style>

div {
    border: 3px solid #4CAF50;
    padding: 5px;
}
```

```
.img1, .img2 {
    float: right;
}

.clearfix {
            overflow:auto;
}
</style>

<div>

<img class="img1" src="pineapple.jpg" alt="Pineapple" width="170" height="170">

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...</div>

<div class="clearfix">

<img class="img2" src="pineapple.jpg" alt="Pineapple" width="170" height="170">

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...</div>
```

---

display property:

```
<style>
span.a {
    display: inline; /* the default for span */
    width: 100px;
    height: 140px; /* no effect because of inline */
    padding: 5px;
    border: 1px solid blue;
    background-color: red;
}

span.b {
    display: inline-block;
    width: 100px;
    height: 140px;
    padding: 5px;
    border: 1px solid blue;
```

```
    background-color: yellow;
}

span.c {
    display: block;
    width: 100px;
    height: 100px;
    padding: 5px;
    border: 1px solid blue;
    background-color: blue;
}
</style>

<h2>display: inline</h2>

<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat. <span class="a">Aliquam</span> <span class="a">venenatis</span> gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet. </div>

<h2>display: inline-block</h2>

<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat. <span class="b">Aliquam</span> <span class="b">venenatis</span> gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet. </div>

<h2>display: block</h2>

<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat. <span class="c">Aliquam</span> <span class="c">venenatis</span> gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet. </div>
```

---

border-boxing property:

```
<style>
* {
    box-sizing: border-box;
}
```

```
.box {
  float: left;
  width: 33.33%;
  padding: 50px;
}

.clearfix::after {
  content: "";
  clear: both;
  display: table;
}
</style>

<div class="clearfix">
 <div class="box" style="background-color:#bbb">
  <p>Some text inside the box.</p>
 </div>
 <div class="box" style="background-color:#ccc">
  <p>Some text inside the box.</p>
 </div>
 <div class="box" style="background-color:#ddd">
  <p>Some text inside the box.</p>
 </div>
</div>

<p> Explaining CSS Box Model </p>
```

CSS Transforms:

CSS transforms allow you to translate, rotate, scale, and skew elements.

A transformation is an effect that lets an element change shape, size and position.

CSS 2D Transforms:

Following are 2D transformation methods:

- translate()
- rotate()
- scale()
- skewX()
- skewY()

- matrix()

translate():

The translate() method moves an element from its current position (according to the parameters given for the X-axis and the Y-axis).

The following example moves the element 50 pixels to the right, and 100 pixels down from its current position:

```
div {
        width: 100px;
        height: 100px;
        -ms-transform: translate(50px, 100px); /* IE 9 */
        -webkit-transform: translate(50px, 100px); /* Safari */
        transform: translate(50px, 100px);
}
<div>
This div element is moved 50 pixels to the right, and 100 pixels down from its current position.
</div>
```

rotate():

The rotate() method rotates an element clockwise or counter-clockwise according to a given degree.

```
div {
        -ms-transform: rotate(20deg); /* IE 9 */
        -webkit-transform: rotate(20deg); /* Safari */
        transform: rotate(20deg);
}
```

Using negative values will rotate the element counter-clockwise.

scale():

The scale() method increases or decreases the size of an element (according to the parameters given for the width and height).

The following example increases the element to be two times of its original width, and three times of its original height:

```
div {
    -ms-transform: scale(2, 3); /* IE 9 */
    -webkit-transform: scale(2, 3); /* Safari */
    transform: scale(2, 3);
}
```

The following example decreases the element to be half of its original width and height:

```
div {
    -ms-transform: scale(0.5, 0.5); /* IE 9 */
    -webkit-transform: scale(0.5, 0.5); /* Safari */
    transform: scale(0.5, 0.5);
}
```

Note: scale(1.5) // increases both width & height

skewX():

The skewX() method skews an element along the X-axis by the given angle.

The following example skews the element 20 degrees along the X-axis:

```
div {
    -ms-transform: skewX(20deg); /* IE 9 */
    -webkit-transform: skewX(20deg); /* Safari */
    transform: skewX(20deg);
}
```

skewY():

The skewY() method skews an element along the Y-axis by the given angle.

The following example skews the element 20 degrees along the Y-axis:

```
div {
    -ms-transform: skewY(20deg); /* IE 9 */
    -webkit-transform: skewY(20deg); /* Safari */
```

```
    transform: skewY(20deg);
}
```

skew():

The skew() method skews an element along the X and Y-axis by the given angles.

The following example skews the element 20 degrees along the X-axis, and 10 degrees along the Y-axis:

```
div {
    -ms-transform: skew(20deg, 10deg); /* IE 9 */
    -webkit-transform: skew(20deg, 10deg); /* Safari */
    transform: skew(20deg, 10deg);
}
```

CSS 3D Transforms:

CSS allows you to format your elements using 3D transformations.

Here are some 3D transform methods:

- Matrix3d
- translate3d(x,y,z)
- translateX(x)
- translateY(y)
- translateZ(z)
- scale3d(x,y,z)
- scaleX(x)
- scaleY(y)
- scaleZ(z)
- rotate3d(x,y,z,angle)
- rotateX(angle)
- rotateY(angle)
- rotateZ(angle)
- perspective(n)

For eg.,

```
transform : rotateZ(90deg);
```

Ref: https://www.w3schools.com/css/css3_3dtransforms.asp

CSS Transitions:

CSS transitions allows you to change property values smoothly (from one value to another), over a given duration.

```
<style>
div {
  width: 100px;
  height: 100px;
  background-color: red;
  -webkit-transition: width 2s; /* For Safari 3.1 to 6.0 */
  transition: width 2s, height 4s;
}

div:hover {
  width: 300px;
  height: 300px;
}
</style>

<div></div>
```

Specify the speed curve of the transition:

The `transition-timing-function` property specifies the speed curve of the transition effect.
The transition-timing-function property can have the following values:

- `ease` - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
- `linear` - specifies a transition effect with the same speed from start to end
- `ease-in` - specifies a transition effect with a slow start
- `ease-out` - specifies a transition effect with a slow end
- `ease-in-out` - specifies a transition effect with a slow start and end

- cubic-bezier(n,n,n,n) - lets you define your own values in a cubic-bezier function.

Delay the transition effect :

The transition-delay property specifies a delay (in seconds) for the transition effect.

The following example has a 1 second delay before starting:

```
<style>
div {
  width: 100px;
  height: 100px;
  background: red;
  -webkit-transition: width 2s; /* Safari */
  transition: width 2s;
  transition-delay: 1s;
}

/* For Safari 3.1 to 6.0 */
#div1 {-webkit-transition-timing-function: linear;}
#div2 {-webkit-transition-timing-function: ease;}
#div3 {-webkit-transition-timing-function: ease-in;}
#div4 {-webkit-transition-timing-function: ease-out;}
#div5 {-webkit-transition-timing-function: ease-in-out;}

/* Standard syntax */
#div1 {transition-timing-function: linear;}
#div2 {transition-timing-function: ease;}
#div3 {transition-timing-function: ease-in;}
#div4 {transition-timing-function: ease-out;}
#div5 {transition-timing-function: ease-in-out;}

div:hover {
  width: 300px;
}
</style>
```

```
<div id="div1">linear</div><br>
<div id="div2">ease</div><br>
<div id="div3">ease-in</div><br>
<div id="div4">ease-out</div><br>
<div id="div5">ease-in-out</div><br>
```

Transition + Transform:

```
<style>
div {
  width: 100px;
  height: 100px;
  background: red;
  -webkit-transition: width 2s, height 2s, -webkit-transform 2s; /* Safari */
  transition: width 2s, height 2s, transform 2s;
}

div:hover {
  width: 300px;
  height: 300px;
  -webkit-transform: rotate(180deg); /* Safari */
  transform: rotate(180deg);
}
</style>
<div></div>
```

-

CSS Animation:

CSS animations allows animation of most HTML elements without using JavaScript or Flash!

- When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

- The animation-duration property defines how long time an animation should take to complete. If the animation-duration property is not specified, no animation will occur, because the default value is 0s (0 seconds).
- The animation-delay property specifies a delay for the start of an animation.
- The animation-iteration-count property specifies the number of times an animation should run. The following example will run the animation 3 times before it stops:

```
@keyframes example {
        from {background-color: red;} // 0 %
        to {background-color: yellow;} // 100 %
}
```

Code:

```
<style>
div {
  width: 100px;
  height: 100px;
  background-color: red;
  position: relative;
  -webkit-animation-name: example; /* Safari 4.0 - 8.0 */
  -webkit-animation-duration: 4s; /* Safari 4.0 - 8.0 */
  -webkit-animation-iteration-count: 3; /* Safari 4.0 - 8.0 */
  animation-name: example;
  animation-duration: 4s;
  Animation-delay: 2s;
  animation-iteration-count: 3;
}

/* Safari 4.0 - 8.0 */
@-webkit-keyframes example {
  0%   {background-color:red; left:0px; top:0px;}
  25%  {background-color:yellow; left:200px; top:0px;}
  50%  {background-color:blue; left:200px; top:200px;}
```

```
  75%  {background-color:green; left:0px; top:200px;}
  100% {background-color:red; left:0px; top:0px;}
}

/* Standard syntax */
@keyframes example {
  0%   {background-color:red; left:0px; top:0px;}
  25%  {background-color:yellow; left:200px; top:0px;}
  50%  {background-color:blue; left:200px; top:200px;}
  75%  {background-color:green; left:0px; top:200px;}
  100% {background-color:red; left:0px; top:0px;}
}
</style>
<div> </div>
```

Run Animation in reverse direction:

The `animation-direction` property specifies whether an animation should be played forwards, backwards or in alternate cycles.

The animation-direction property can have the following values:

- `normal` - The animation is played as normal (forwards). This is default
- `reverse` - The animation is played in reverse direction (backwards)
- `alternate` - The animation is played forwards first, then backwards
- `alternate-reverse` - The animation is played backwards first, then forwards

Code:

```
<style>
div {
  width: 100px;
  height: 100px;
  background-color: red;
  position: relative;
```

```
  -webkit-animation-name: example; /* Safari 4.0 - 8.0 */
  -webkit-animation-duration: 4s; /* Safari 4.0 - 8.0 */
  -webkit-animation-direction: reverse; /* Safari 4.0 - 8.0 */
  animation-name: example;
  animation-duration: 4s;
  animation-direction: reverse;
}

/* Safari 4.0 - 8.0 */
@-webkit-keyframes example {
  0%   {background-color:red; left:0px; top:0px;}
  25%  {background-color:yellow; left:200px; top:0px;}
  50%  {background-color:blue; left:200px; top:200px;}
  75%  {background-color:green; left:0px; top:200px;}
  100% {background-color:red; left:0px; top:0px;}
}

/* Standard syntax */
@keyframes example {
  0%   {background-color:red; left:0px; top:0px;}
  25%  {background-color:yellow; left:200px; top:0px;}
  50%  {background-color:blue; left:200px; top:200px;}
  75%  {background-color:green; left:0px; top:200px;}
  100% {background-color:red; left:0px; top:0px;}
}
</style>
```

Specify the speed curve for the animation:

The `animation-timing-function` property specifies the speed curve of the animation.

The animation-timing-function property can have the following values:

- `ease` - Specifies an animation with a slow start, then fast, then end slowly (this is default)
- `linear` - Specifies an animation with the same speed from start to end

- ease-in - Specifies an animation with a slow start
- ease-out - Specifies an animation with a slow end
- ease-in-out - Specifies an animation with a slow start and end
- cubic-bezier(n,n,n,n) - Lets you define your own values in a cubic-bezier function.

Specify the fill-mode for an Animation

CSS animations do not affect an element before the first keyframe is played or after the last keyframe is played. The animation-fill-mode property can override this behavior.
The animation-fill-mode property specifies a style for the target element when the animation is not playing (before it starts, after it ends, or both).
The animation-fill-mode property can have the following values:

- none - Default value. Animation will not apply any styles to the element before or after it is executing
- forwards - The element will retain the style values that is set by the last keyframe (depends on animation-direction and animation-iteration-count)
- backwards - The element will get the style values that is set by the first keyframe (depends on animation-direction), and retain this during the animation-delay period
- both - The animation will follow the rules for both forwards and backwards, extending the animation properties in both directions

The following example lets the element retain the style values from the last keyframe when the animation ends:

```
<style>
div {
  width: 100px;
  height: 100px;
  background: red;
  position: relative;
  -webkit-animation-name: example;  /* Safari 4.0 - 8.0 */
  -webkit-animation-duration: 3s;  /* Safari 4.0 - 8.0 */
  -webkit-animation-fill-mode: forwards; /* Safari 4.0 - 8.0 */
```

```
  animation-name: example;
  animation-duration: 3s;
  animation-fill-mode: forwards;
}

/* Safari 4.0 - 8.0 */
@-webkit-keyframes example {
  from {top: 0px;}
  to {top: 200px; background-color: blue;}
}

@keyframes example {
  from {top: 0px;}
  to {top: 200px; background-color: blue;}
}
</style>

<p>Let the div element retain the style values from the last keyframe when the animation ends:</p>

<div></div>
```

-

CSS Object fit property:

The CSS object-fit property is used to **specify how an <img> or <video> should be resized to fit its container.**

This property tells the content to fill the container in a variety of ways; such as "preserve that aspect ratio" or "stretch up and take up as much space as possible".

Look at the following image from Paris, which is 400x300 pixels:

However, if we style the image above to be 200x400 pixels, it will look like this:

```
img {
      width: 200px;
      height: 400px;
}
```

We see that the image is being squeezed to fit the container of 200x400 pixels, and its original aspect ratio is destroyed.

If we use `object-fit: cover;` it will cut off the sides of the image, preserving the aspect ratio, and also filling in the space, like this:

```
img {
      width: 200px;
      height: 400px;
      object-fit: cover;
}
```

The `object-fit` property can have the following values:

- `fill` - This is default. The replaced content is sized to fill the element's content box. If necessary, the object will be stretched or squished to fit
- `contain` - The replaced content is scaled to maintain its aspect ratio while fitting within the element's content box

- cover - The replaced content is sized to maintain its aspect ratio while filling the element's entire content box. The object will be clipped to fit
- none - The replaced content is not resized
- scale-down - The content is sized as if none or contain were specified (would result in a smaller concrete object size).

Ex 2:

```
<div style="width:100%;height:400px;">

  <img src="rock600x400.jpg" alt="Norway" style="float:left;width:50%;height:100%;object-fit:cover;">

  <img src="paris.jpg" alt="Paris" style="float:left;width:50%;height:100%;object-fit:cover;">

</div>
```

-

```
margin-top: 100px;
margin: 0 auto;  // vertical margin - 0 px, horizontal margin - auto
margin: 25px 50px 75px 100px; (top, right, bottom, left)
margin: 25px 50px 75px; (top, right & left, bottom)
margin: 25px 50px; (top & bottom, left & right)
margin: auto / inherit;
```

Note: Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins
.
This does not happen on left and right margins! Only top and bottom margins

```
padding-top: 100px;
padding: 25px 50px 75px 100px; (top, right, bottom, left)
padding: 25px 50px 75px; (top, right & left, bottom)
padding: 25px 50px; (top & bottom, left & right)
```

```
width: 100 px / 20% ;
max-width: 100px;
```

```
text-align: left;
text-decoration: none;
text-transform: uppercase;
text-indent: 50px;
text-shadow: 3px 2px red;
```

```
letter-spacing: 3px;
line-height: 0.8;
word-spacing: 10px;
direction: rtl;
vertical-align: bottom;
border-radius :25px;
opacity: 0.5;
```

Note : If you do not want to apply opacity to child elements, like in our example above, use **RGBA** color values

---
```
font-family: "Times New Roman"
font-family: "Garamond", "Times", serif;
font-style: italic;
font-size: 40px;
font-weight: bold / 100;
font-variant: small-caps;
```

Note:

The font-weight property can also be assigned a number value to style text on a numeric scale ranging from 100 to 900. Valid values are multiples of 100 within this range such as 200 or 500. When using numeric weights, there are a number of default font weights that we can use:

1. 400 is the default font-weight of most text.
2. 700 signifies a bold font-weight.
3. 300 signifies a light font-weight.

```
/* This is a single-line comment */

/* This is
a multi-line
comment */
```

```
a:link
a:visited
a:hover
a:active
```

---
list-style-type: circle;
list-style-image: url('sqpurple.gif');
list-style-position: inside;
---
tr:hover {background-color: #f5f5f5;}

tr:nth-child(even) {background-color: #f2f2f2;}

---
position: static / relative / absolute / fixed / sticky ;

left:-1;
z-index: -1;
overflow: visible / hidden /scroll / auto;
overflow-x: visible / hidden /scroll / auto;
float: left / right / none / inherit;
display: inline-block / inline / block / none ;
visibility: hidden;
clear: left / right / both / none / inherit;
---
background: linear-gradient(to right, red , yellow);
background: linear-gradient(to bottom right, red, yellow); // diagonal
background: linear-gradient(-90deg, red, yellow); // using angles
background: linear-gradient(to right, red,orange,yellow,green,blue,indigo,violet); // multiple colors
---
background: radial-gradient(red 5%, yellow 15%, green 60%);
background: radial-gradient(circle, red, yellow, green);
---
text-shadow: 2px 2px; // 2px horizontal shadow, 2px vertical shadow.
text-shadow: 2px 2px red;
text-shadow: 2px 3px 4px red; // 2px horizontal shadow , 3px vertical shadow, 4px shadow ;
---
box-shadow: 2px 5px; // 2px horizontal shadow, 5px vertical shadow
box-shadow: 2px 6px grey;
box-shadow: 10px 11px 5px grey; // 10px horizontal, 11px vertical, 5px shadow effect.
box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
---
text-overflow: clip / ellipsis / inherit;
word-wrap: normal / break-word;
word-break: keep-all / break-all;

2D transform:
transform: translate(50px, 100px);
transform: rotate(20deg);
transform: scale(2, 3);
transform: skewX(20deg);

transform: skewY(20deg);
transform: skew(20deg,10deg);

3D transform:
transform: rotateZ(90deg);
---
transition: width 2s, height 4s;

transition-property: width;
transition-duration: 2s;
transition-timing-function: linear;
transition-delay: 1s;
(or)
transition: width 2s linear 1s;
---
animation-name: example;

animation-duration: 5s;

animation-timing-function: linear;

animation-delay: 2s;

animation-iteration-count: infinite;

animation-direction: alternate;

(or)

animation: example 5s linear 2s infinite alternate;

animation-fill-mode: forwards;

---

filter: blur(1px) / brightness(100%) / contrast(50%) / sepia(20%) / invert(50%) / hue-rotate(20%) / saturate(7) ;
object-fit: cover / fill / contain / none / scale-down ;

-

FlexBox:

https://docs.google.com/document/d/1WSNmjQdLj3q8czBevdK0j0P-gjbIfMG8ESy-w3kMfBY/edit

---
-

Grid

https://docs.google.com/document/d/1uBAbC8URL988XTlAxp0gMhu_QBEX3MBzDQWrTD97SlY/edit#

Css Gradients:

To create a linear gradient you must define at least two color stops. Color stops are the colors you want to render smooth transitions among. You can also set a starting point and a direction (or an angle) along with the gradient effect.

Linear Gradient - from left to right:

The following example shows a linear gradient that starts from the left. It starts red, transitioning to yellow:

```
background: linear-gradient(to right, red , yellow);

background: linear-gradient(to bottom right, red, yellow); // diagonal

background: linear-gradient(-90deg, red, yellow); // using angles

background: linear-gradient(to right, red,orange,yellow,green,blue,indigo,violet); // multiple colors
```

Radial Gradient:

```
background: radial-gradient(red 5%, yellow 15%, green 60%);

background: radial-gradient(circle, red, yellow, green);
```

Ref : https://www.w3schools.com/css/css3_gradients.asp

Shadow:

```
text-shadow: 2px 2px; // 2px horizontal shadow, 2px vertical shadow.

text-shadow: 2px 2px red;

text-shadow: 2px 3px 4px red; // 2px horizontal shadow , 3px vertical shadow, 4px shadow ;
```

Border around text:

```
h1 {
    color: red;
    text-shadow: -1px 0 black, 0 1px black, 1px 0 black, 0 -1px black;
}
```

Box-shadow:

```
box-shadow: 2px 5px; // 2px horizontal shadow , 5px vertical shadow;
box-shadow: 2px 6px grey;
box-shadow: 10px 11px 5px grey; // 10px horizontal, 11px vertical, 5px shadow effect.
box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
```

Styling the images:

Thumbnail images:

```
<style>
img {
    border: 1px solid #ddd;
    border-radius: 4px;
    padding: 5px;
    width: 150px;
}

img:hover {
    box-shadow: 0 0 2px 1px rgba(0, 140, 186, 0.5);
}
</style>
<a target="_blank" href="paris.jpg">
  <img src="paris.jpg" alt="Paris" style="width:150px">
</a>
```

Responsive images:

```
<style>
img {
  max-width: 100%;
  height: auto;
}
</style>
<img src="trolltunga.jpg" alt="Norway" width="1000" height="300">
```

Center an image:

```
<style>
img {
  display: block;
  margin: auto;
}
</style>
<img src="paris.jpg" alt="Paris" style="width:50%">
```

Cards:

```
<style>
body {margin:25px;}

div.polaroid {
  width: 80%;
  background-color: white;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
  margin-bottom: 25px;
}

div.container {
  text-align: center;
  padding: 10px 20px;
}
</style>
```

```
<div class="polaroid">
  <img src="rock600x400.jpg" alt="Norway" style="width:100%">
  <div class="container">
    <p>The Troll's tongue in Hardanger, Norway</p>
  </div>
</div>
```

Transparent images:

```
img {
    opacity: 0.5;
}
<img src="img_forest.jpg" alt="Forest" width="170" height="100">
```

Image text:

```
<style>
.container {
    position: relative;
}

.topright {
    position: absolute;
    top: 8px;
    right: 16px;
    font-size: 18px;
}

img {
    width: 100%;
    height: auto;
    opacity: 0.3;
}
</style>

<div class="container">
```

```
  <img src="trolltunga.jpg" alt="Norway" width="1000" height="300">
  <div class="topright">Top Right</div>
</div>
```

Image filters:

The CSS filter property adds visual effects (like blur and saturation) to an element.

```
<style>
img {
    width: 33%;
    height: auto;
    float: left;
    max-width: 235px;
}

.blur {-webkit-filter: blur(8px);filter: blur(1px);}
.brightness {-webkit-filter: brightness(150%);filter: brightness(100%);}
.contrast {-webkit-filter: contrast(180%);filter: contrast(180%);}
.grayscale {-webkit-filter: grayscale(100%);filter: grayscale(100%);}
.huerotate {-webkit-filter: hue-rotate(180deg);filter: hue-rotate(180deg);}
.invert {-webkit-filter: invert(100%);filter: invert(100%);}
.opacity {-webkit-filter: opacity(50%);filter: opacity(50%);}
.saturate {-webkit-filter: saturate(7); filter: saturate(7);}
.sepia {-webkit-filter: sepia(100%);filter: sepia(100%);}
.shadow {-webkit-filter: drop-shadow(8px 8px 10px green);filter: drop-shadow(8px 8px 10px
green);}
</style>

<img src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="blur" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="brightness" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="contrast" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="grayscale" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
```

```
<img class="huerotate" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="invert" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="opacity" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="saturate" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="sepia" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
<img class="shadow" src="pineapple.jpg" alt="Pineapple" width="300" height="300">
```

Flip an image:

```
img:hover {
    -webkit-transform: scaleX(-1);
    transform: scaleX(-1);
}
<img src="paris.jpg" alt="Paris" width="400" height="300">
```

Image hover overlay:

Ref: https://www.w3schools.com/css/css3_images.asp

Responsive Image Gallery:

```
<style>
div.gallery {
    border: 1px solid #ccc;
}

div.gallery:hover {
    border: 1px solid #777;
}

div.gallery img {
    width: 100%;
    height: auto;
}

div.desc {
    padding: 15px;
    text-align: center;
}

* {
    box-sizing: border-box;
```

```
}

.responsive {
    padding: 0 6px;
    float: left;
    width: 24.99999%;
}

@media only screen and (max-width: 700px){
    .responsive {
        width: 49.99999%;
        margin: 6px 0;
    }
}

@media only screen and (max-width: 500px){
    .responsive {
        width: 100%;
    }
}

.clearfix:after {
    content: "";
    display: table;
    clear: both;
}
</style>


<div class="responsive">
  <div class="gallery">
    <a target="_blank" href="img_fjords.jpg">
      <img src="img_fjords.jpg" alt="Trolltunga Norway" width="300" height="200">
    </a>
    <div class="desc">Add a description of the image here</div>
  </div>
</div>
```

Basic button styling:

```
<style>
.button {
    background-color: #4CAF50;
```

```
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
}
</style>

<button>Default Button</button>
<a href="#" class="button">Link Button</a>
<button class="button">Button</button>
<input type="button" class="button" value="Input Button">
```

Disabled button:

```
.disabled {
    opacity: 0.6;
    cursor: not-allowed;
}
<button class="disabled">Disabled Button</button>
```

Ref : https://www.w3schools.com/css/css3_buttons.asp

Simple pagination:

If you have a website with lots of pages, you may wish to add some sort of pagination to each page:

Code:

```
<style>
.center {
        text-align:center;
}
.pagination {
    display: inline-block;
}
```

```
.pagination a {
    color: black;
    float: left;
    padding: 8px 16px;
    text-decoration: none;
    transition: background-color .8s;
}

.pagination a.active {
    background-color: green;
    color: white;
}

.pagination a:hover:not(.active) {background-color: grey;}
</style>
<div class=”center”>
          <div class="pagination">
            <a href="#">&laquo;</a>
            <a href="#">1</a>
            <a href="#" class="active">2</a>
            <a href="#">3</a>
            <a href="#">4</a>
            <a href="#">5</a>
            <a href="#">6</a>
            <a href="#">&raquo;</a>
          </div>
</div>
```
