**HTML**: HTML stands for hypertext markup language, it works by using something called markup tags. These tags describe the content that is displayed in the browser window, this content can be things like headings, paragraphs, images and even multimedia elements such as audio and video, the way html describes the content is known as markup

**CSS**: CSS is short for cascading style sheets and adds visual enhancements like colors and layout to the web page, this is commonly known as styling. It works by enhancing the HTML elements and telling them how to display. Have you ever noticed a friend changed their color or style of their hair? Well, your friend's hair is like the HTML and the color and style is CSS.

**Javascript**: JavaScript is the powerhouse of a web page. It has the ability to manipulate the content that you see on the screen as you interact with it.

**Hoisting:**

Web Hoisting:Web hosting is a service where you place your website and files on the hosting companies web server.

You're essentially renting the space in return for stable and secure storage. There are 4 types of web hoisting:

1. Shared hoisting

2. Virtual private hoisting

3. Dedicated hoisting

4. Cloud hoisting

**Status Code:**

There are five groups of status codes. They're grouped by the first digit of the error number. Informational is grouped from 100-199. Successful responses are grouped from 200-299. Redirection message are from 300-399. Client error responses ranged from 400-499 and server error responses are from 500- 599.

**Difference between website and web application**:

The key difference between a website, and web application is the level of interactivity, and dynamic content. The easy way to remember this is that a website is more informative and a web application is more interactive. Think of ordering food online. Let's say you would like to order some food and you go to your favorite site, the browser then displays a web page, you select some food from the menu and submit your order. Compare this to for example, a company website that displays information about themselves and the services they provide. In the food ordering example, the content displayed is specific to your user account and location, the web application displays content based on the user's input and interaction. Whereas with the company website, the user simply views the content and this content is the same for everyone who visits the website.

**HTML**:

The first version of HTML was released in 1991 along with the first web browser and web server. HTML stands for Hypertext Markup Language. But what does that mean exactly? Hypertext is text which contains links to other text. Markup refers to tags and elements used within a document.

**1990**: Tim Berner’s Lee

**1993**: Internet engineering Task Force(IETF)

**1996**: World Wide Web Consortium (W3C)

**2000**: International Organisation for Standards(IOS) and International Electrotechnical commission (IEC)

**2006**: Web Hypertext Application Technology Working Group (WHATWG)

**Package manager:**

A package manager is a tool that automatically downloads and installs dependencies. We also refer to dependencies as packages. A package manager also provides the capability to publish your own packages.

**JavaScript bundles** **(Gulp and Webpack)**:

Once all your dependencies are downloaded, you need to include them in your HTML file. But adding all of them into an HTML file would take forever. This is where you will use a bundling tool. The purpose of a bundler is to automatically combine them into a single file. If you're bundle is significantly large, many bundles can split your dependencies into multiple bundles.

**SPA and Traditional websites:**

**How traditional website works:** Before the advent of modern JavaScript frameworks, most websites were implemented as multi-page applications. But this makes traditional applications resource intensive to web servers because sending entire web pages for every request consumes excessive bandwidth and uses CPU time to generate dynamic pages. If your website is complex, the site browsing experience may appear slow to users. It will be even slower if they have a poor or limited internet connection. To solve this problem, many web developers build their web applications as SPAs.

**SPA:** It's called single-page, but that doesn't mean your website has only one page of content. What it means is that there is only one HTML page that gets sent from the server to the browser, but that page will update its content as your users interact with your website. A SPA allows the user to interact with the website without the application needing to download entire new web pages. Instead, it rewrites the current web page as the user interacts with it. The result is a browsing experience that feels faster and more responsive to user input.

**REACT:**

React is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time.

When your component is first initialized, the render method is called, generating a lightweight representation of your view. From that representation, a string of markup is produced and injected into the document. When your data changes, the render method is called again. In order to perform updates as efficiently as possible, we diff the return value from the previous call to render with the new one and generate a minimal set of changes to be applied to the DOM. The data returned from render is neither a string nor a DOM node — it’s a lightweight description of what the DOM should look like. We call this process **reconciliation**.

**Javascript(concepts and ideas in functional programming):**

Here are some of the most important ones:

* First-class functions
* Higher-order function
* Pure functions and side-effects

### **First-class functions:**

A function in JavaScript is just a value - from this vantage point, almost no different then a string or a number. It means that a function in JavaScript is just another value that we can:

* pass to other functions
* save in a variable
* return from other functions

**Higher-order functions**

A higher-order function is a function that has either one or both of the following characteristics:

* It accepts other functions as arguments
* It returns functions when invoked

### **Pure functions and side-effects**

Another concept of functional programming are pure functions. A pure function returns the exact same result as long as it's given the same values. Another rule for a function to be considered pure is that it should not have side-effects. A side-effect is any instance where a function makes a change outside of itself.

This includes:

* changing variable values outside of the function itself, or even relying on outside variables
* calling a Browser API (even the console itself!)
* calling Math.random() - since the value cannot be reliably repeated

**var, let and const:**

**var**: console.log(user); var user;: The above code will not throw an error and return undefined . It means we can use user even before initialization.

**let** : console.log(user); var user;: The above code will throw an error as Cannot access user before initialization. It means we can not use user before initialization.

**const**: same as let but we cannot reassign the value to const.

**Destucturing**:

it's possible to extract the properties from objects into distinct variables using destructuring. It means destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

var car={speed:400, color:black}

**Object.keys()** return keys of object in an array. For above object, Object.keys(car); will return ['speed','color'];

**Object.values()** return values of object in an array. For above object, Object.values(car); will return [400, 'black]

**Object.entries()** returns an array listing both the keys and the values. Object.entries(car); will return [ ['speed', 400], ['color', black] ];

**Template Literal:**

Template literals are an alternative way of working with strings, which was introduced in the ES6 addition to the JavaScript language. The above code snippet is an example of a template string, which is also known as a template literal. `Hello, World!` (Backtick syntax). Essentially, using template literals allows programmers to embed variables directly in between the backticks, without the need to use the **+** operator and the single or double quotes to delimit string literals from variables

Usage of Template Literal:

let greet = "Hello";

let place = "World"; console.log(`${greet} ${place} !`)

This opens up a host of possibilities. For example, it's possible to evaluate a ternary expression inside a template literal.

**Learn about For of and for in loop;**

**Spread Operator:**

It is the shortest and simplest method to copy the properties of an object onto a newly created object. Think of the spread operator as a magical multi-purpose tool that can spread out array items and join objects together. Note that the spread operator is characterized by three dots. We know, that a spread operator in JavaScript, is used to unpack a box, for example, to unpack an array.

**Rest Operator:**

The rest operator, on the other hand, is used to build a smaller box, and pack items into it. The rest operator allows you to take items from an array and use them to create a separate sub-array. It's important to know, that the rest parameter, must be the last parameter in the function definition.

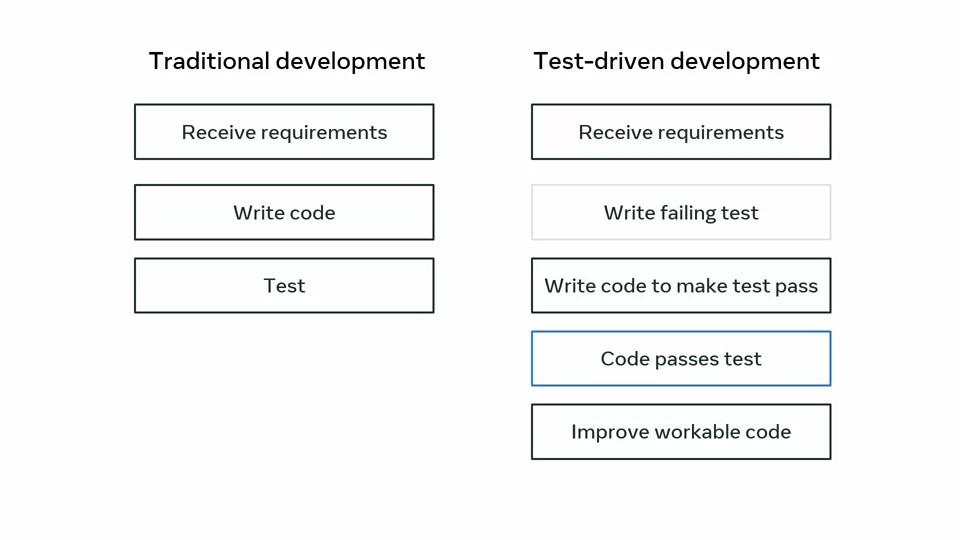
**TESTING**

Refactoring: Updating(such as Optimizing ) the code without changing or affecting the functionality

Mocking: Separate code from related dependencies during testing. In other words , we can use mocking feature to make sure that Unit testing is standalone. Mocking allows you to separate the code that you are testing from its related dependencies.

Jest: Jest run the tests for code in your current project to verify the expected output.

TDD(Test Driven development):  TDD for short is a streamlined process of writing code that will satisfy some requirements.



**Version Control:** A system that records all the changes and modification to files for tracking purposes.

**Git:** Git is a version control system designed to help users keep track of changes to files within their projects. Git was also designed to tackle some of the shortcomings of other version control systems. The benefits that Git offers over similar systems include, better speed and performance, reliability, free and open source axis, and an accessible syntax

**GitHub:** GitHub is a Cloud-based hosting service that lets you manage Git repositories from a user interface. A Git repository is used to track all changes to files in a specific folder, and keep a history of all those changes.

**GIT Workflow:** Git uses workflows which can be broken into three states namely, modified, staged, and committed.

Let's start with the first state, adding removing and updating any file inside the repository is considered a modified state. Git knows the file has changed, but does not track it. This is where the staging state comes in. Let's turn to it now. In order for Git to track a file, it needs to be put in the staged area. Once added, any modifications are tracked. Which offers a security blanket prior to committing the changes. Then, the last state is the committed state. Committing a file in Git is like a save point in many ways. Git will save the file, and have a snapshot of the current changes.

To keep track of file: git add filename

To unstage file : git restore –stage filename

To commit the file: git commit -m “message”

Create new branch: git checkout -b branch name(create branch as well move user from main branch to created branch) or git branch branchname(just create branch)

Pull Request: The purpose of a pull request is to obtain a peer review of changes made to the branch. In other words, to validate that the changes are correct.

Token: ghp\_H6brKn0Q23gg3SVM8EdxbhmpWUO4N92aczvJ

# **Semantic HTML cheat sheet**

There are hundreds of semantic tags available to help describe the meaning of your HTML documents. Below is a cheat sheet with some of the most common ones you’ll use in this course and in your development career.

## **Sectioning tags**

Use the following tags to organize your HTML document into structured sections.

**<header>** The header of a content section or the web page. The web page header often contains the website branding or logo.

**<nav>** The navigation links of a section or the web page.

**<footer>** The footer of a content section or the web page. On a web page, it often contains secondary links, the copyright notice, privacy policy and cookie policy links.

**<main>** Specifies the main content of a section or the web page.

**<aside>** A secondary set of content that is not required to understand the main content.

**<article>** An independent, self-contained block of content such as a blog post or product.

**<section>** A standalone section of the document that is often used within the body and article elements.

**<details>** A collapsed section of content that can be expanded if the user wishes to view it.

**<summary>** Specifies the summary or caption of a **<details>** element.

**<h1><h2><h3><h4><h5><h6>** Headings on the web page. **<h1>** indicates the most important heading whereas **<h6>** indicates the least important.

## Content tags

**<blockquote>** Used to describe a quotation.

**<dd>** Used to define a description for the preceding **<dt>** element.

**<dl>** Used to define a description list. **<dt>** Used to describe terms inside **<dl>** elements.

**<figcaption>** Defines a caption for a photo image.

**<figure>** Applies markup to a photo image.

**<hr>** Adds a horizontal line to the parent element.

**<li>** Used to define an item within a list. **<menu>** A semantic alternative to **<ul>** tag.

**<ol>** Defines an ordered list.

**<p>** Defines a paragraph.

**<pre>** Used to represent preformatted text. Typically rendered in the web browser using a monospace font.

**<ul>** Unordered list

## Inline tags

**<a>** An anchor link to another HTML document. **<abbr>** Specifies that the containing text is an abbreviation or acronym. **<b>** Bolds the containing text. When used to indicate importance use **<strong>** instead. **<br>** A line break. Moves the subsequent text to a new line. **<cite>** Defines the title of creative work (for example a book, poem, song, movie, painting or sculpture). The text in the **<cite>** element is usually rendered in italics. **<code>** Indicates that the containing text is a block of computer code. **<data>** Indicates machine-readable data. **<em>** Emphasizes the containing text. **<i>** The containing text is displayed in italics. Used to indicate idiomatic text or technical terms. **<mark>** The containing text should be marked or highlighted. **<q>** The containing text is a short quotation. **<s>** Displays the containing text with a strikethrough or line through it. **<samp>** The containing text represents a sample. **<small>** Used to represent small text, such as copyright and legal text. **<span>** A generic element for grouping content for CSS styling. **<strong>** Displays the containing text in bold. Used to indicate importance. **<sub>** The containing text is subscript text, displayed with a lowered baseline. **<sup>** The containing text is superscript text, displayed with a raised baseline. **<time>** A semantic tag used to display both dates and times. **<u>** Displays the containing text with a solid underline. **<var>** The containing text is a variable in a mathematical expression.

## Embedded content and media tags

**<audio>** Used to embed audio in web pages. **<canvas>** Used to render 2D and 3D graphics on web pages. **<embed>** Used as a containing element for external content provided by an external application such as a media player or plug-in application. **<iframe>** Used to embed a nested web page. **<img>** Embeds an image on a web page. **<object>** Similar to **<embed>** but the content is provided by a web browser plug-in. **<picture>** An element that contains one **<img>** element and one or more **<source>** elements to offer alternative images for different displays/devices. **<video>** Embeds a video on a web page. **<source>** Specifies media resources for **<picture>**, **<audio>** and**<video>** elements. **<svg>** Used to define Scalable Vector Graphics within a web page.

## Table tags

**<table>** Defines a table element to display table data within a web page. **<thead>** Represents the header content of a table. Typically contains one **<tr>** element. **<tbody>** Represents the main content of a table. Contains one or more **<tr>**elements. **<tfoot>** Represents the footer content of a table. Typically contains one **<tr>** element. **<tr>** Represents a row in a table. Contains one or more **<td>** elements when used within **<tbody>** or **<tfoot>**. When used within **<thead>**, contains one or more **<th>** elements. **<td>** Represents a cell in a table. Contains the text content of the cell. **<th>** Defines a header cell of a table. Contains the text content of the header. **<caption>** Defines the caption of a table element. **<colgroup>** Defines a semantic group of one or more columns in a table for formatting. **<col>** Defines a semantic column in a table.

Basic meta tags (meta tags For SEO)

**<meta name="description"/>** provides a brief description of the web page **<meta name=”title”/>** specifies the title of the web page **<meta name="author" content="name">** specifies the author of the web page **<meta name="language" content="english">** specifies the language of the web page **<meta name="robots" content="index,follow" />** tells search engines how to crawl or index a certain page **<meta name="google"/>** tells Google not to show the sitelinks search box for your page when showing search results **<meta name="googlebot" content=”notranslate” />** tells Google you don’t want to provide an automatic translation for your page if the user uses a different language **<meta name="revised" content="Sunday, July 18th, 2010, 5:15 pm" />** specifies the last modified date and time on which you have made certain changes **<meta name="rating" content="safe for kids">** specifies the expected audience for your page **<meta name="copyright" content="Copyright 2022">** specifies a Copyright

## <meta http-equiv="..."/> tags

**<meta http-equiv="content-type" content="text/html">** specifies the format of the document returned by the server **<meta http-equiv="default-style"/>** specifies the format of the styling document **<meta http-equiv="refresh"/>** specifies the duration of the page before it’s considered stale **<meta http-equiv=”Content-language”/>** specifies the language of the page **<meta http-equiv="Cache-Control" content="no-cache">** instructs the browser how to cache your page

## Responsive design/mobile meta tags

**<meta name="format-detection" content="telephone=yes"/>** indicates that telephone numbers should appear as hypertext links that can be clicked to make a phone call **<meta name="HandheldFriendly" content="true"/>** specifies that the page can be properly visualized on mobile devices **<meta name="viewport" content="width=device-width, initial-scale=1.0"/>** specifies the area of the window in which web content can be seen

CSS FLEXBOX:

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/#aa-basics-and-terminology>

| **Unit** | **Description and relativity** |
| --- | --- |
| **em** | Font size of the parent where present. |
| **ex** | x-co-ordinate or height of the font element. |
| **ch** | Width of the font character. |
| **rem** | Font size of the root element. |
| **lh** | Value computed for line height of parent element. |
| **rlh** | Value computed for line height of root element which is <html>. |
| **vw** | 1% of the viewport width. |
| **vh** | 1% of the viewport height. |
| **vmin** | 1% of the smaller dimension of viewport. |
| **vmax** | 1% of the larger dimension of viewport. |
| **%** | Denotes a percentage value in relation to its parent element. |

FlexBox:

Flex grow, which specifies how much the item will grow compared to other flexible items, flex shrink, which specifies how much the item will shrink compared to other flexible items. And flex basis, which specifies the initial length of the flexible item.

GRID:

Grid shorthand consists of the following properties with default values:

**grid**

A grid will allow you organize the various elements on your page.

**grid-template-rows: none**

This feature allows you configure your elements so that they are organized similarly to rows on a table.

**grid-template-columns: none**

This feature allows you configure your elements but with this setting the elements are organized like columns on a table.

**grid-template-areas: none**

This feature allows you configure the names of a grid and how they sit in relation to one another.

**grid-auto-rows: auto**

Default setting for all row sizes that have not been explicitly configured.

**grid-auto-columns: auto**

Default setting for all column sizes that have not been explicitly configured.

**grid-auto-flow: row**

Default location for rows that are not explicitly allocated.

**column-gap: normal**

This sets the gap between the columns

**row-gap: normal**

This sets the gap between the rows

**Grid properties for container**

**grid-template-columns: measurement units | % units |repeat()**

Defines the line names, and maintains a constant size of column items. Can accept a range of different measurement sizes.

**grid-template-rows: measurement units | % units |repeat()**

Defines the line names, and maintains a constant size of rows. Can accept a range of different measurement sizes.

**grid-auto-columns: measurement unit (fixed value for all columns)**

Determines the default size for columns that have not been explicitly configured.

**grid-auto-rows: measurement unit (fixed value for all rows)**

Determines the default size for rows that have not been explicitly configured.

**grid-template: “header header” auto**

This allows you define and maintain named cells on a grid

**“main right” 75vh**

This defines two cells named main and right, that have a sizing of 75% of the viewport height.

**“footer footer” 20rem**

This defines two cells named footer and footer, that have a sizing of 20 root em (rem). This defines the size in relation to the html font size.

**Gap**

**grid-gap: measurement units**

Determines the gap between rows and columns

**grid-column-gap: measurement units**

Determines the gap between columns

**grid-row-gap: m-unit-1 m-unit-2**

Determines the gap between columns

**Alignment**

**justify-items: start | center | end | stretch**

Defines the default space that is allot to each item on the grid

**align-items: start | center | end | stretch**

Defines the default space related to an item along the grid’s block axis

**place-items: start | stretch /\* shorthand for two properties above \*/**

This feature allows you align items with the block and inline directions.

**Justification**

**justify-content: start | center | end | stretch | space-between | space-evenly | space-around**

Defines browser allocation of space to content items in relation to the main-axis

**align-content: start | center | end | stretch | space-between | space-evenly | space-around**

Defines browser allocation of space to content items in relation to cross axis and block axis

**place-content: center | start**

This feature allows you align items with the block and inline directions.

**Positioning**

**grid-auto-flow: row | column | dense**

This relates to how the items are placed automatically within the grid

**grid-auto-columns: measurement units**

This relates to the size for columns created without specific size specifications

**grid-auto-rows: measurement units**

This relates to the size for rows created without specific size specifications

**Grid properties for items (child)**

**grid-column: column position /\* E.g. 1/2 \*/**

Allows for specifying where on the grid the column is to start.

**grid-column-start: column start position**

This property determines the starting column position an item is placed on a grid.

**grid-column-end: column end position**

This property determines the end column position an item is placed on a grid.

**grid-row: row position /\* E.g. 1/2 \*/**

Allows for specifying where on the grid the row is to start.

**grid-row-start: row start position**

This property determines the starting row position an item is placed on a grid.

**grid-row-end: row end position**

This property determines the end row position an item is placed on a grid.

**Justification and alignment**

**justify-self: start | center | end | stretch**

Determines how an item is positioned inside its aligned container in relation to the appropriate axis.

**align-self: start | center | end | stretch**

Aligns an item within a grid area.

**place-self: start | stretch /\* shorthand for two properties above \*/**

This setting lets one align and justify an item within a block.

**Flexbox**

The syntax for creating a flexbox:

1

2

3

4

selector{

    display: flex | inline-flex

}





Here the selector can refer to any of the following flex attributes

* Attribute selector
* Class Selector
* ID Selector
* Type Selectors
* Universal Selectors

The display relates to how you want the selector to be shown. Setting display to flex makes the given selector a flex box. Setting display to **inline-flex** makes the selector a flex box container while will be inline.

**Properties for flexbox container**

**flex-direction: row | row-reverse | column | column-reverse**

It is possible to specify the direction your elements will follow. Traditionally text goes from left to right which is flex’s default setting however it can be set from right to left or even top to bottom. The four flex-direction are:

* row : organized from left to right
* row-reverse: organized from right to left
* column: organized from top to bottom
* column-reverse: organized from bottom to top.

**flex-wrap: wrap | nowrap**

The standard layout is to plot the elements from left to right in a straight line. The wrap feature allows you customize this to match the size of the window displaying the page.

* wrap: Automatically wrap the items with as the window space gets smaller.
* Nowrap: Default setting, items remain rigid and don’t respond to adjustments made to the window size.

**align-items: flex-start | flex-end | center |Stretch**

This determines how the flex items are to be positioned on the page. Items can be aligned in a variety of ways

* Flex-start: Similar to standard writing, items start at the top left-hand corner and are positioned from left to right
* Flex-end: Position begins in the bottom right hand corner.
* Center: Item is positioned from the center.
* Stretch: item expands to fill the container.

**justify-content: flex-start | flex-end | center | space-between | space-evenly**

Justify-content determines the alignment of the flex items.

* Flex-start: goes from right to left along the main axis.
* Flex-end: goes from left to right along the main axis.
* Center: Starting at the middle, alignments expands from there.
* Space-between: first and last item are flush with the left and right wall respectively, every other item is evenly spaced.
* Space-evenly: each item is equidistant from each other and the boundary wall

**Properties for flexbox items (child)**

**flex-grow: factor of flex’s main size**

This attribute enables the flex container to grow proportionally to the other containers present.

**flex-shrink: factor of flex’s main size**

This allows elements to shrink in relation to items around it.

**flex-basis: auto | factor of main’s size | measurement unit**

The sets the initial main size of an item. It can be overridden if other stylized elements are configured.

**order:position in flex /\* Set ascending by default \*/**

The standard positioning of items is by source order, however this feature will enable you to configure where the items appear on the page.

**align-self: start | center | end | stretch**

This determines where on the page the child items will be positioned. Similar to the main flex attributes, start is to the left and end is to the right.

**Text effects cheat sheet**

The effects developers use on text items on a web page are chosen mainly because of their styling and layout style. Interesting effects can be created by combining these with other CSS properties.

The visual representation of text content can be changed by four main properties: text-transform, font-style, font-weight and text-decoration.

| **Property** | **Values** | **Description** |
| --- | --- | --- |
| Text-transform | None, uppercase, lowercase, capitalize, full-width | Modify text properties |
| Font-style | Normal, italic, oblique | Font styling options such as italics |
| Font-weight | Normal, weight, lighter, bolder, 100-900 | Other font styling options like change of emphasis such as making text bold |
| Text-decoration | None, underline, overline, line-through | Shorthand for auxiliary elements added to text using other properties such as text-decoration-line |

The additional properties that help configure styling effects are below.

| **Text-align** | **For horizontal alignment of text** |
| --- | --- |
| Text-align-last | Alignment for the last line when text set to justify |
| Text-combine-upright | Multiple characters into the space of a single character placed upright like in Mandarin |
| Text-decoration-color | Color configuration of the text-decoration |
| Text-decoration-line | Line type in text-decoration such as underline, overline and so on |
| Text-decoration-style | Styles added to lines under text such as wavy, dotted and so on |
| Text-decoration-thickness | Thickness of the decoration line |
| Text-emphasis | Shorthand for other properties such as color and style |
| Text-indent | The indentation of the first line |
| Text-justify | Specifies the justification method used when text-align is "justify" |
| Text-orientation | Orientation of text in a line such as sideways, upright and so on |
| Text-shadow | Adds shadow to text |
| Text-underline-position | Declare position of underline set using the text-decoration property |

Other than these, there are some more properties that help modify the alignment and define the scope of text with their containers.

| **Property** | **Values** | **Description** |
| --- | --- | --- |
| Text-overflow | Clip, ellipsis | Determines overflow behavior of text with the container |
| Word-wrap | Normal, anywhere, break-word | Applies to inline elements, alias for overflow-wrap |
| Word-break | Normal, break-all, keep-all, break-word | Used for long words to decide if words should break or overflow |
| Writing-mode | Horizontal-tb, vertical-lr, vertical-rl | Can set the text direction vertical or horizontal |

**Animation and effects cheat sheet**

Transform property: Syntax: transform: transform function-values

Example

.sample-class {

transform: rotate(60deg);}

Keyword-value type: none

.sample-class {

transform: none;

}

Function-value type: matrix()

Variations: matrix(), matrix3d()

.sample-class {

transform: matrix(1.0, 2.0, 3.0, 4.0, 5.0, 6.0);

}

Function-value type: rotate(deg)

Variations: rotate(), rotate3d(), rotateX(), rotate(), rotateZ()

.sample-class {

transform: rotate3d(3,2,1, 100deg);

}

Note: In rotate3d(), the respective values represent x, y, z co-ordinate and degree of rotations

Function-value type: translate(x,y)

Variations: translate(), translate3d(), translateX(), translateY(), translateZ()

.sample-class {

transform: translate3d(10px, 20px, 30px);

}

Note: In translate3d(), the respective values represent translation along the x, y, z co-ordinates

Function-value type: scale(factor)

Variations: scale(), scale3d(), scaleX(), scaleY(), scaleZ()

.sample-class {

transform: scale3d(2, 1, 0.3);

}

Note: In scale3d(), the respective values represent scaling times along the x, y, z co-ordinates

Function-value type: skew(deg, deg)

Variations: skew(), skewX(), skewY()

.sample-class {

transform: skew(100deg);

}

Global value types:

.sample-class {

transform: inherit;

}

.sample-class {

transform: initial;

}

.sample-class {

transform: revert;

}

.sample-class {

transform: revert-layer;

}

.sample-class {

transform: unset;

}

Multiple transform over the same element

Syntax

Transform can be applied for rotate(), scale() and translate() that can be listed together. Each of these properties can have their own values and the actions will give a combined effect.

Example

.sample-class {

transform: rotate(45deg) scale(1.5) translate(45px);

}

Additional property under transform:transform-origin

Determines the anchor point for the centering of transform.

Example .sample-class {

transform-origin: 10px 10px;

}

.sample-class {

transform-origin: right bottom;

}

Transition property

Transition shorthand

Transition shorthand has four following sub-properties, each of which can also be individually defined.

transition-property

transition-duration

transition-timing-function

transition-delay

You have to list the values without naming them individually. Values skipped will be assigned their default values.

Syntax

transition: property duration timing-function delay;

Example

transition: margin-left2s ease-in-out 0.5s;

Animations and @keyframes

animation property:

Syntax

animation: name duration timing-function delay iteration-count direction fill-mode play-state;

Example

.sample-class {

animation: none 2 ease 0.5 4 normal none running;

}

The animation property is a shorthand for the sub-properties below:

animation-name

animation-duration

animation-timing-function

animation-delay

animation-iteration-count

animation-direction

animation-fill-mode

animation-play-state

The values not mentioned are given default values.

Animation-name property is used to tie-in the @keyframes rule.

@keyframes

Syntax: @keyframes mymove {

  from {property: value}

  to { property: value }

}

Example:

@keyframes animation-name {

from {bottom: 0px;}

to {bottom: 100px;}

}

Percentage denotes the timing of the animation.

Alternative syntax

@keyframes animation-name {

/\* declare actions here \*/

}

Example

@keyframes animation-name {

0%,100%{

background-color: blue;

}

50% {

background-color: green;

}

}

Multiple animations

Works the same as regular animation, multiple rules can be set.

#some-class{

animation: animation-a 2s linear infinite alternate,

animation-b 3s ease infinite alternate;

}