

# Table of Contents

For the latest slides see:

<https://accviscom.com/klarence-ouyang/assets/pdf/slides.pdf>

<https://accviscom.com/klarence-ouyang/assets/pdf/slides-with-notes.pdf>

## **Links to Slides**

### **Competency 1**

- [Screen reader practice](#)
- [Lecture: What is a Website?](#)
- [Setup VS Code](#)
- [Intro to HTML](#)
- [HTML Syntax](#)
- [Skeleton Code](#)
- [Basic Text Elements](#)
- [How to organize folders and files of a website](#)
- [block vs. inline elements](#)
- [Choosing appropriate image formats](#)
- [Adding images](#)
- [Semantic Elements](#)
- [Setup FTP](#)

## **Links to Slides**

### **Competency 2**

- [The Anchor Element - Links](#)
- [IDs and Classes](#)
- [Intro to CSS](#)
- [CSS Selectors](#)
- [CSS Color](#)
- [CSS Typography](#)
- [CSS size units](#)
- [Browser developer tools](#)
- [Styling lists](#)
- [div & span tags](#)
- [Code Validation](#)
- [Website Checklist](#)

## **Links to Slides**

### **Competency 3**

- [The display property](#)
- [The CSS Box Model](#)
- [Link states](#)
- [Web fonts](#)
- [File structure and file trees](#)
- [Build and style navigation](#)
- [Shared css](#)
- [Background images](#)
- [Fluid width](#)

## **Links to Slides**

### **Competency 4**

- [Position](#)
- [Forms](#)
- [Flexbox](#)
- [Wireframe a Website](#)

## **Links to Slides**

### **Competency 5**

- [CSS Grid](#)
- [Media queries & mobile-first design](#)

## **Links to Slides**

### **Competency 6**

- Lecture: [Accessibility for Everyone](#)
- Lecture: [Designing for Accessibility](#)
- Demo [Lighthouse](#)

- Lecture: Accessibility for Everyone
- Practice using provided assistive technologies (in-class only)
- Lecture: Designing for Accessibility
- Demo Lighthouse

# What is a Website?

<https://accviscom.com/klarence-ouyang/assets/pdf/what-is-a-website.pdf>

<https://drive.google.com/file/d/1rDh7KrtuC7zja7QIFgkg4g6XNLYlvhLb/view?usp=sharing>

# Setup Visual Studio Code

[https://docs.google.com/document/d/18sibh0OX4uC9IJTpqZF\\_W0nxXuO6beivYwL34y0yQPU/edit?usp=sharing](https://docs.google.com/document/d/18sibh0OX4uC9IJTpqZF_W0nxXuO6beivYwL34y0yQPU/edit?usp=sharing)

<https://code.visualstudio.com/download>

<https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer>

# Intro to HTML

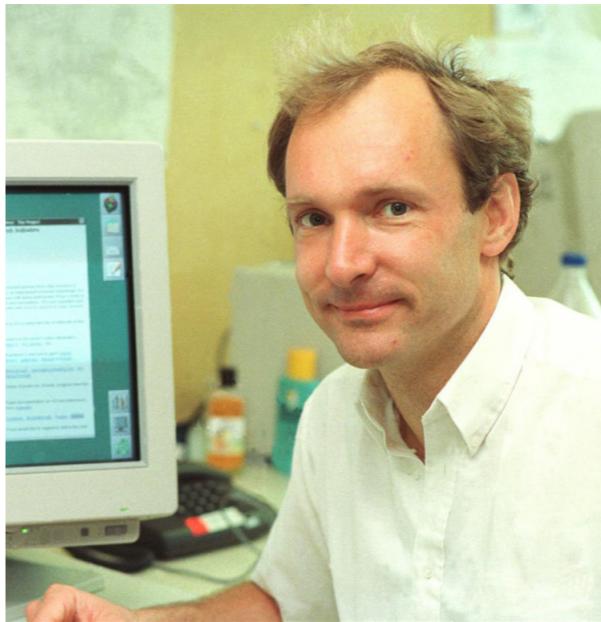
the markup language  
for the Web

# What is HTML

**HTML stands for Hypertext**

**Markup Language.**

In 1990, computer scientist Tim Berners-Lee created HTML to display information from people in remote places.



Who - Tim Berners-Lee , creator of the World Wide Web

What - a Markup Language for Hypertext

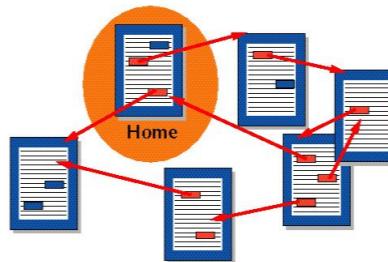
Where - all over the world

When - 1990

Why - to display information from people in remote places

## HyperText

- Hypertext is text displayed on a computer display or other electronic devices with references (hyperlinks) to other text that the reader can immediately access.
- Hypertext documents are interconnected by hyperlinks, which are typically activated by a mouse click, keypress set, or screen touch.
- Hypermedia - Apart from text, the term "hypertext" is also sometimes used to describe tables, images, and other presentational content formats with integrated hyperlinks.



<https://en.wikipedia.org/wiki/Hypertext>

# **HTML**

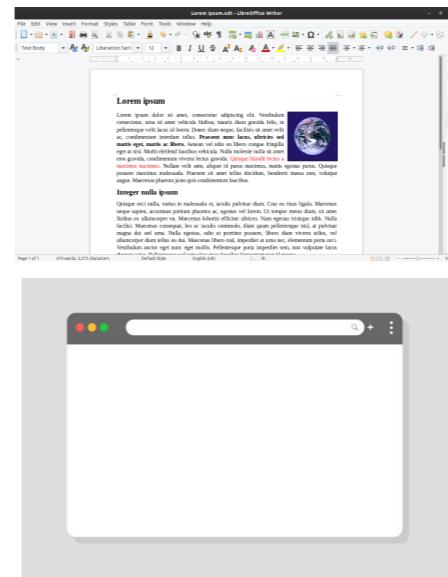
## **Hypertext Markup Language**

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

# HTML Syntax

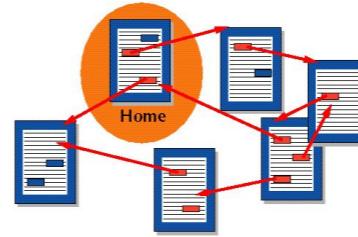
# HTML Basics

- **HTML Documents**
  - Version Declaration
  - HTML Elements
  - HTML Attributes



## HTML Basics

- **HTML Documents**
  - Version Declaration
  - HTML Elements
  - HTML Attributes



Each of these blue rectangles are a document, an HTML document.

The orange circle is where we currently are, the home page. The small red rectangles are links that bring you to the different HTML documents.

## HTML Basics

- HTML Documents
  - **Version Declaration**
  - HTML Elements
  - HTML Attributes

```
<!doctype html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
```



```
</body>
</html>
```

The current version of HTML is version 5

HTML Specification - <https://html.spec.whatwg.org/multipage/>

doctype is not case-sensitive, <!DOCTYPE html>

Most HTML and CSS is case-insensitive. Exceptions are attribute values and CSS values for URL's and some font names.

W3C recommends using lowercase.

XHTML requires lowercase.

### HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

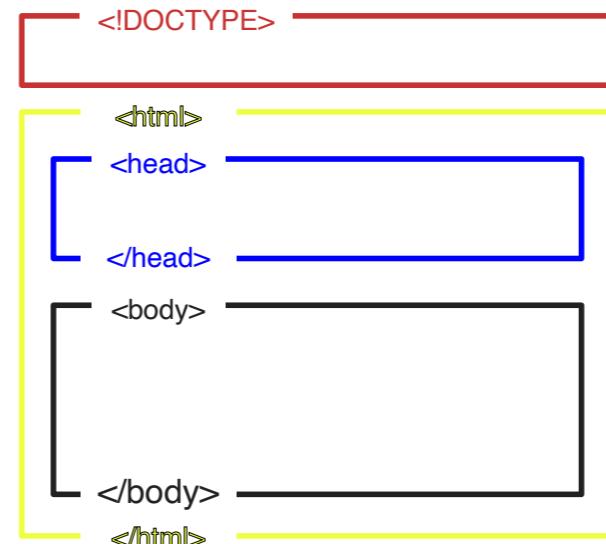
### XHTML 1.1:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

# HTML Is About Structure and Content

All HTML pages have consistent structure:

1. The `<!DOCTYPE html>` declaration defines this document to be HTML5.
2. The `<html>` element is the root element of an HTML page.
3. The `<head>` element contains meta information about the document.
4. The `<title>` element specifies a title for the document.
5. The `<body>` element contains the visible page content.



- <https://developer.mozilla.org/en-US/docs/Web/HTML>

In this example, the head and the body element are children elements of the parent html tag.

## HTML Basics

- HTML Documents
  - Version Declaration
  - **HTML Elements**
  - HTML Attributes

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

The current version of HTML is version 5

HTML Specification - <https://html.spec.whatwg.org/multipage/>

# HTML Basics

- HTML Documents
  - Version Declaration
  - **HTML Elements**
  - HTML Attributes

<!-- heading levels go up to 6 -->

```
<h1>My main title</h1>
<h2>My top level heading</h2>
<h3>My subheading</h3>
<h4>My sub-subheading</h4>
```

```
<p>My first paragraph.</p>
<p>My first paragraph.</p>
```

```
<ol>
  <li>1st Item</li>
  <li>2nd Item</li>
  <li>3rd Item</li>
</ol>
```

The current version of HTML is version 5

HTML Specification - <https://html.spec.whatwg.org/multipage/>

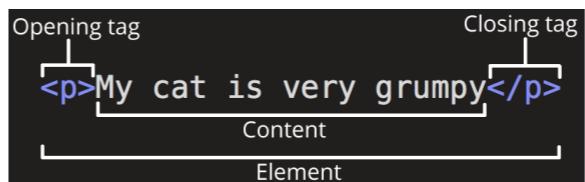
# HTML Basics

- HTML Documents
  - Version Declaration
  - **HTML Elements**
    - HTML Attributes

**<!DOCTYPE html>**

**<html>**

**<body>**



**</body>**

**</html>**

HTML tags are used to define elements.

Most elements have a opening and closing tag.

A tag consists of an opening or left arrow bracket (<) and a closing or right angle bracket (>). Most elements have a closing tag (</>) which also has a forward slash (/) right after the opening arrow bracket (<).

However, there are some elements that only have one tag (a Void Element)

The <img> is a void element.



## HTML Basics

- HTML Documents
  - Version Declaration
  - **HTML Elements**
    - HTML Attributes

```
<!-- Self-closing tags -->
<!-- Void (empty) Elements don't have content -->
<!-- so they don't have a closing tag -->
<!DOCTYPE html>
<html>
<body>
 😊
</body>
</html>
```

HTML tags are used to define elements.

Most elements have a opening and closing tag.

However, there are some elements that only have one tag (a Void Element)

The `<img>` is a void element.

```

```

Self-closing tags (`<tag />`) do not exist in HTML.

## Void Elements

aka Empty Elements

- Do **not** have a closing tag
- Cannot have any child nodes (i.e., nested elements or text nodes)
  - content is created by using attributes
    - src <https://en.wikipedia.org/wiki/File:Smiley.svg> 😊

HTML tags are used to define elements.

Most elements have a opening and closing tag.

However, there are some elements that only have one tag (a Void Element)

The <img> is a void element.

```

```

Self-closing tags (<tag />) do not exist in HTML.

## Nesting Elements

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

    <p>These dolls <strong>nest</strong>
       inside each other.
    

</p>

</body>
</html>
```

Parent  
Child

These dolls **nest** inside each other.



HTML tags are used to define elements.

Most elements have a opening and closing tag.

However, there are some elements that only have one tag (a Void Element, aka Empty Element)

The `<img>` is a void element.

```

```

Self-closing tags (`<tag />`) do not exist in HTML.

<!-- Anything wrapped between two tags is known as a parent-child relationship. This simply means that opening and closing tags wrap another html element. This becomes important later when building website structure -->

## Nesting Elements

```
<!DOCTYPE html>
<html>
<body>
<p>
  My cat is <strong>very</strong> grumpy.
  <a href="https://en.wikipedia.org/wiki/Grumpy_Cat">
    
  </a>
</p>
</body>
</html>
```

My cat is **very** grumpy.



HTML tags are used to define elements.

Most elements have a opening and closing tag.

However, there are some elements that only have one tag (a Void Element)

The `<img>` is a void element.

```

```

Self-closing tags (`<tag />`) do not exist in HTML.

## HTML Basics

- HTML Documents
  - Version Declaration
  - HTML Elements
  - **HTML Attributes**

```
<!DOCTYPE html>
<html>
<body>
    <p class="editor-note">My cat is very grumpy</p>
</body>
</html>
```

Attribute

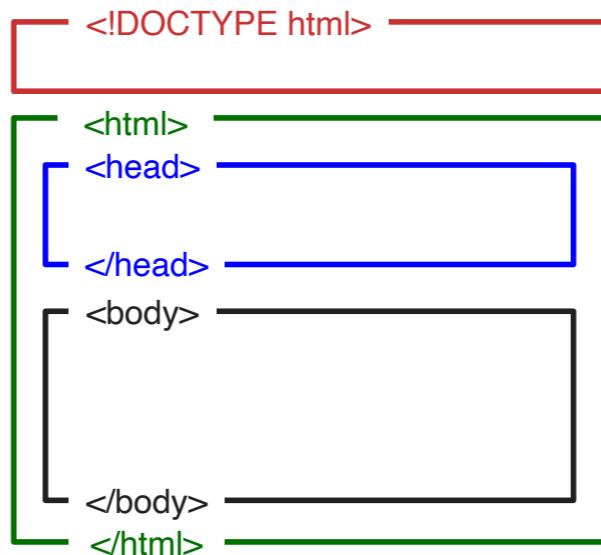
We introduced src and alt attributes for the image tag.

**Skeleton Code  
(Boilerplate HTML)**

## Skeleton Code

All HTML pages have consistent structure:

1. The `<!DOCTYPE html>` declaration defines this document to be HTML5.
2. The `<html>` element is the root element of an HTML page.
3. The `<head>` element contains meta information about the document.
4. The `<title>` element specifies a title for the document.
5. The `<body>` element contains the visible page content.



## Skeleton Code aka HTML Boilerplate

```
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>HTML 5 Boilerplate</title>

  </head>
  <body>
    // Your Code Here
    <script src="index.js"></script>
  </body>
</html>
```

<https://codepen.io/Klarence/pen/LEPKOOM>

Give link, and show how to use emmet (! + tab or enter)

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

Both of these are valid, even though HTML attribute values are one of the few places HTML is case sensitive.

NOTE: The quizzes code in the book must be lowercase.

<https://www.freecodecamp.org/news/basic-html5-template-boilerplate-code-example/>

# Basic Text Elements

## Basic Text Elements

- headings `<h1>` thru `<h6>`
- paragraphs `<p>`
- Strong and Emphasis `<strong>` & `<em>`
- small `<small>`
- links `<a>` (anchor tag)
- lists `<ol>` and `<ul>`
  - List Items `<li>`
- Breaking Elements `<br>` & `<hr>`

<https://codepen.io/Klarence/pen/yyBdPzX>

tag and element are often used interchangeably, but an element usually consists of two tags. The element also consists of all of the content in between the tags, including other, nested elements.

All the HTML (version 5) elements/tags are case-insensitive. Case-insensitive means the tags/elements which are used in the code are understandable by the browser. Best practice is to always use lowercase.

## Basic Text Elements

- headings <h1> thru <h6>
- paragraphs <p>
- Strong and Emphasis <strong> & <em>
- small <small>
- links <a> (anchor tag)
- lists <ol> and <ul>
  - List Items <li>
- Breaking Elements <br> & <hr>

<https://codepen.io/Klarence/pen/yyBdPzX>

tag and element are often used interchangeably, but an element usually consists of two tags. The element also consists of all of the content in between the tags, including other, nested elements.

All the HTML (version 5) elements/tags are case-insensitive. Case-insensitive means the tags/elements which are used in the code are understandable by the browser. Best practice is to always use lowercase.

Remember HTML for content, CSS for presentation.

## Header Element

**<head> vs <header> vs heading <h1> – <h6>**

```
<!doctype html>
<html lang="en">
<head>
  <!-- meta info -->
</head>
<body>
<header>
  <!-- intro content -->
</header>
<main>
  <h1>Page Title</h1>
</main>
</body>
</html>
```

# Header Element

`<header>, taglines`

```
<header>
  <!-- intro content, logo, nav, etc. -->
</header>

<header>
  <h1>Company, <span class="diff-style">Tagline</span></h1>
</header>

<header>
  <h1>Company</h1>
  <p>Tagline</p>
</header>
```

`<hgroup>
 <h1>Company</h1>
 <p>Tagline</p>
</hgroup>`

Don't use more than one heading element in an hgroup element.

<https://codelucky.com/html-hgroup-tag/> (This is WRONG)



# File and Folder Organization

## File paths

- `/` is the root
- `./` is the current directory (cwd)
- `../` is the parent of the current directory.
- `../../` is the grandparent



<https://www.youtube.com/watch?v=ephId3mYu9o>

if you omit the slash in the beginning `./` (relative to current directory) is assumed

<https://superuser.com/questions/153165/what-does-represent-while-giving-path>

block vs inline  
Display of elements

## block vs inline

block	inline
takes up the full width available	only takes up as much width as necessary
will break to a new line	will not break onto a new line
doesn't allow other elements to sit behind it	allows other elements to sit behind it
height & width respected	height & width don't apply (vertical margin & padding don't work)
all sides margin work as expected <u>*margin between block elements collapse</u>	top and bottom margin doesn't affect other inline elements



demo

<https://codepen.io/Klarence/pen/OPLdXbz>

HTML - Block and Inline - W3Schools.com-

[https://www.youtube.com/watch?v=M4n-WSkehml&list=PLP9lO4UYNF0VdAajP\\_5pYG-jG2JRrG72s](https://www.youtube.com/watch?v=M4n-WSkehml&list=PLP9lO4UYNF0VdAajP_5pYG-jG2JRrG72s)

# Choosing Image Formats

# Image Formats

## Older Formats

- JPEG
- PNG
- SVG
- GIF

## Modern Formats

- WEBP
- AVIF
- APNG

# Image Formats

## Older Formats (Historical Support)

- **JPEG**  
photos, no transparency, lossy
- **PNG**  
supports transparency
- **SVG**  
Vector based, good for icons, diagrams, etc.
- **GIF**  
for simple images/animations

## Modern Formats (Better Performance)

- **WebP**  
good for web graphics  
not good for offline use, 8-bit color
- **AVIF**  
good for digital photos  
best picture, best compression, least support
- **APNG**  
better support but not as performant as AVIF and WebP,  
more performance than GIF.

JPEG is the most popular. WebP has support for all major browsers.

You should use the picture element if you plan to use the modern formats.

I use WebP for pretty much everything.

AVIF has higher compression efficiency, with file sizes 50% smaller than JPEG and 20-30% smaller than WebP ([source](#)).

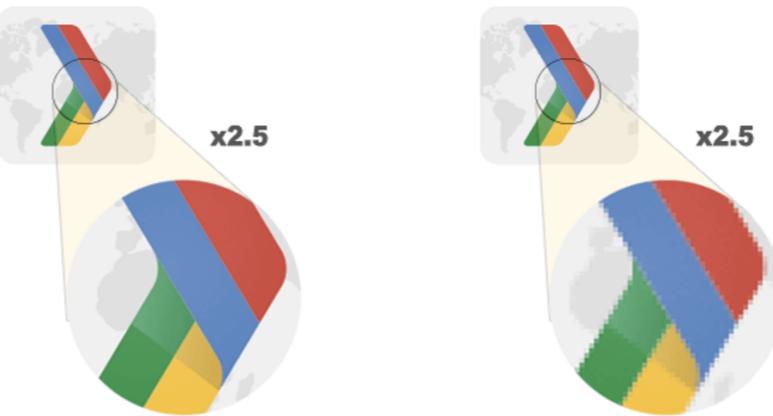
[https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image\\_types](https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image_types)

<https://web.dev/articles/choose-the-right-image-format>

## Vector vs Raster

**Vector graphics** use lines, points, and polygons to represent an image.

**Raster graphics** represent an image by encoding the individual values of each pixel within a rectangular grid.



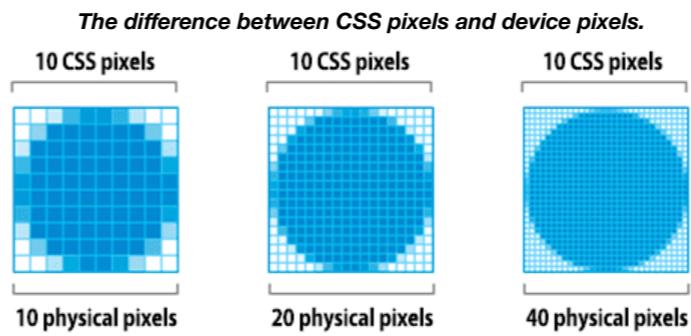
Vectors are good for logos, icons, diagrams, things that need to be scaled infinitely.

Raster Graphics are good for photos.

<https://web.dev/articles/choose-the-right-image-format>

## Hi-Res Screens

You don't have to worry about using different pixel values in CSS based on the devices resolution.



[https://web.dev/articles/choose-the-right-image-format#implications\\_of\\_high-resolution\\_screens](https://web.dev/articles/choose-the-right-image-format#implications_of_high-resolution_screens)

# Adding Images

## Adding Images

1. `<img>`
2. `<picture>`
3. Inline `<svg>`
4. **background-image**  
CSS property

There are four ways to add images to your webpage.

- `<img>`
  - remember to add an alt attribute
  - remember width & height for page load
- `<picture>`
  - newer tag with features for fallbacks
- Inline `<svg>`
  - good for customizing parts of the SVG
- background-image CSS property
  - generally avoid due to poor a11y
  - okay for decoration (not content)

## Adding Images

### 1. <img>

- remember to add an **alt** attribute
- remember **width & height** for page load

### 2. <picture>

- newer tag with features for fallbacks

### 3. Inline <svg>

- good for customizing parts of the SVG

### 4. **background-image** CSS property

- generally avoid due to poor a11y
- okay for decoration (not content)

## <img> The Image Element

- <img> is a self-closing element (aka Empty Element or Void Element)
  - the **src** attribute is the attribute that references the images location or path.
  - the **alt** attribute is descriptive text that *screen readers* read aloud or if the file doesn't load the value will display as text on the screen
  - **width & height** attributes are not necessary, but help page load
    - CSS is used for (responsive) sizing



Void Element is the technical term

<https://htmlandcssguidebook.com/html/images/#optional-attributes>

Mention title is a global attribute and is not recommended for a11y reasons. SR don't read them. You should have the same experience for both users.

## <img> The Image Element

```

```



Void Element is the technical term

<https://htmlandcssguidebook.com/html/images/#optional-attributes>

Mention title is a global attribute and is not recommended for a11y reasons. SR don't read them. You should have the same experience for both users.

## <img> The Image Element

The **src** attribute's value refers to the image location or file path.

```
<!--Relative image path-->  

```

```
<!--Absolute image path-->  

```



Void Element is the technical term

<https://htmlandcssguidebook.com/html/images/#optional-attributes>

Mention title is a global attribute and is not recommended for a11y reasons. SR don't read them. You should have the same experience for both users.

## <picture> The Picture Element

The **picture** element is a newer element that allows you to specify multiple sources. The source will be chosen depending on the supported image type, device size, orientation, etc.

```
<picture>
  <source srcset="folder/hi-res-file.avif"
          type="image/avif"
          media="(min-width: 1280px)"
  <source srcset="folder/hi-res-file.jpg"
          type="image/jpeg"
          media="(min-width: 1280px)"
  
</picture>
```

You should always have an image element that specifies an **alt** and a backup **src** attributes.



If the picture element or a specific source's type isn't supported by the browser then it is skipped. Just like if the media attribute doesn't match it is skipped.

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/picture>

The picture element still uses the img element.

<source> attributes

- srcset
- media
- type

<https://htmlandcssguidebook.com/html/images/#optional-attributes>

Mention title is a global attribute and is not recommended for a11y reasons. SR don't read them. You should have the same experience for both users.

# Semantic Elements

## HTML 5 Semantic Elements

- <header>
- <footer>
- <main>
- <article>
- <section>
- <aside>
- <nav>
- <figure> & <figcaption>

- <details>
- <summary>
- <mark>
- <time>



\* <div> & <span> are non-semantic tags used for layout.

<https://www.freecodecamp.org/news/semantic-html5-elements/>

Semantics are important for SEO, a11y, navigating with keyboard or screenreader.

## HTML 5 Semantic Elements

- <header>
- <footer>
- <main> - only one per page
- <article>
- <section> - should always have a heading
- <aside>
- <nav>
- <figure> & <figcaption>

- <details>
  - <summary>
  - <mark>
  - <time>
- \* <div> & <span> are non-semantic tags used for layout.



div is used to group content together and both div & span are used to apply styles to the content.

<https://www.freecodecamp.org/news/semantic-html5-elements/>

Semantics are important for SEO, a11y, navigating with keyboard or screenreader.

Usually a <figure> is an image, illustration, diagram, code snippet, etc., that is referenced in the main flow of a document, but that can be moved to another part of the document or to an appendix without affecting the main flow.

A caption can be associated with the <figure> element by inserting a <figcaption> inside it (as the first or the last child). The first <figcaption> element found in the figure is presented as the figure's caption.

The <figcaption> provides the accessible name for the parent <figure>.

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/figure>

# FTP Setup

[FTP Instructions](#)

# **Links**

## **The Anchor Element < a >**

## Anchor Element <a>

```
<a href="https://google.com">Google</a>
```

**The most important attribute for links (<a>) is the href attribute.**

**It indicates where the link take you or what to do.**

```
<a href="https://google.com" target="_blank">Google</a>
```

**The target attribute with a value of \_blank open the link in a new tab/browser.**

## Anchor Element <a>

- **Absolute links**

- Includes domain (and protocol)
  - <https://www.mysite.com>
  - <tel:+123456789>
  - <mailto:name@email.com>

- **Relative links**

- path to a file in your local workspace

- **Fragment / Jump links**

- References an id attribute
  - Scrolls to section of page
  - Can autoplay media or highlight text



<https://developer.mozilla.org/en-US/docs/Web/URI/Fragment>

**id and class  
attributes & selectors**

# **id and class**

## **IDs**

- are unique
- Each element can only have one ID
- Each page can only have one ID
- lowerCamelCase is a common convention

## **Classes**

- are not unique
- can be used on many elements
- One element can have multiple classes
- Are case sensitive
  - don't use uppercase,
  - stick to dash and underline separators

## **id and class** Selectors

### **IDs**

#nameOfTheId

### **Classes**

.one-class  
.anotherclass  
.yet\_another\_class

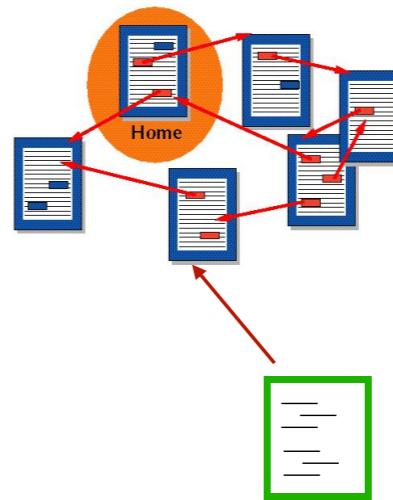
CSS Naming Convention  
Block Element Modifier  
<https://getbem.com/naming/>

# CSS Intro

# Cascading Style Sheets

## Stylesheet

A **stylesheet** is a set of CSS rules used to control the layout and design of a webpage or document.



HTML is the Content

CSS is the Presentation, How it looks

<https://developer.mozilla.org/en-US/docs/Glossary/Stylesheet>

## Cascade

What does the “cascading” in CSS mean?

- The “cascading” in CSS refers to the fact that styling rules “cascade” down from several sources. This means that CSS has an inherent hierarchy and styles of a higher precedence will overwrite rules of a lower precedence.

<https://discuss.codecademy.com/t/what-does-the-cascading-in-css-mean/340936>

## Cascading Order



Rule of thumb: Closest to the code wins.

- **Inline styles:** Highest in priority (*unless overridden by !important*)  
Internal CSS: Styles within a <style> tag - only apply to that document.
- **External CSS:** Styles from external CSS files linked to the HTML document, which apply globally across multiple pages.
- **Browser/User styles:** Defaults or user-defined preferences.

Only use !important for debugging. Don't deploy it live to production.

CSS Rules Hierarchy is based on Specificity and the Cascade order.  
Cascade is secondary to specificity.

<https://www.geeksforgeeks.org/what-does-the-cascading-portion-of-css-means/>

## Cascading Order

- **Inline styles:** Highest in priority (*unless overridden by !important*)  
•  
•  
•

```
<p style="font-size: 24px">  
    This is my first paragraph  
</p>
```

Only use !important for debugging. Don't deploy it live to production.

CSS Rules Hierarchy is based on Specificity and the Cascade order.  
Cascade is secondary to specificity.

<https://www.geeksforgeeks.org/what-does-the-cascading-portion-of-css-means/>

## Cascading Order

- 

**Internal CSS:** Styles within a `<style>` tag - only apply to that document.

```
<!DOCTYPE html>
<html lang="en">
  <head>

    <style>
      .first-p {
        color: red;
      }
    </style>

  </head>
  <body>
    <p class="first-p">
      This is my first paragraph
    </p>
  </body>
</html>
```

Only use `!important` for debugging. Don't deploy it live to production.

CSS Rules Hierarchy is based on Specificity and the Cascade order.  
Cascade is secondary to specificity.

<https://www.geeksforgeeks.org/what-does-the-cascading-portion-of-css-means/>

## Cascading Order

- **Internal CSS:** Styles within a `<style>` tag - only apply to that document.
- **External CSS:** Styles from external CSS files linked to the HTML document, which apply globally across multiple pages.
- **Browser/User styles:** Defaults or user-defined preferences.

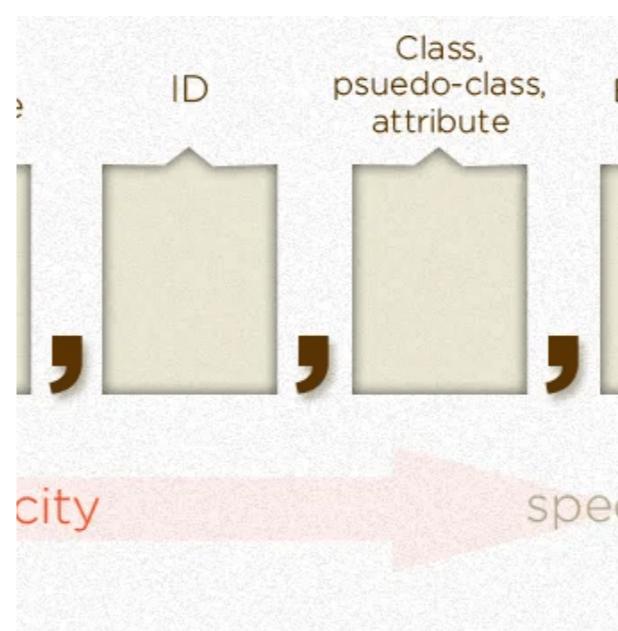
Only use `!important` for debugging. Don't deploy it live to production.

CSS Rules Hierarchy is based on Specificity and the Cascade order.  
Cascade is secondary to specificity.

<https://www.geeksforgeeks.org/what-does-the-cascading-portion-of-css-means/>

## Specificity

- If the element has inline styling, that automatically wins (1,0,0,0 points)
- For each ID value, apply 0,1,0,0 points
- For each class value (or pseudo-class or attribute selector), apply 0,0,1,0 points
- For each element reference, apply 0,0,0,1 point

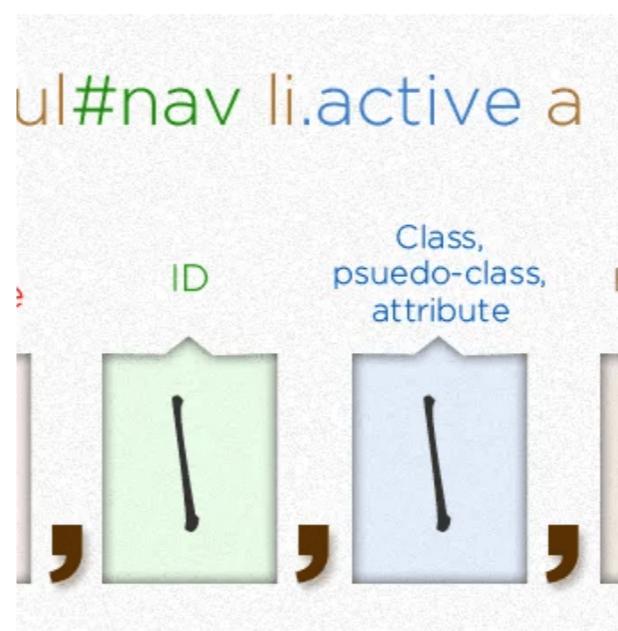


<https://css-tricks.com/specifc-on-css-specificity/>

<https://specificity.keegan.st/>

## CSS Specificity

```
<nav>  
  
</nav>
```



<https://css-tricks.com/specifics-on-css-specificity/>

## !important

```
<span>  
<strong>!important</strong>; Highest priority, regardless of source.  
</span>
```

```
...  
strong {  
font-weight: bold !important;  
}
```

**!important:** Highest priority, regardless of source.

- beats all specificity and cascade
- meant only for debugging

<https://www.geeksforgeeks.org/what-does-the-cascading-portion-of-css-means/>

## Inheritance

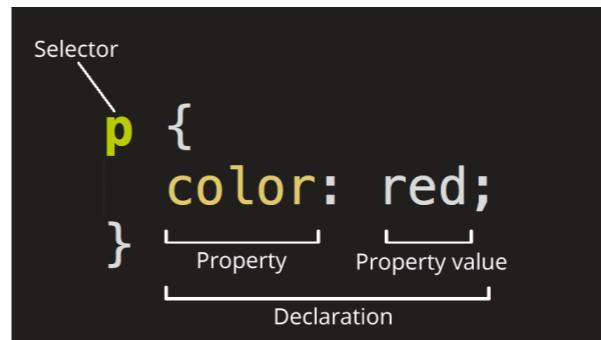
Some CSS property values set on parent elements are **inherited** by their child elements, and some aren't

[https://developer.mozilla.org/en-US/docs/Learn\\_web\\_development/Core/Styling\\_basics/Handling\\_conflicts#inheritance](https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Handling_conflicts#inheritance)

text-align, color,

## CSS Syntax

- **CSS Ruleset**
  - Selector
  - **Declaration**
    - Property
    - Property Value



Sometimes the entire ruleset is referred to as a rule, sometimes each declaration is referred to as a rule.

[https://developer.mozilla.org/en-US/docs/Learn\\_web\\_development/Getting\\_started/Your\\_first\\_website/Styling\\_the\\_content](https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Your_first_website/Styling_the_content)

# CSS Selectors

## CSS Selectors

- Element Selector (aka Tag or Type Selector)
- **Class Selector**
- Pseudo Selectors (Pseudo-class /Pseudo-element)
- ID selector
- Attribute selector
- Universal Selector

Pseudo-class (Pseudo-element ) selector

Pseudo-elements behave in a similar way. However, they act as if you had added a whole new HTML element into the markup, rather than applying a class to existing elements.

Pseudo-elements start with a double colon ::. ::before is an example of a pseudo-element.

[https://developer.mozilla.org/en-US/docs/Learn\\_web\\_development/Core/Styling\\_basics/Pseudo\\_classes\\_and\\_elements](https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Pseudo_classes_and_elements)

## attribute selector

**attributes can be selected as well**

```
[hidden] {  
    display: none;  
    // do NOT override  
}  
  
// you don't need to add this as  
browsers will automatically have  
this style
```

[https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute\\_selectors](https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors)

[https://developer.mozilla.org/en-US/docs/Web/HTML/Global\\_attributes/hidden](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/hidden)

## attribute selector

**attributes can be selected as well**

```
[title] {  
  color: red;  
  font-size: 10rem;  
  // to help find when title's are being used  
}  
  
// you should not use the title attribute
```

This is a style I would use for local  
development but not actually deploy live  
to production

[https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute\\_selectors](https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors)

[https://developer.mozilla.org/en-US/docs/Web/HTML/Global\\_attributes/title](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/title)

## CSS Combinators

- Descendant combinator ( space )
- Child combinator (>)
- Next-sibling combinator (+)
- Subsequent-sibling combinator (~)
  - Class Selectors
  - ID selectors

[https://developer.mozilla.org/en-US/docs/Learn\\_web\\_development/Core/Styling\\_basics/Combinators#child\\_combinator](https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Combinators#child_combinator)

# CSS Color

## CSS Color

- Hexadecimal
- Named Colors
- RGB(a)
- And more
  - HSL, HWB, etc.

[https://www.w3schools.com/cssref/css\\_colors.php](https://www.w3schools.com/cssref/css_colors.php)

[https://developer.mozilla.org/en-US/docs/Web/CSS/color\\_value](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value)

## Hexadecimal Color

- Hexadecimal
  - #000000 to #FFFFFF
  - A base16 numbering systems
    - 16 symbols (0-9 and A-F)
    - #rrggb (rr - red, gg - green, bb - blue)
    - #rrggbbaa - you can add two digits for alpha (transparency)

[https://www.w3schools.com/colors/colors\\_hexadecimal.asp](https://www.w3schools.com/colors/colors_hexadecimal.asp)

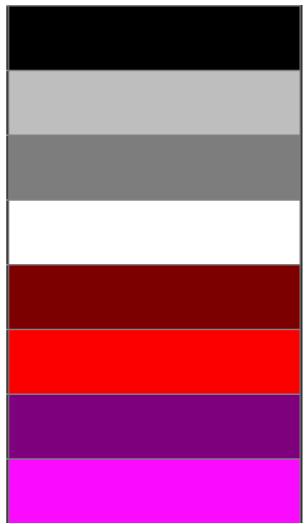
Each set of two digits represent R, G, B with a total of 256 values for each color. 00 for no color (black) and ff for full color (#ffffff is white) .

Hexidecimals with transparency can be used as well.

<https://caniuse.com/?search=alpha%20hexadecimal>

## Named Colors

black  
silver  
gray  
white  
maroon  
red  
purple  
fuchsia  
green  
lime  
olive  
yellow



<https://developer.mozilla.org/en-US/docs/Web/CSS/named-color#value>

## RGB(a) Colors

```
color: rgb(255, 255, 255);
```

- RGB
  - rgb(red, green, blue)
  - There are 256 values from 0 - 255.
- RGBA
  - rgba(red, green, blue, alpha)
  - The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

[https://www.w3schools.com/cssref/css\\_colors.php](https://www.w3schools.com/cssref/css_colors.php)

[https://developer.mozilla.org/en-US/docs/Web/CSS/color\\_value](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value)

# CSS Typography

## CSS Typography

- Font-Family
- Font-Size
- Font-Weight
- Font-Style
- Color
- Line-Height
- Text-Transform
- Text-Decoration
- Letter-Spacing

<https://codepen.io/Klarence/pen/RNwNqpZ>

# CSS size units

## CSS Size Units

- Pixels
- Ems & Rems
- Percentages
- Viewport Units

## CSS Pixels

- **Pixels (px)**
  - $1\text{px} = 1/96\text{th}$  of 1in

## Ems & Rems

- **Ems** (em)
  - The size of the character "m"
  - A relative size based on its parent container
- **Rems** (rem)
  - Root Em, the size of the character "m" on the root element (html)
    - Default is 16px for most browsers

## CSS Viewport Units

- **Viewport Units**
  - vh - Viewport Height
  - vw - Viewport Width
  - vmin - the smaller of vw or vh
  - vmax - the larger of vw or vh

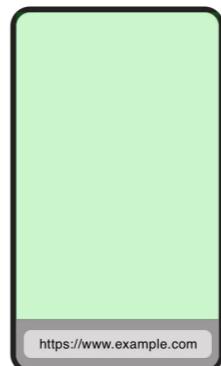
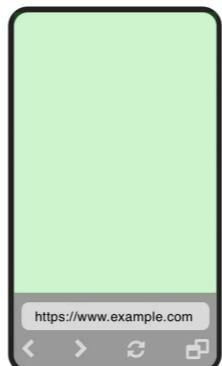
New units Viewport Inline (vi) and Viewport Block (vb)

<https://www.geeksforgeeks.org/new-css-viewport-units-vi-vb-vmax-vmin/>

<https://blog.webdevsimplified.com/2022-08/css-viewport-units/>

## CSS Viewport Units

- Viewport Unit Modifiers
  - s - Small
  - l - Large
  - d - Dynamic



<https://blog.webdevsimplified.com/2022-08/css-viewport-units/>

# Browser Dev Tools

# Inspect - Dev Tools

## Elements Panel

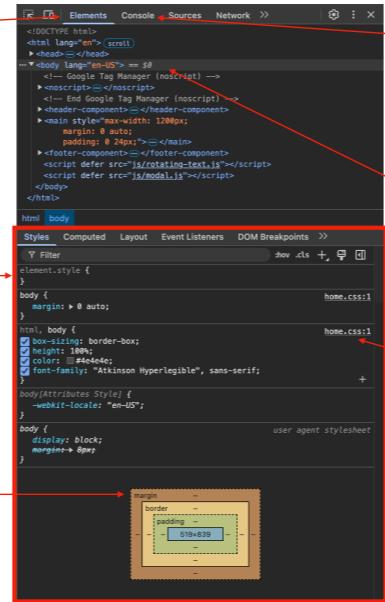
Shows the DOM (HTML Structure) And the Style Pane by default.

## element.style

You can apply temporary inline styles.  
It will go away when the page is refreshed.  
(click in between the brackets to edit)

## Box Model

Lets you know the size of your elements content as well as any padding, border, and margin.



## Console Panel

Mainly used for JavaScript, but it will notify you if there are issues linking to your images or CSS files

## Highlighted Element

Let you know which element is selected and updates the below sub-panels (Styles).

## Existing Selectors

You can also modify existing selectors

## Styles Sub-panel

Mainly used for JavaScript, but it will notify you if there are issues linking to your images or CSS files

Go over the Computed and Layout Panes.

## Chrome Inspect

- Keyboard Shortcut to open
  - Mac: cmd + shift + i
  - PC: ctrl + shift + i or F12
- Elements Panel
  - Prototyping
- Network Tab
  - Loading
- Lighthouse
  - Performance



# List Styling

<https://codepen.io/Klarence/pen/mybZqjd>

**The Division & Span elements**  
`<div> | <span>`

## The Division and Span Element

- <div> and <span> have no meaning
- They should only be used when no other elements make sense
- They are used to wrap content in order to apply styles to the nested elements.
  - <div> is a **block** container (*by default*)
  - <span> is a **inline** container (*by default*)

Demo span and div

<https://codepen.io/Klarence/pen/KwKKXGr>

# Code Validation

[https://validator.w3.org/#validate\\_by\\_input](https://validator.w3.org/#validate_by_input)

[https://jigsaw.w3.org/css-validator/#validate\\_by\\_input](https://jigsaw.w3.org/css-validator/#validate_by_input)

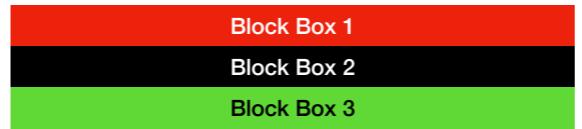
# Website Checklist

[https://docs.google.com/document/d/1X8zbE8bv-QmD\\_GZBkRXzh2cGK3QcKkcv6kMhr0H4EXE/edit?usp=sharing](https://docs.google.com/document/d/1X8zbE8bv-QmD_GZBkRXzh2cGK3QcKkcv6kMhr0H4EXE/edit?usp=sharing)

# The display Property

## Display - Block

- **Block**
- Inline
- Inline-Block
- None



The element generates a block box, generating line breaks both before and after the element when in the normal flow.

## Display - Inline

- Block
- **Inline**
- Inline-Block
- None

Inline Box 1 | Inline Box 2 | Inline Box 3

**The element generates one or more inline boxes that do not generate line breaks before or after themselves. In normal flow, the next element will be on the same line if there is space.**

## Display - Inline-block

- Block
- Inline
- **Inline-Block**
- None

The quick brown fox jumps over the lazy dog. The  
quick brown . jumps over the lazy dog

The quick brown fox jumps over the lazy dog. The  
quick brown fox jumps over the lazy dog.

The element generates a block box that will be flowed with surrounding content as if it were a single inline box.

Key
Inline Box 1
Inline-Block Box 2
Inline-Block Box 3

## Display - None

- Block
- Inline
- Inline-Block
- **None**

Turns off the display of an element so that it has no effect on layout (the document is rendered as though the element did not exist). All descendant elements also have their display turned off.



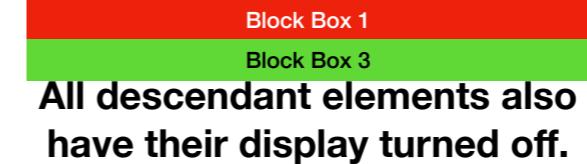
To have an element take up the space that it would normally take, but without actually rendering anything, use the visibility property instead.

Flexbox and CSS Grid will go over in depth later in the class.

## Display - None

- Block
- Inline
- Inline-Block
- **None**

**Turns off the display of an element so that it has no effect on layout (the document is rendered as though the element did not**



**All descendant elements also have their display turned off.**

To have an element take up the space that it would normally take, but without actually rendering anything, use the visibility property instead.

Flexbox and CSS Grid will go over in depth later in the class.

## Display - None

- Block
- Inline
- Inline-Block
- **None**

Turns off the display of an element so that it has no effect on layout (the document is rendered as though the element did not exist).

All descendant elements also have their display turned off.

**a11y:** The element will be removed from the accessibility tree. This will cause the element and all its descendant elements to no longer be announced by screen reading technology.

To have an element take up the space that it would normally take, but without actually rendering anything, use the visibility property instead.

Flexbox and CSS Grid will go over in depth later in the class.

## Display

- Block
- Inline
- Inline-Block
- None
- **Other Display Values**
  - **flex and grid**

### Flexbox (aka flex)

is a layout method for arranging items in *rows OR columns*.

- 1-dimensional

### CSS Grid (aka grid)

is a grid-based layout system, with *rows AND columns*.

- 2-dimensional

## Display - types for layouts

### Flexbox (aka flex)

- is a layout method for arranging items in rows **OR** columns.
- 1-dimensional
- Focus: Alignment
- Good for Flexible Designs
  - content-first designs (components)
  - aligning, distributing, & ordering elements
  - small design implementations
  - elements can be shifted & moved around

### CSS Grid (aka grid)

- is a grid-based layout system, with rows **AND** columns.
- 2-dimensional
- Focus: Grid Layout
- Good for complex designs with structure
  - Overlapping elements
  - Gaps between block elements
  - Precise positioning

Click on links in slide Flexbox and CSS Grid Sections.

You can use both Flexbox and Grid together (as well as the other display properties).

Before CSS Grid, I would often create nested flexbox containers in order to achieve a 2-dimensional layout.

# The CSS Box Model

" Each HTML element you create in your HTML file is represented as a rectangular box. The **content**, **padding**, **border**, and **margin** are built up around one another like the *layers of an onion*. "

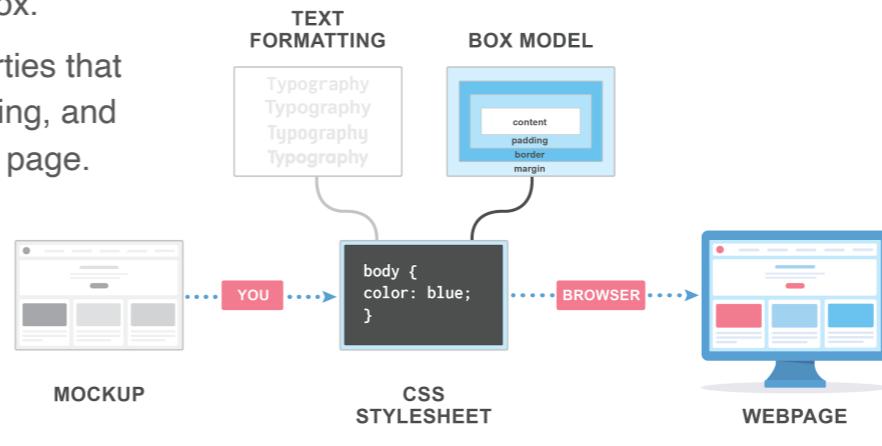
### CSS Box Model

CSS calculates all the boxes on the page to display the content.

## Just a bunch of boxes

CSS treats each element in your HTML document as a box.

The box has different properties that determine its size, spacing, and where it appears on the page.

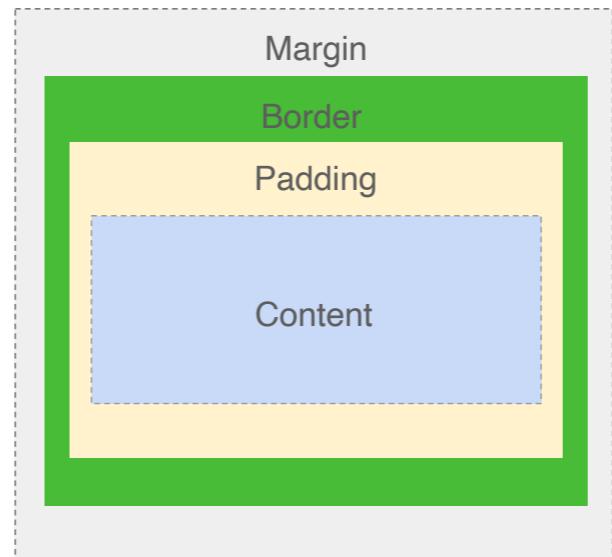


Modified from AdobeStock\_243484168

<https://internetingishard.com/html-and-css/css-box-model/> (no longer exists)

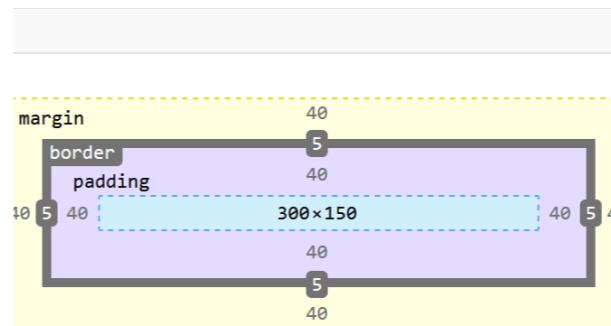
# CSS Box Model

- **Content:** Content of the box where text and images appear.
- **Margin:** The empty area around the border. the margin is completely transparent and has no background color.
- **Border:** A border goes around the padding and the content. The border is affected by the background color of the box
- **Padding:** The padding clears a space for the content within it, but spaces the border and margin. It is completely transparent and can be defined using padding.



## The Box Model

- Each element is made up of 4 boxes
  - content
  - padding
  - border
  - margin



[https://developer.mozilla.org/en-US/docs/Learn\\_web\\_development/Core/Styling\\_basics/Box\\_model#use\\_browser\\_devtools\\_to\\_view\\_the\\_box\\_model](https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Box_model#use_browser_devtools_to_view_the_box_model)

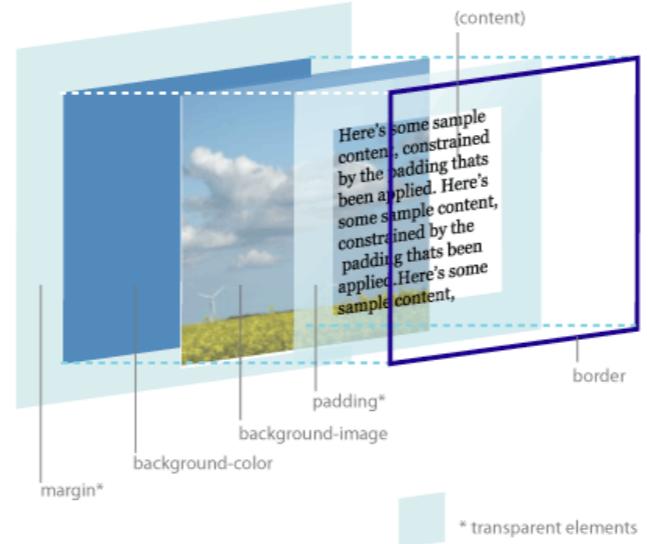
## The CSS Box Model

Let's walk through the properties that affect the CSS box model:

- **Content inside your containers:** Text and images will both cause your box to expand to contain the content inside.
- **Height/Width:** Adding height and width to any of your elements will cause it to expand depending on what properties you set.
- **Margin:** Margin properties are used to create space outside elements. Margin will move your box away from other elements that surround it.
- **Padding:** Padding properties are used to expand space inside the box model.

# CSS Box Model Hierarchy

The box model is the building block of CSS. **Everything you see on the web is a rectangle.**

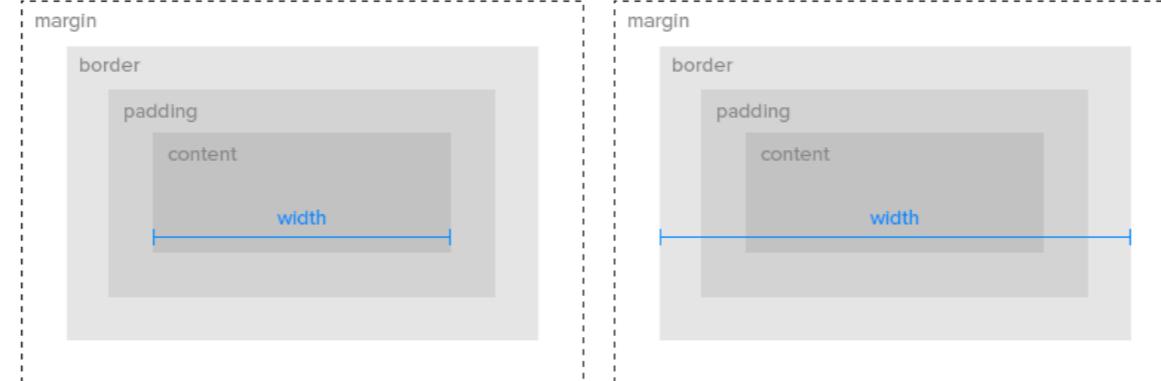


[https://djmarti.com/foothill/coin65/week\\_05/boxModel\\_diagramSummary.html](https://djmarti.com/foothill/coin65/week_05/boxModel_diagramSummary.html)

<https://hackernoon.com/css-box-model-45ecf4ac219e>

## The box-sizing Property

(default)  
box-sizing: content-box;      box-sizing: border-box;



By default when you use height or width you are setting the width of the content.

This can be confusing since the visual portion of a box (excludes margin) is bigger than the width/height you specify.

You need to add the padding and the border (both sides).

You can change the default behavior for the width and height to apply to the border by using box-sizing: border-box;

T, R, B, L - You can remember the order by associating it with clockwise or the sentence:

If you forgot the order you are in TRouBLE.

<https://stackoverflow.com/questions/44453391/what-is-the-difference-between-border-box-and-content-box-in-css>

# Link States

## Link States

### Order Matters

```
a {}  
a:visited {}  
a:hover {}  
a:focus {}  
a:active {}
```

The order that you put your link states matter.  
If you put them in the wrong order, bugs may happen.

a:active is the pressed state.

## Link States

### Order Matters

a {}  
a:visited {}  
a:hover {}  
**a:focus {}**  
a:active {}

**a:focus-visible {}**  
**a:focus-within {}**  
**a:target {}**

There are additional focus states that were added.

a:focus-visible {}  
a:focus-within {}  
a:target {}

These are outside the scope of the book.

# Web Fonts

## Web Fonts

- Cloud Hosted (Google, Adobe, etc.)
- Does Font Mapping for you

<https://fonts.google.com/>

## Font Mapping

In a font-family, you need to **map** each family member with its **weight** and **style**.

```
@font-face {  
    font-family: "MyWebFont";  
    src: url("fonts/myfont-regular.woff2") format("woff2"),  
        url("fonts/myfont-regular.woff") format("woff");  
    font-weight: 400;  
    font-style: normal;  
}
```

## Font Mapping

```
@font-face { font-family: 'MyWebFont';
  src: url('myfont-thin.woff2') format('woff2');
  font-weight: 100; }

@font-face { font-family: 'MyWebFont';
  src: url('myfont-light.woff2') format('woff2');
  font-weight: 300; }

@font-face { font-family: 'MyWebFont';
  src: url('myfont-regular.woff2') format('woff2');
  font-weight: 400; // normal }

@font-face { font-family: 'MyWebFont';
  src: url('myfont-bold.woff2') format('woff2');
  font-weight: 700; // bold }

@font-face { font-family: 'MyWebFont';
  src: url('myfont-black.woff2') format('woff2');
  font-weight: 900; }

@font-face { font-family: 'MyWebFont';
  src: url('myfont-thin-italic.woff2') format('woff2');
  font-weight: 100; font-style: italic; }

@font-face { font-family: 'MyWebFont';
  src: url('myfont-light-italic.woff2') format('woff2');
  font-weight: 300; font-style: italic; }

@font-face { font-family: 'MyWebFont';
  src: url('myfont-italic.woff2') format('woff2');
  font-weight: 400; font-style: italic; }

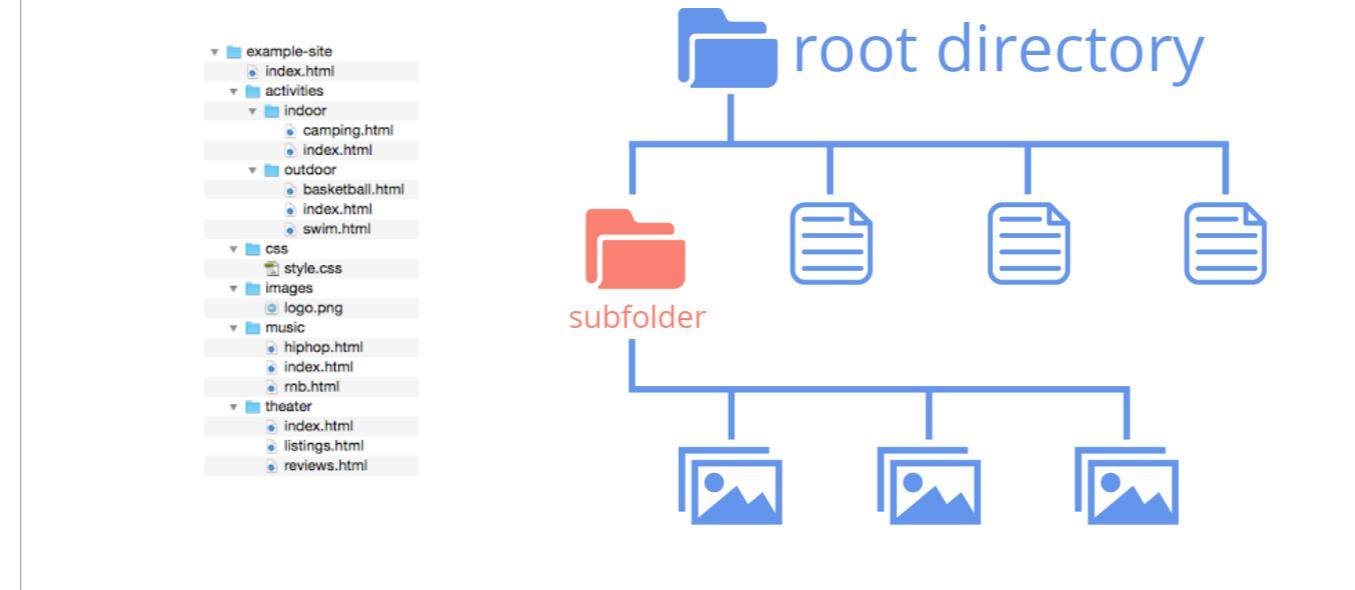
@font-face { font-family: 'MyWebFont';
  src: url('myfont-bold-italic.woff2') format('woff2');
  font-weight: 700; font-style: italic; }

@font-face { font-family: 'MyWebFont';
  src: url('myfont-black-italic.woff2') format('woff2');
  font-weight: 900; font-style: italic; }
```

<https://codepen.io/Klarence/pen/RNwNqpZ>

# File Structure & File Trees

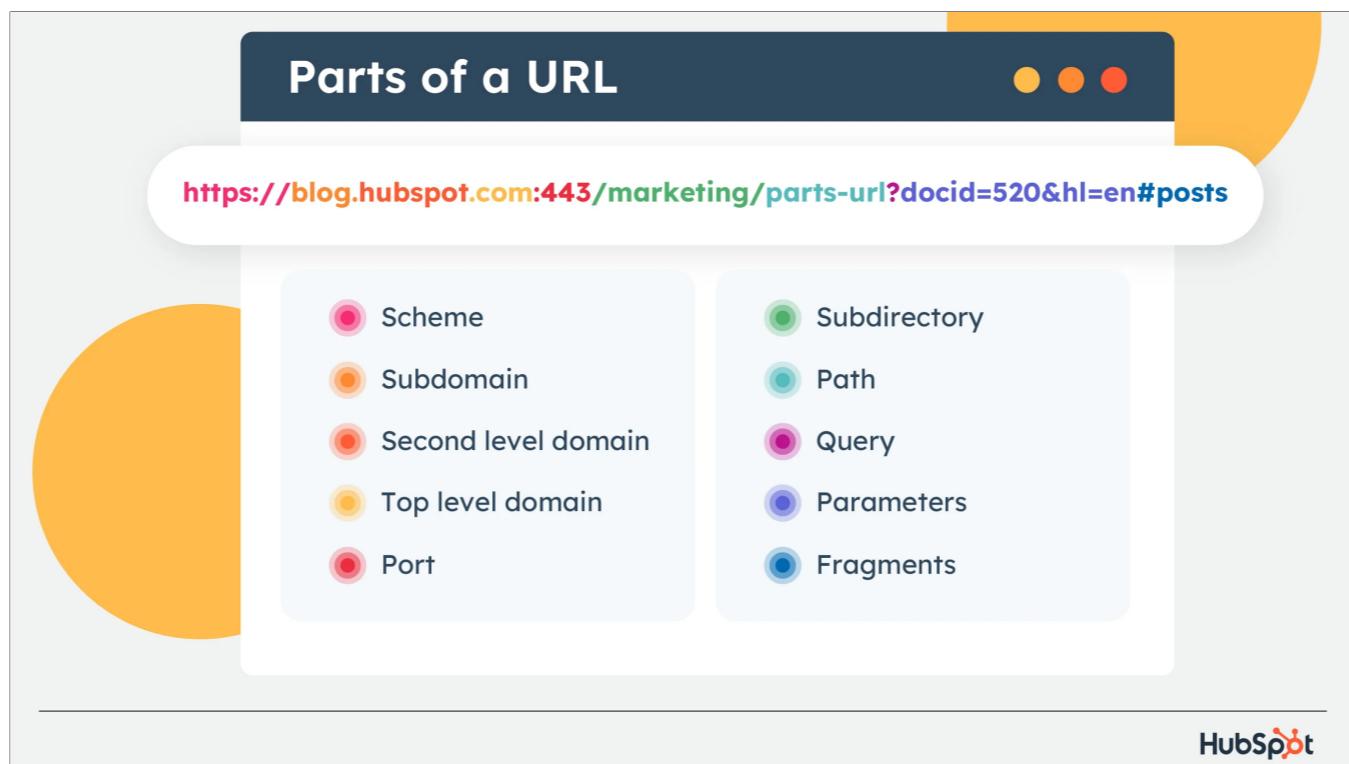
## File Structure & File Trees



You don't need to add the file name to a URL if it is **index.html**.

If a url route ends with a subdirectory instead of a file,  
the browser will look for an index.html in that directory to display.

<https://blog.hubspot.com/marketing/parts-url>



<https://blog.hubspot.com/marketing/parts-url>

# Navigation Styling

<https://codepen.io/Klarence/pen/mydPONN>

<https://htmlandcssguidebook.com/advanced/responsive-nav/>

# Shared CSS

## Shared CSS

You can reference a single CSS file, in multiple HTML files

Show base styles on how multiple html files are referenced.

# Background Images

## Background Images

- background-image: url("");
- Good for background patterns and textures
- Do NOT use for actual content
  - If you must, use ARIA attributes
    - role="img"
    - aria-label="alt text"



Link to Adding Images

<https://www.magicpattern.design/tools/css-backgrounds>

<https://www.transparenttextures.com/>

# Fluid Width

## Fluid Design

Allows content to scale and reflow automatically regardless of device size (viewport).

- Uses percentage-based widths for elements.
  - Best practice on the web is to avoid horizontal scrolling
    - Only setting fluid widths. no heights, unless you want a scrollbar
  - Provides a smooth transition across different screen sizes.
  - Considered a core component of responsive design.

<https://www.youtube.com/watch?v=-vDf7xDVDe0>

While both responsive design and fluid design aim to adapt a website layout to different screen sizes, the key difference is that responsive design uses predefined breakpoints to adjust the layout at specific screen sizes using CSS media queries, while fluid design scales proportionally across all screen sizes without fixed breakpoints, relying primarily on relative units like percentages to adjust content size and placement. - Google AI

## Fluid Width

- Percentage widths
- Using min-width & max-width
  - Not setting width

We'll go over **flex integers** (ratio) and the similar **grid fractional unit** later in future classes.

## Fixed, Responsive, Adaptive & Fluid

### Fixed

- Scrolling / things may get cut off
- panning and zooming on mobile

### Responsive

- Elements Stack, adjust layout on different screen sizes

### Adaptive

- Breakpoints (Media Queries)
- for different Device Sizes (Mobile, Tablet, Laptop, etc.)

### Fluid

- Scalable type and dimensions
- Smooth transition

<https://www.youtube.com/watch?v=-vDf7xDVDe0>

While both responsive design and fluid design aim to adapt a website layout to different screen sizes, the key difference is that responsive design uses predefined breakpoints to adjust the layout at specific screen sizes using CSS media queries, while fluid design scales proportionally across all screen sizes without fixed breakpoints, relying primarily on relative units like percentages to adjust content size and placement. - Google AI

A "fixed" design maintains a static layout regardless of screen size, while a "responsive" design dynamically adjusts to different screen sizes using fluid grids and media queries; a "fluid" layout primarily uses percentage-based widths to scale proportionally with the screen, and an "adaptive" design offers predefined layouts for specific screen sizes, providing more control over how content appears on different devices.

Key Differences:

Fixed Design:

Static layout with set pixel dimensions.

No adjustments based on screen size, may lead to large empty spaces on larger screens.

Simplest to implement but not user-friendly on various devices.

Fluid Design:

Uses percentage-based widths for elements, allowing them to scale proportionally with the screen.

Provides a smooth transition across different screen sizes.

Considered a core component of responsive design.

Responsive Design:

Combines fluid grids, media queries, and flexible image scaling to optimize for various devices.

Dynamically adjusts layout based on screen size using CSS rules.

Considered the most user-friendly approach for diverse devices.

Adaptive Design:

Predefined layouts for specific screen size breakpoints.

Offers more control over how content appears on each device compared to fully responsive design.

May require creating multiple versions of the layout for different screen sizes.

# Fixed, Responsive, Adaptive & Fluid

## Fixed

- Static layout with set pixel dimensions.
- No adjustments based on screen size, may lead to large empty spaces on larger screens.
- Simplest to implement but not user-friendly on various devices.

## Responsive

- Combines fluid grids, media queries, and flexible image scaling to optimize for various devices.
- Dynamically adjusts layout based on screen size using CSS rules.
- Considered the most user-friendly approach for diverse devices.

## Adaptive

- Predefined layouts for specific screen size breakpoints.
- Offers more control over how content appears on each device compared to fully responsive design.
- May require creating multiple versions of the layout for different screen sizes

## Fluid

- Uses percentage-based widths for elements, allowing them to scale proportionally with the screen.
- Provides a smooth transition across different screen sizes.
- Considered a core component of responsive design.

<https://www.youtube.com/watch?v=-vDf7xDVDe0>

While both responsive design and fluid design aim to adapt a website layout to different screen sizes, the key difference is that responsive design uses predefined breakpoints to adjust the layout at specific screen sizes using CSS media queries, while fluid design scales proportionally across all screen sizes without fixed breakpoints, relying primarily on relative units like percentages to adjust content size and placement. - Google AI

A "fixed" design maintains a static layout regardless of screen size, while a "responsive" design dynamically adjusts to different screen sizes using fluid grids and media queries; a "fluid" layout primarily uses percentage-based widths to scale proportionally with the screen, and an "adaptive" design offers predefined layouts for specific screen sizes, providing more control over how content appears on different devices.

# Position

<https://codepen.io/Klarence/pen/VYwGqjp>

## Document Flow

**Document Flow** describes the default way block and inline elements are laid out and affect each other.

- By default, elements have `position: static;`
- Positioning properties don't work (`top`, `right`, `bottom`, `left`)

Z-index also doesn't work on static position.

## Position Basics

### The position Values

- static (default)
- relative
- absolute
- fixed
- sticky

### Properties for Positioning

- top, right, bottom, left
- Doesn't work with static

# Position

	Static (default)	Relative	Absolute	Fixed	Sticky
<b>Doc Flow</b>	Normal Flow	Normal Flow Leaves space	Out of Flow Removes Space	Out of Flow Removes Space	Normal Flow Leaves space
<b>Position Offset</b> (Top/Bottom & Left/Right)	-	Relative to itself	Relative to 1st parent container w/ position: relative;	Relative to the browser window	Same as Relative but becomes Sticky to remain on page

Position Offset does not work in static.

# Forms

<https://codepen.io/Klarence/pen/pvoOyWR?editors=1010>

## Forms

Forms are used to gather input data from users of websites. Like contact forms and checking out of an e-commerce site.

```
<form action="./submit-form.js" method="post">  
// ... All the form inputs and data  
</form>
```

All forms should have an action.

The action should trigger a method/open a file or reference the page that the user is on.

The default action is #, which means the same page the user is on.

The method attribute's default value is get.

## Forms - Input

```
<input type="text" name="firstName"/>
```

Common type values

- text
- email
- url
- number
- password
- range
- file
- submit

The type attribute differentiates what type of input to display.

## Forms - Input

```
<input type="text" name="firstName"/>
```

The name attribute is used to associate the data for an input for a form.

The name attribute is used by the form to send the data in the input to the backend server.

## Forms - Input Labels

All form inputs should have a label.

```
<label>
  First Name
  <input type="text" name="firstName"/>
</label>
```

```
<label for="lastName">Last Name</label>
<input type="text" name="lastName" id="lastName"/>
```

## Forms - Input Labels

All form inputs should have a label.

```
<label>
  First Name
  <input type="text" name="firstName"/>
</label>
```

```
<label for="lastName">Last Name</label>
<input type="text" name="lastName" id="lastName"/>
```

1. Implicit Association
2. Explicit Association with for and id attributes matching. (recommended, in case folks refactor the code)

Labels are good for Accessibility and Usability.

Some forms try to save space by using placeholders instead of labels. This is considered a bad practice as users often forget what was there after the placeholder disappears. This is often even more problematic for those with cognitive disabilities.

Clicking on a label for a checkbox or a radio button will interactive as if you clicked on the actual form element the large is associated with. This is helpful because off the size of those interactive elements.

## Forms - Dropdown

```
<label>Select Your Airport  
<select name="airport">  
    <option value="LAX">Los Angeles, California</option>  
    <option value="SJC">San Jose, California</option>  
    <option value="SJO">San Jose, Costa Rica</option>  
</select>  
</label>
```

Select Your Airport: Choose one ▾

The select element displays the dropdown.

The option elements nested inside

The form will submit the value instead of the text in the selected option.

The <select> element can now be customized with CSS, March 24, 2025

<https://developer.chrome.com/blog/a-customizable-select>

## Forms - Dropdown

```
<label>Select Your Airport  
<select name="airport">  
    <option disabled hidden selected>Choose one</option>  
    <option value="LAX">Los Angeles, California</option>  
    <option value="SJC">San Jose, California</option>  
    <option value="SJO">San Jose, Costa Rica</option>  
</select>  
</label>
```

Select Your Airport: Choose one ▾

Select Your Airport: ✓ Los Angeles, California  
San Jose, California  
San Jose, Costa Rica

Note: hidden attribute doesn't work in Safari

<https://htmlandcssguidebook.com/html/forms/#showing-a-grayed-out-first-option>

## Forms - Textarea

```
<label>Tell us about yourself:  
  <textarea name="selfDesc"></textarea>  
</label>
```

The textarea element is useful for when a user will enter multiple lines of text.

## Submitting Forms

```
<input type="submit" value="Submit" />  
<button type="submit">Submit</button>
```

The first `<button>` tag (without a specified type) will implicitly submit the form.  
Also keep in mind the ENTER key can submit a form.

## Clearing Form Values

```
<input type="reset" value="Reset" />  
<button type="reset">Reset</button>
```

Note: many users find this annoying.

I think a lot of people accidentally clear the form.

Think about the user experience and whether clearing the form is useful feature for your use case.

## Common Attributes

- <input type="text" required>
- <input type="text" disabled>
- <input type="text" placeholder="Example or Hint Text">

required and disabled are boolean attributes

placeholders - Do not replace labels. Use them to provide concise, helpful hints or examples without replacing labels

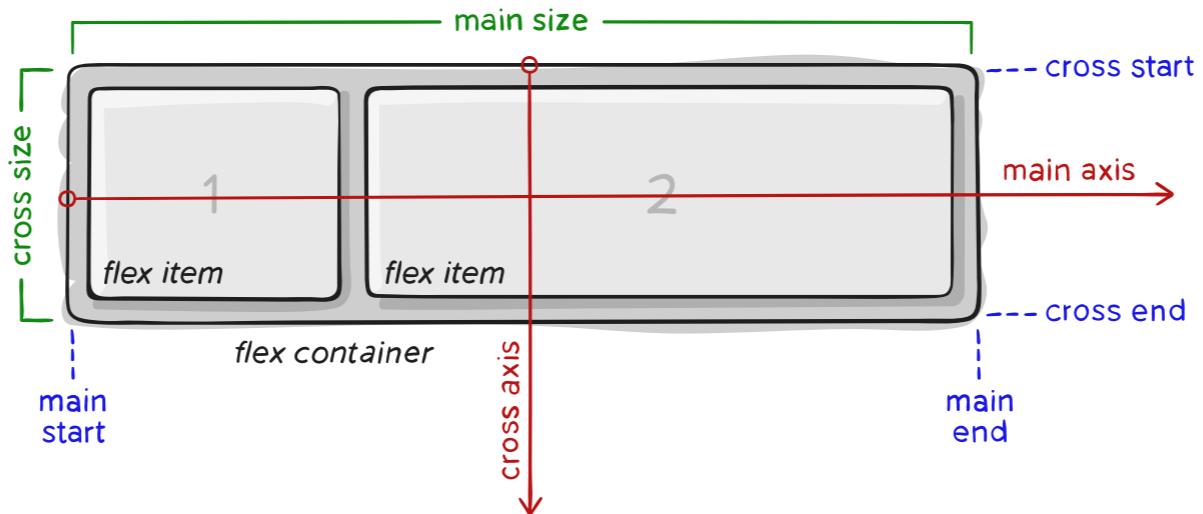
# Flexbox

<https://codepen.io/bwhitener/pen/MWeYLbV>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

<https://www.joshwcomeau.com/css/interactive-guide-to-flexbox/>

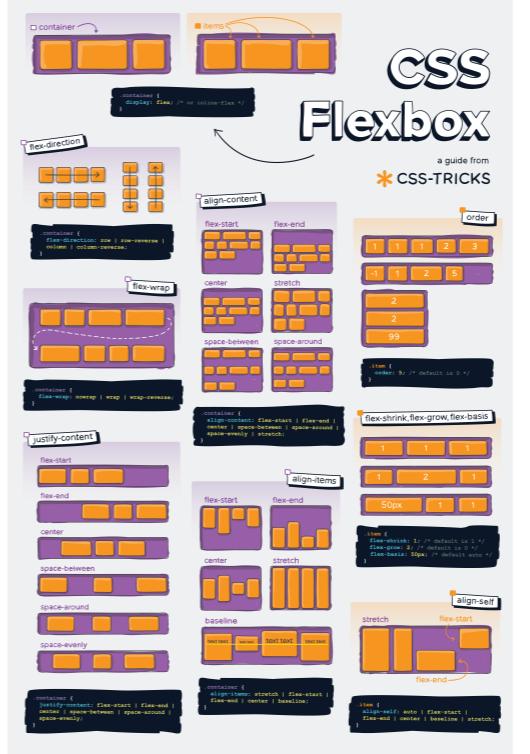
## Flexbox



By default the main axis is horizontal and left-to-right.

By default the cross axis is vertical and top-to-bottom.

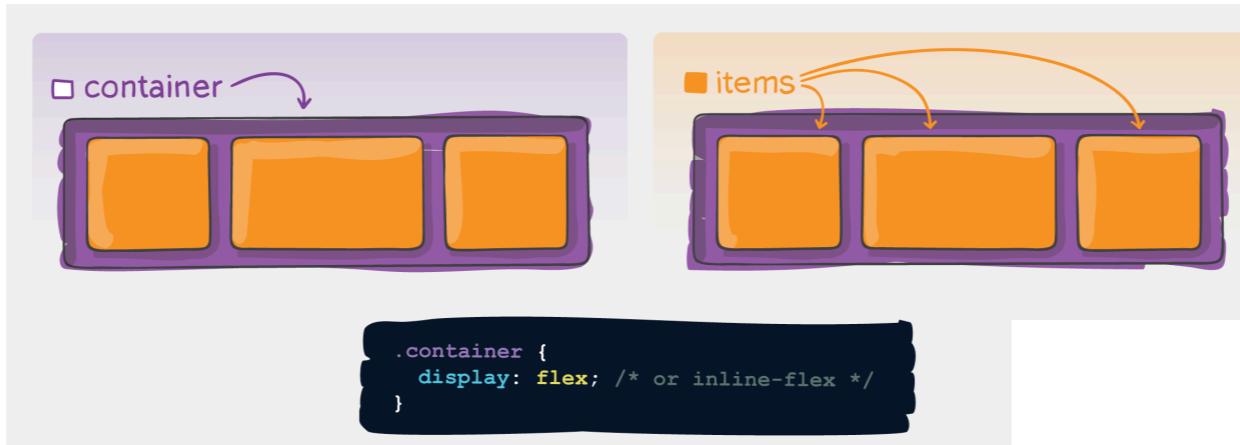
# Flexbox



<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

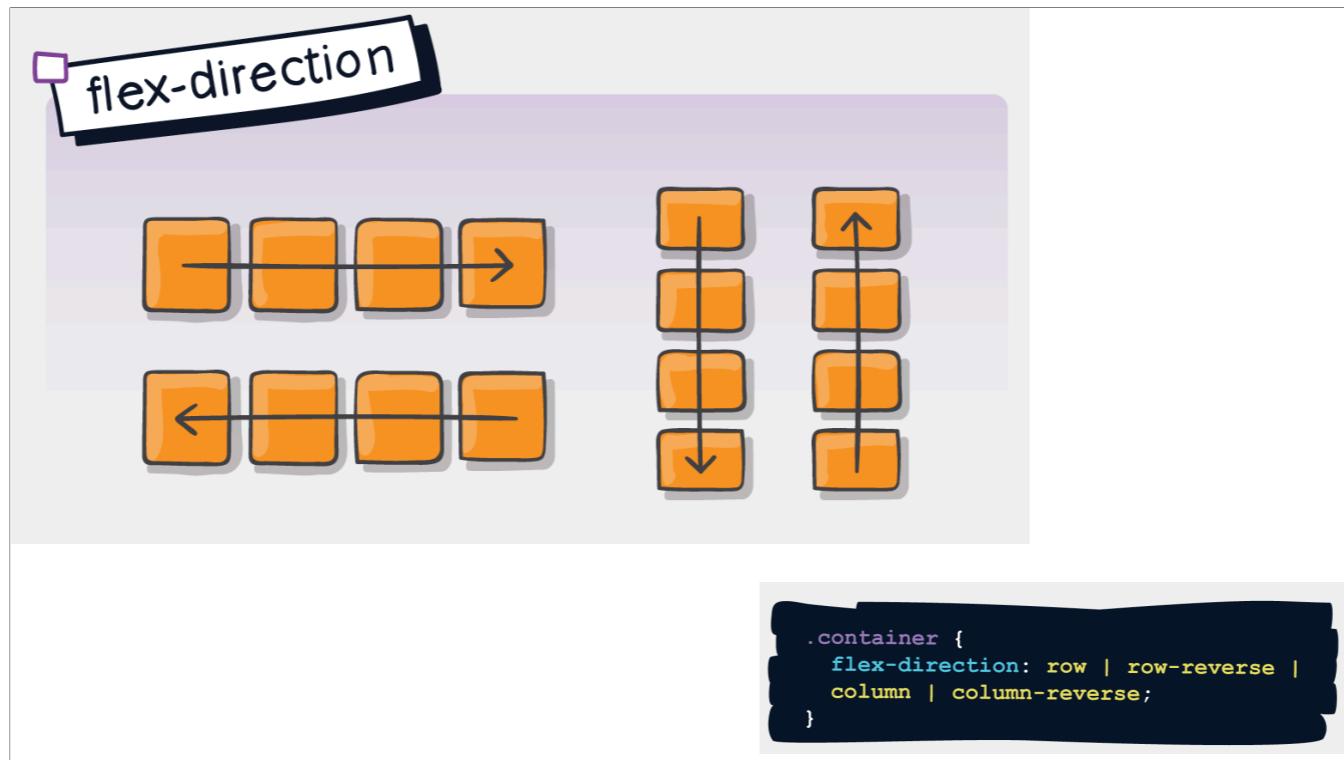
<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

# Flexbox



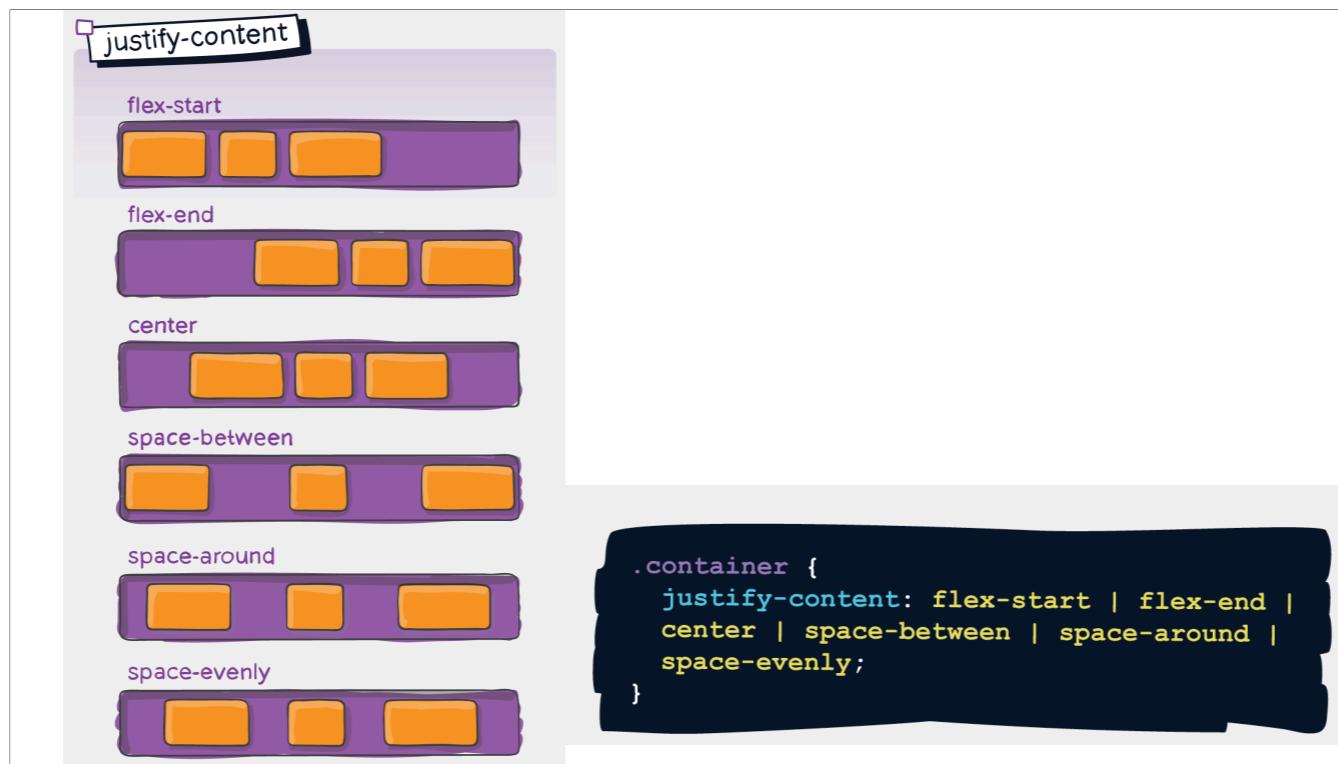
<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>



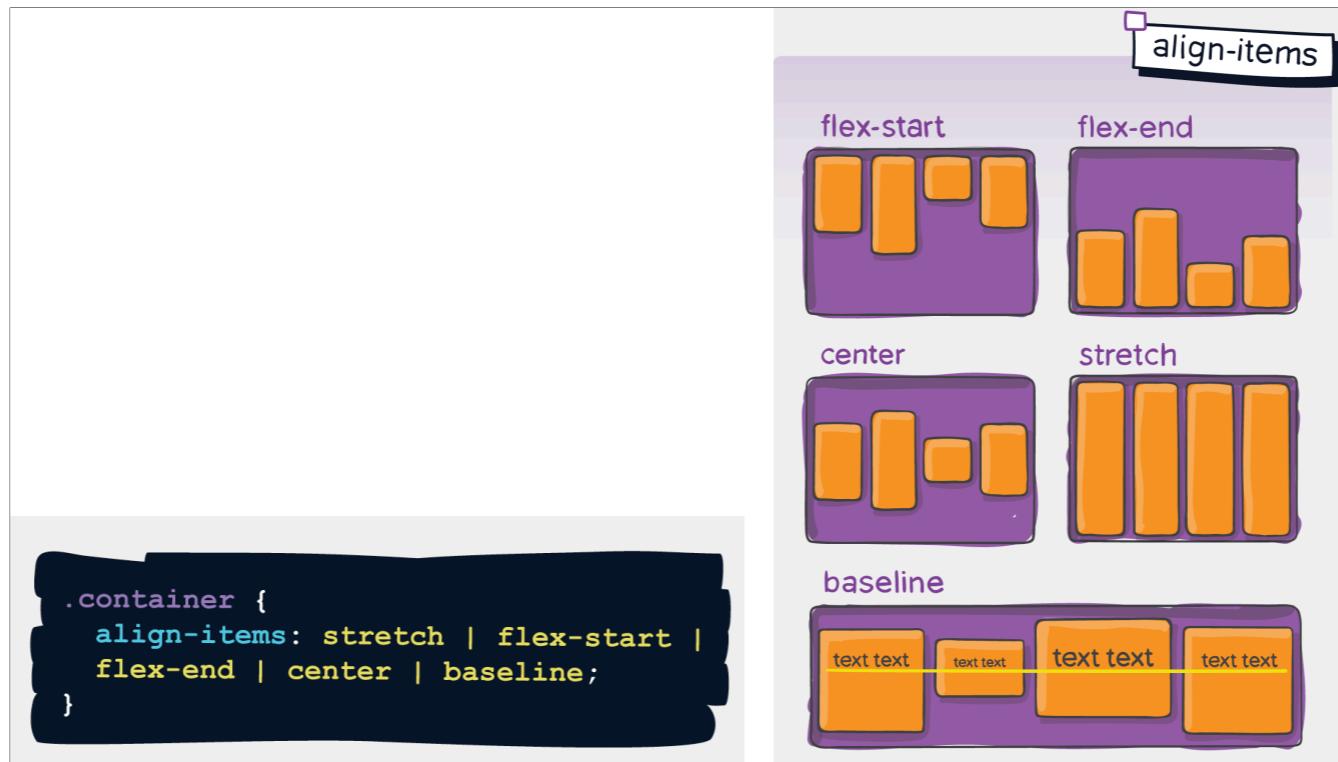
<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>



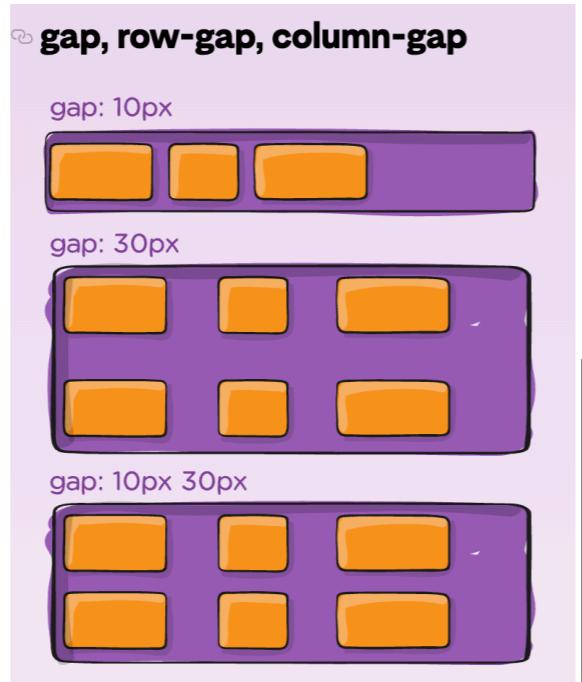
<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>



<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

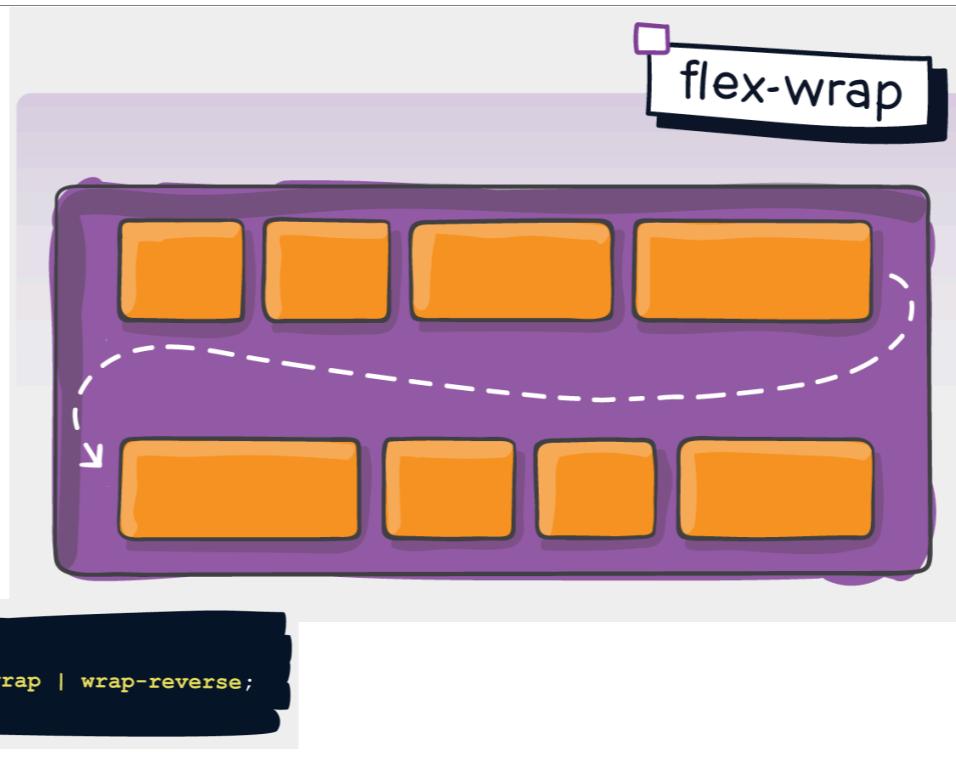


```
.container {  
  display: flex;  
  ...  
  gap: 10px;  
  gap: 10px 20px; /* row-gap column gap */  
  row-gap: 10px;  
  column-gap: 20px;  
}
```

<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

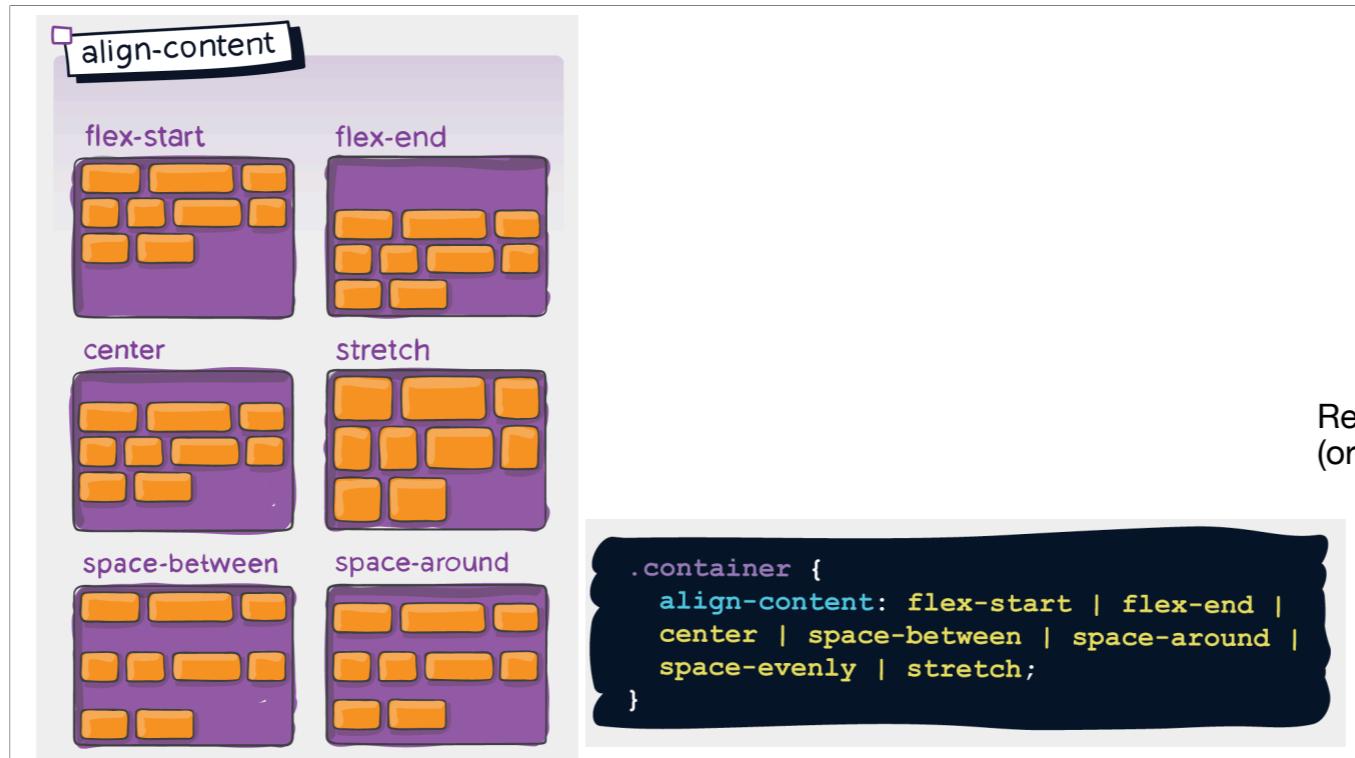
<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

# Flexbox



<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>



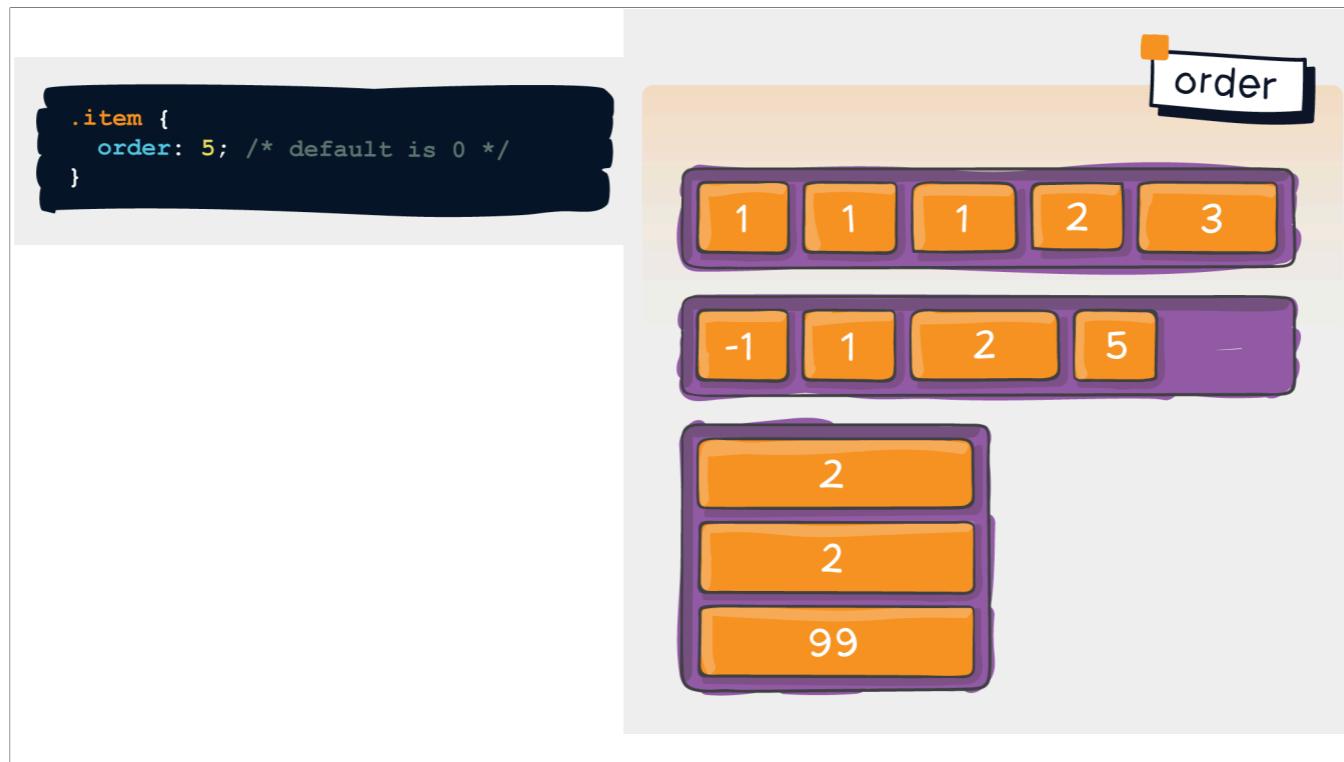
<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>  
<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

align-content - must have `flex-wrap: wrap/wrap-reverse;`

## Flexbox

### Flex-item Attributes

Some less common attributes are set on the flex-item.



<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

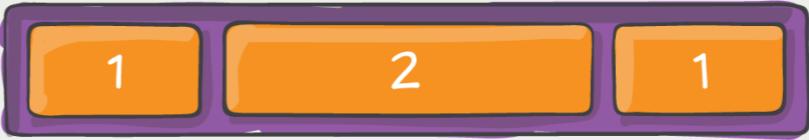
# Flexbox

## flex-shrink, flex-grow, flex-basis

```
.item {  
  flex-shrink: 1; /* default is 1 */  
  flex-grow: 2; /* default is 0 */  
  flex-basis: 50px; /* default auto */  
}
```



```
.item {  
  flex-shrink: 1; /* default is 1 */  
  flex-grow: 2; /* default is 0 */  
  flex-basis: 50px; /* default auto */  
}
```



```
.item {  
  flex-shrink: 1; /* default is 1 */  
  flex-grow: 2; /* default is 0 */  
  flex-basis: 50px; /* default auto */  
}
```

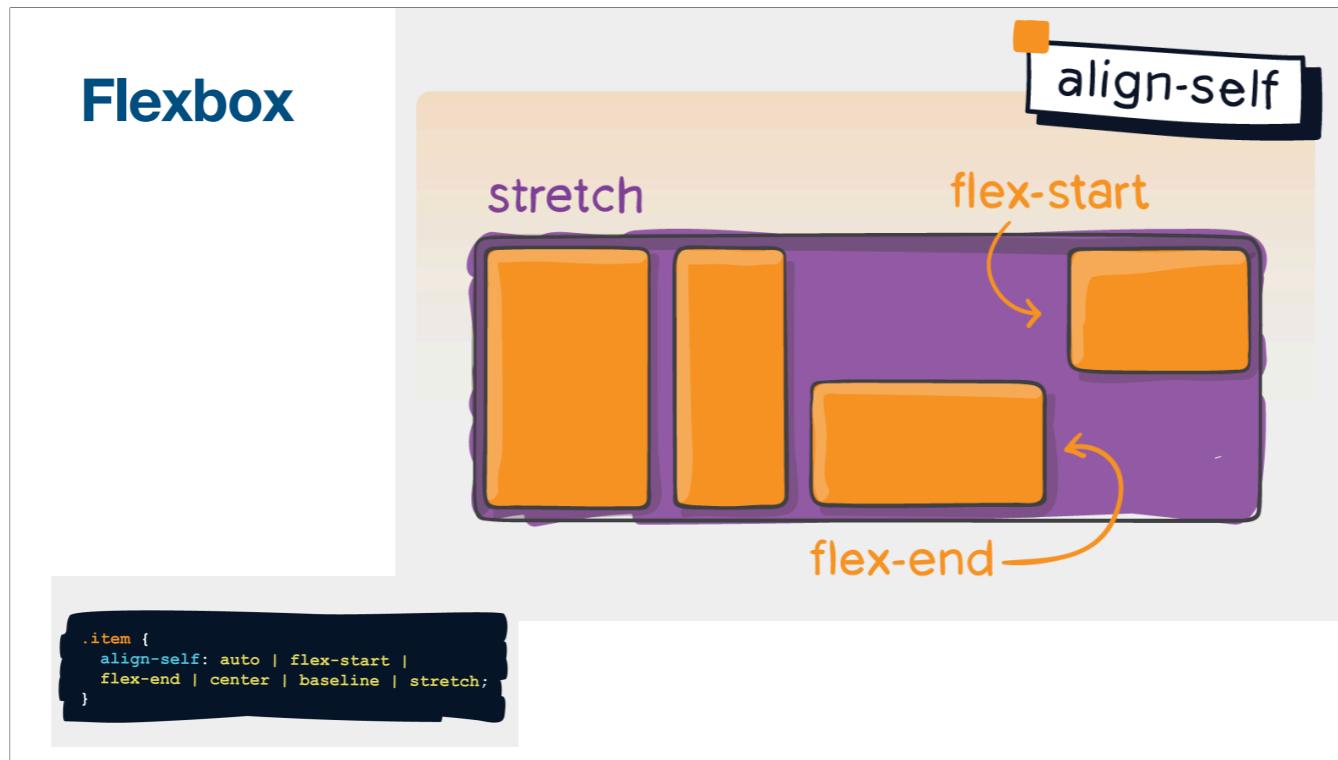


<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

Flex-basis - pretty much the same as width/height (depending on flex-direction is row or column)

## Flexbox



<https://css-tricks.com/wp-content/uploads/2022/02/css-flexbox-poster.png>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

# Annotate a Website

# CSS Grid

<https://codepen.io/Klarence/pen/jEEEMvp>

<https://css-tricks.com/snippets/css/complete-guide-grid/#aa-important-css-grid-terminology>

<https://www.joshwcomeau.com/css/interactive-guide-to-grid/>

<https://cssgridgarden.com/>

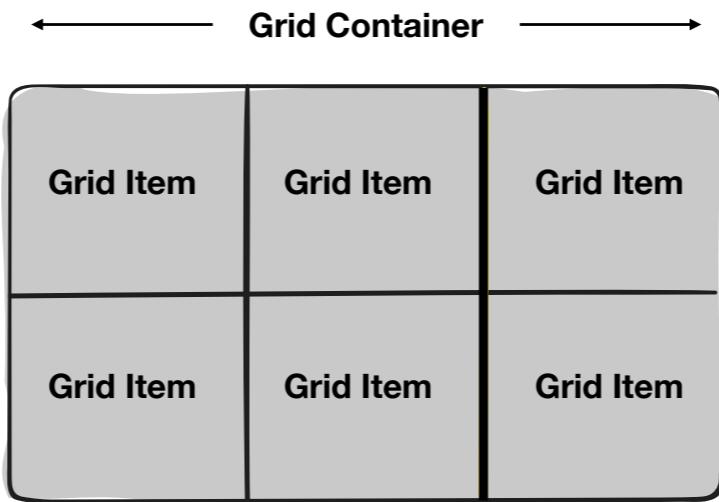
Table Layout	Grid Frameworks	CSS Grid
<pre>&lt;table&gt; &lt;tr&gt;   &lt;th colspan="4"&gt;     Header   &lt;/th&gt; &lt;/tr&gt; &lt;tr&gt;   &lt;td colspan="2"&gt;     Main   &lt;/td&gt;   &lt;td&gt;&lt;/td&gt;   &lt;td&gt;Sidebar&lt;/td&gt; &lt;/tr&gt; &lt;tr colspan="4"&gt;   &lt;td&gt;Footer&lt;/td&gt; &lt;/tr&gt; &lt;/table&gt;</pre>	<pre>&lt;div class="container"&gt;   &lt;div class="row"&gt;     &lt;header class="col-12"&gt;       Header     &lt;/header&gt;   &lt;/div&gt;   &lt;div class="row"&gt;     &lt;main class="col-6"&gt;       Main     &lt;/main&gt;     &lt;aside class="col-3 push-3"&gt;       Sidebar     &lt;/aside&gt;   &lt;/div&gt;   &lt;div class="row"&gt;     &lt;footer class="col-12"&gt;       Footer     &lt;/footer&gt;   &lt;/div&gt; &lt;/div&gt;</pre>	<pre>&lt;div class="container"&gt;   &lt;header class="header"&gt;Header&lt;/header&gt;   &lt;main class="main"&gt;Main&lt;/main&gt;   &lt;aside class="sidebar"&gt;Sidebar&lt;/aside&gt;   &lt;footer class="footer"&gt;Footer&lt;/footer&gt; &lt;/div&gt;</pre>

When before layout techniques were created, folks used the table element to create layouts out of necessity.

Popular Frameworks like [Bootstrap](#) and [Foundation](#) used premade CSS classes to set elements (or their containers) to a grid.

# CSS Grid

## Terminology



## CSS Grid

A 2 dimensional layout system that can handle both columns and rows.

### Grid Container (Parent)

The container element to which most grid properties should be applied (including `display: grid;`). This should be the direct parent to the elements that will be arranged into grid cells.

### Grid Item (Children)

The direct child elements to the grid parent. These are the elements that will be arranged to fit in the grid cells.

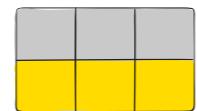
# CSS Grid

## Terminology



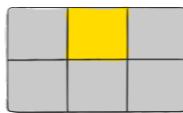
### Grid Lines

The lines that defined the edges of grid columns or rows. These are numbered from left to right and top to bottom, respectively. They are important for positioning individual grid children into specific grid cells.



### Grid Cells

The spaces defined by the intersections of grid columns and grid rows, similar to the cells in a spreadsheet. The grid children are arranged into these.



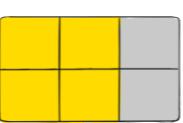
### Grid Track

The space between two adjacent grid lines. You can think of them as the columns or rows of the grid. Here's the grid track between the second and third-row grid lines.



### Grid Area

The total space surrounded by four grid lines. A grid area may be composed of any number of grid cells. Here's the grid area between *row* grid lines 1 and 3, and *column* grid lines 1 and 3

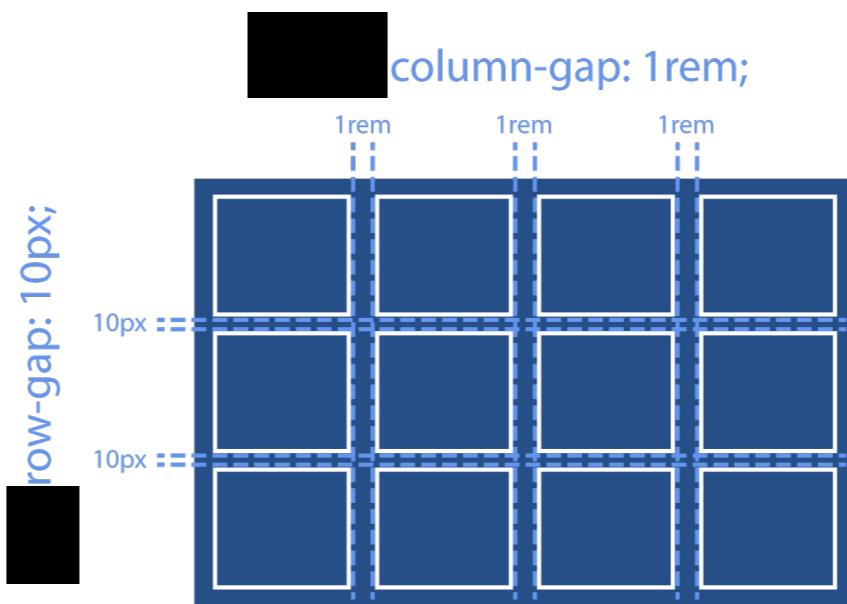


# CSS Grid

## Gap

### Gap

Specifies the size of the grid lines. You can think of it like setting the width of the gutters between the columns & rows.



gap (formerly grid-gap)

gap: 1rem 10px;

Is a shorthand for row-gap and column-gap

If no row-gap is specified, it's set to the same value as column-gap

NOTE: grid-row-gap and grid-column-gap should still work, but the syntax has been deprecated.

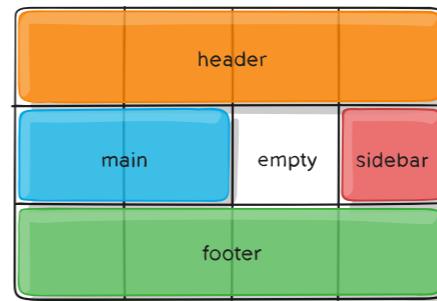
# CSS Grid

## Template

The **grid-template-columns** and **grid-template-rows** properties allow us to define any number of columns or rows, respectively, into which the **grid-child** elements will be placed.

Sometimes you will **grid-template-rows** set to **auto**, or you may not see it at all. This is because rows don't need to be set explicitly and if you have additional elements after your last column, a new row will be created implicitly.

```
.parent-container {  
    width: 1000px;  
    display: grid;  
    grid-template-columns: 1fr 250px 15.625rem 25%;  
    grid-template-rows: 1fr 250px 33%;  
}
```



You can set different unit values for each column in **grid-template-columns** (same for **grid-template-rows**).

Grid comes with fractional (fr) units, which uses up a fraction of the remaining grid space.

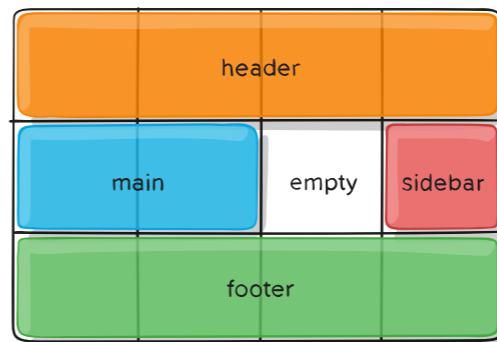
# CSS Grid

## Placement

The **grid-column** and **grid-row** properties assigns each grid item to a place on the grid by specifying the grid line numbers.

The long hand versions are **grid-row-start**, **grid-row-end**, **grid-column-start**, and **grid-column-end**.

```
.header {  
  grid-column: 1 / 5;  
  grid-row: 1 / 2;  
}  
  
.main {  
  grid-column-start: 1;  
  grid-column-end: 3;  
  grid-row-start: 2;  
  grid-row-end: 3;  
}
```



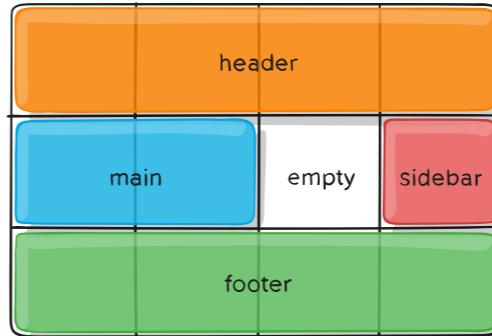
You can set different unit values for each column in `grid-template-columns` (same for `grid-template-rows`)

# CSS Grid

## Assigning Items

Another way of assigning children items to grid-cells is a named **grid-area**, created with **grid-template-areas**

```
.parent-container {  
    display: grid;  
    grid-template-columns: 1fr 250px 15.625rem 25%;  
    grid-template-rows: 1fr 250px 33%;  
    grid-template-areas:  
        "header header header header"  
        "main main . sidebar"  
        "footer footer footer footer"  
}  
.header {  
    grid-area: header;  
}  
.main {  
    grid-area: main;  
}
```



The dot (period, ".") is used to indicate an empty grid-cell.  
If you are a visual person this will be a lot easier for you.

# Grid Item Alignment

<https://css-tricks.com/snippets/css/complete-guide-grid/#aa-important-css-grid-terminology>

<https://www.joshwcomeau.com/css/interactive-guide-to-grid/>

## CSS Grid

### Alignment

By default, columns and rows stretch to fill the grid container.

However, if the grid container is larger than the width of all the columns you can align the columns using **justify-content**.

If it is larger than the height of all the rows than you can use **align-content**.

Likewise you can align a grid item inside of a grid-area using;  
**justify-items** (for horizontal alignment - column)  
**align-items** (for vertical alignment - row).

If you want to align an individual grid-item, you can use **justify-self** and **align-self**.

A grid-area can be a set of multiple columns/rows (or it could be a single column/row).

# CSS Grid

## Alignment

<b>Justify</b>	Columns
<b>Align</b>	Rows
<b>Content</b>	Grid Container (Parent)
<b>Items</b>	Grid Items (Children)

If a property contains a certain word, it is associated with a part of the grid.

## CSS Grid

### Alignment

By default, columns and rows stretch to fill the grid container.

However, if the grid container is larger than the width of all the columns you can align the columns using **justify-content**.

If it is larger than the height of all the rows than you can use **align-content**.

Likewise you can align a grid item inside of a grid-area using;  
**justify-items** (for horizontal alignment - column)  
**align-items** (for vertical alignment - row).

If you want to align an individual grid-item, you can use **justify-self** and **align-self**.

A grid-area can be a set of multiple columns/rows (or it could be a single column/row).

## CSS Grid

### Alignment

By default, columns and rows stretch to fill the grid container.

However, if the grid container is larger than the width of all the columns you can align the columns using **justify-content**.

If it is larger than the height of all the rows than you can use **align-content**.

Likewise you can align a grid item inside of a grid-area using;  
**justify-items** (for horizontal alignment - column)  
**align-items** (for vertical alignment - row).

If you want to align an individual grid-item, you can use **justify-self** and **align-self**.

A grid-area can be a set of multiple columns/rows (or it could be a single column/row).

# Media Queries & Mobile-first Design

## Mobile First

Layouts for mobile are often much less complex than those for larger sizes.

**Mobile First** - start with CSS for the smallest possible size and add complexity with media queries for larger and larger viewports

<https://htmlandcssguidebook.com/css/responsive/>

Mobile First is considered a best practice.

## Media Queries

Conditional CSS.

A media query is basically an if statement for CSS.

"If the criteria here are met, apply the CSS contained within."

```
@media media type and (condition: breakpoint) {  
    // CSS rules  
}
```

<https://htmlandcssguidebook.com/css/responsive/>

Mobile First is considered a best practice.

# Media Queries

## Media Types

- **screen**—for computer screens, tablets and, smart-phones
- **print**—for printers
- **all**—for all media types
- **speech**—for screen readers that “read” the page out loud

```
@media screen and (min-width: 1024px) {  
    // CSS rules for desktop and larger  
}
```

<https://htmlandcssguidebook.com/css/responsive/>

Mobile First is considered a best practice.

If you see "only screen", this was used to hide the styles from older browsers. <https://stackoverflow.com/a/14168210/4267591>

# Media Queries

## Common Breakpoints Sizes

- 320px – 480px: Mobile devices
- 481px – 768px: iPads/Tablets
- 769px – 1024px: Small screens/laptops
- 1025px – 1200px: Desktops/large screens
- 1201px & more — Extra large screens, TV
- 1440px & more

```
@media screen and (min-width: 481px) {  
    // CSS rules for tablets and larger  
}  
  
@media screen and (min-width: 481px) and  
        (max-width: 768) {  
    // CSS rules for tablets only  
}
```

# Accessibility for Everyone

[https://docs.google.com/presentation/d/1Lh-T7f9F6yoDxtRGqxvhaK\\_WuOfP9xIzr8m1qzMNu4A/edit?usp=sharing](https://docs.google.com/presentation/d/1Lh-T7f9F6yoDxtRGqxvhaK_WuOfP9xIzr8m1qzMNu4A/edit?usp=sharing)

# Designing for Accessibility

[https://docs.google.com/presentation/d/1Eytx-Msk5uxlS9tDdWeXZcki1\\_hWhUG9-V6XYP7-KT8/edit?usp=sharing](https://docs.google.com/presentation/d/1Eytx-Msk5uxlS9tDdWeXZcki1_hWhUG9-V6XYP7-KT8/edit?usp=sharing)

# Screen reader practice

# Lighthouse

**End**

## Common Links

- [Link to boilerplate / skeleton code](#)