Web Design and Programming

Week 3

9 May 2024

Instructor: Dr. Peeraya Sripian

Course schedule

Week	Date	Topic
1	4/18	Intro to WWW, Intro to HTML
2	4/25	CSS Fundamental
	5/2	Holiday (GW)
3	5/9	CSS and Bootstrap
4	5/16	Work on midterm project
5	**COIL 22 MAY 18:00-19:30 (counted as 1 class, replacing 23 May)**	
6	5/30	Midterm project presentation week
7	6/6	PHP fundamentals + Installation XAMPP
8	6/13	PHP fundamentals 2
9	6/20	mySQL fundamentals
10	6/27	Assessing MySQL using PHP, MVC pattern + Intro of Final project
11	7/4	Cookies, sessions, and authentication + Proposal of final project
12	7/11	Javascript and PHP validation
13	7/18	Final project development
14	7/25	Final project presentation

This week’s learning objective

- CSS basics
  - Styles rule structure
  - Adding styles to the document
  - Cascade property
- CSS Selectors
  - ID, Class, Universal
- CSS Box Model
  - Floating and positioning
  - Flexbox
  - Grid

Markup Tips

- It is important to mark up content **semantically**, in a way that accurately describes the content's meaning or function.
- This ensures your content is accessible in the **widest** range of viewing environments:
  - Desktop and mobile browsers
  - Assistive reading devices
  - Search engine indexers

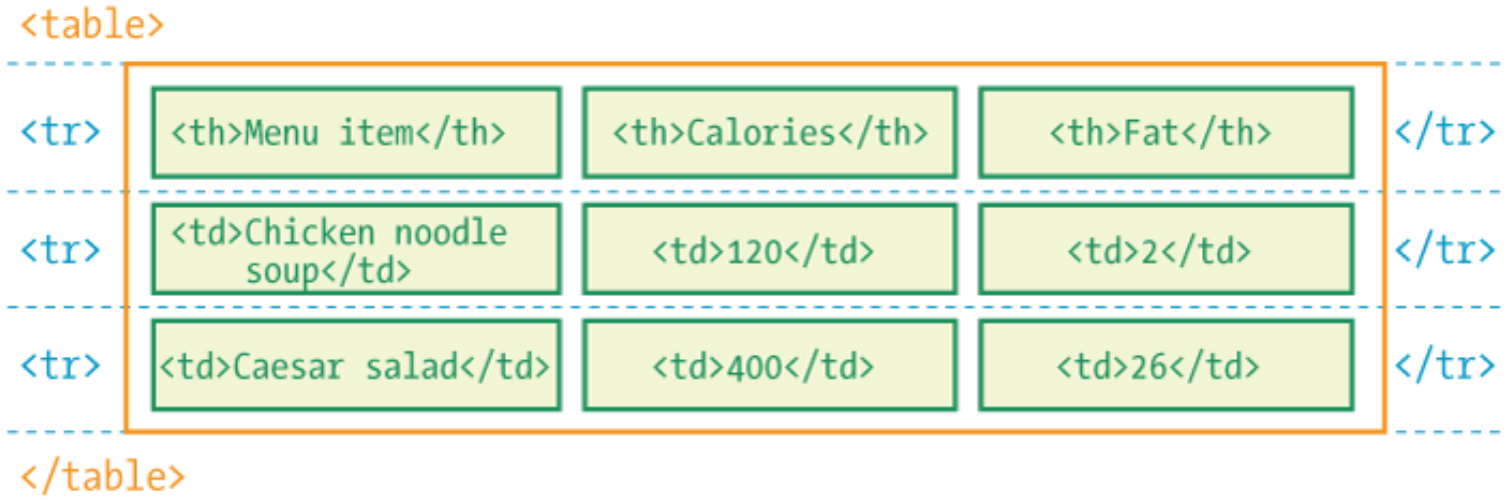
Page Organizing Elements

HTML5 introduced elements that give meaning to the typical sections of a web page:

- `main`
- `header`
- `footer`
- `section`
- `article`
- `aside`
- `nav`

HTML Table Structure (cont'd.)

- How it looks using markup (`table`, `tr`, `th`, and `td`):



NOTE: Columns are implied by the number of cells in each row (not created with column elements).


CSS
Introduction

CSS

- W3C standard for defining the presentation of documents written in HTML

- A separate language with its own syntax.

- HTML – document structure
- CSS – Presentation (how it is delivered to the user, whether on a computer screen, cell phone, printed on paper, etc)

The Benefits of CSS

- Precise type and **layout control**
- **Less work:** Change look of the whole site with one edit
- **Accessibility:** Markup stays semantic
- **Flexibility:** The same HTML markup can be made to appear in dramatically different ways

Style Separate from Structure

- These pages have the exact same HTML source but different style sheets:

(csszengarden.com)

How Style Sheets Work

1. Start with a marked up document (like HTML, but could be another XML markup language).
2. Write styles for how you want elements to look using CSS syntax.
3. Attach the styles to the document (there are a number of ways).
4. The browser uses your instructions when rendering the elements.

Style Rules

Each rule *selects* an element and *declares* how it should display.

```
h1 { color: green; }
```

```
<h1>This is H1</h1>
```

This rule selects all **`h1`** elements and declares that they should be green.

This is H1

```
strong { color: red; font-style: italic; }
```

```
<p>I wrote up a streamlined adaptation of his recipe that requires <strong>(strong) much</strong> less time and serves 6 people instead of <em>five</em>times that amount.</p>
```

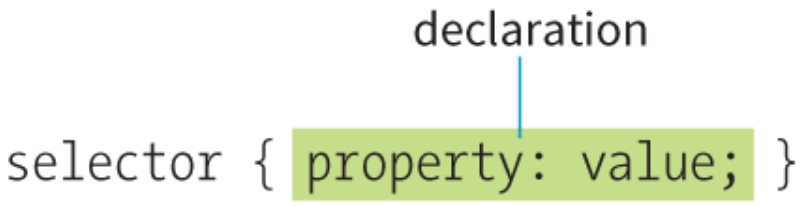
This rule selects all `strong` inline elements and declares that they should be red and in an italic font.

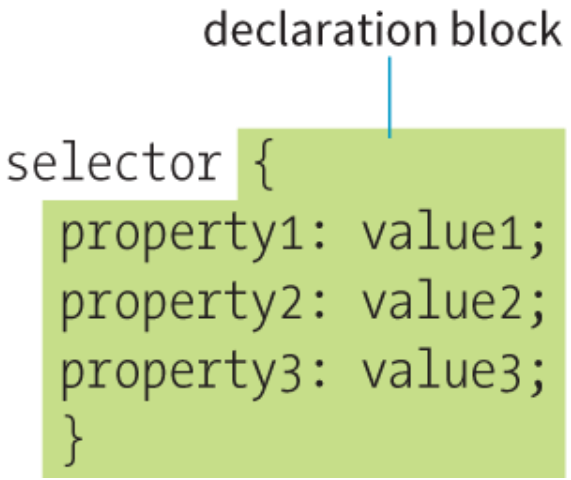
I wrote up a streamlined adaptation of his recipe that requires ***(strong) much*** less time and serves 6 people instead of *five*times that amount.

Style Rule Structure

- A style rule is made up of a selector a declaration.

- The declaration is one or more property / value pairs.





Selectors

- There are many types of selectors. Here are just two examples:

```
p {property: value;}
```

Element type selector: Selects all elements of this type (`p`) in the document.

```
#intro {property: value}
```

ID selector (indicated by the `#` symbol) selects by ID value. In the example, an element with an **id of "intro"** would be selected.

Declarations

- The **declaration** is made up of a **property/value pair** contained in curly brackets **{ }:**
  - **selector {property: value;}**

- **Example**

```
h2 { color: red;
     font-size: 2em;
     margin-left: 30px;
     opacity: .5;
   }
```

This is H2

Declarations (cont’d)

- End each declaration with a semicolon to keep it separate from the next declaration.
- White space is ignored, so you can stack declarations to make them easier to read.
- **Properties** are defined in the CSS specifications.
- **Values** are dependent on the type of property:
  - Measurements
  - Color values
  - Keywords
  - More

CSS Comments

```
/* comment goes here */
```

- Content between `/*` and `*/` will be ignored by the browser.
- Useful for leaving notes or section label in the style sheet.
- Can be used within rules to temporarily hide style declarations in the design process.

```
<style>
/*This is how you write comment in CSS
<style> tag or .css file*/
  h1
  {
    color:green;
  }

  h2{
    color:red;
    font-size: 2em;
    margin-left: 30px;
    opacity: .5;
  }
  strong{
    color:red; font-style:italic;
  }
</style>
```

Adding Styles to the Document

- There are three ways to attach a style sheet to a document:

- **External style sheets**

  A separate, text-only *.css* file associated with the document with the `link` element or `@import` rule

- **Embedded style sheets**

  Styles are listed in the `head` of the HTML document in the `style` element.

- **Inline styles**

  Properties and values are added to an individual element with the `style` attribute.

External Style Sheets

- The style rules are saved in a separate text-only *.css* file and attached via **`link`** or **`@import`**.

Via **`link`** element in HTML:

```
<head>
  <title>Titles are require</title>
  <link rel="stylesheet"
href="/path/example.css">
</head>
```

Via **`@import`** rule in a style sheet:

```
<head>
  <title>Titles are required</title>
  <style>
    @import url("/path/example.css");
    p {font-face: Verdana;}
  </style>
</head>
```

Embedded Style Sheets

- Embedded style sheets are placed in the **`head`** of the document via the **`style`** element:

```
<head>
  <title>Titles are required</title>
  <style>
    /* style rules go here */
  </style>
</head>
```

Inline Styles

- Apply a style declaration to a single element with the **`style`** *attribute*:

```
<p style="font-size: large;">Paragraph text...</p>
```

To add multiple properties, separate them with semicolons:

```
<h3 style="color: red; margin-top: 30px;">Intro</h3>
```

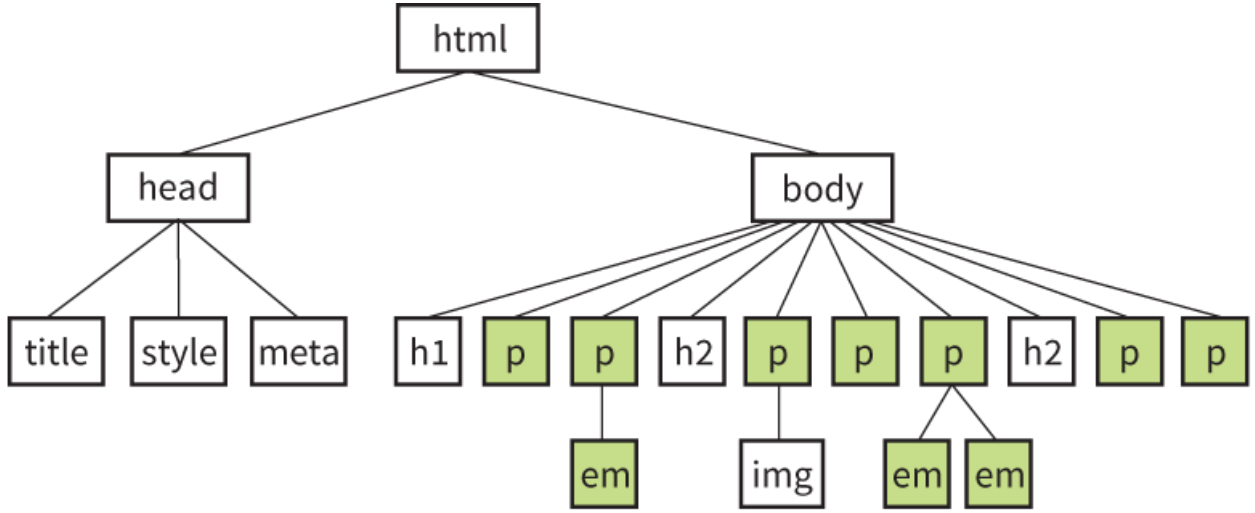
Document Structure

- Documents have an implicit structure.
- We give certain relationships names, as if they're a family:
- All the elements contained in a given element are its descendants.
- An element that is directly contained within another element is the child of that element.
- The containing element is the parent of the contained element.
- Two elements with the same parent are siblings.

Inheritance

- Many properties applied to elements are passed down to the elements they contain. This is called inheritance.

- For example, applying a sans-serif font to a `p` element causes the `em` element it contains to be sans-serif as well:




```
p {font-size: large; font-family: sans-serif;}
```

The cascade property

It is called “cascade” for a reason!

The Cascade

- The **cascade** refers to the system for resolving conflicts when several styles apply to the same element.
- Style information is passed down (it “cascades” down) until overwritten by a style rule with more **weight**.
- Weight is considered based on:
    - **Priority** of style rule source
    - **Specificity** of the selector
    - **Rule order**

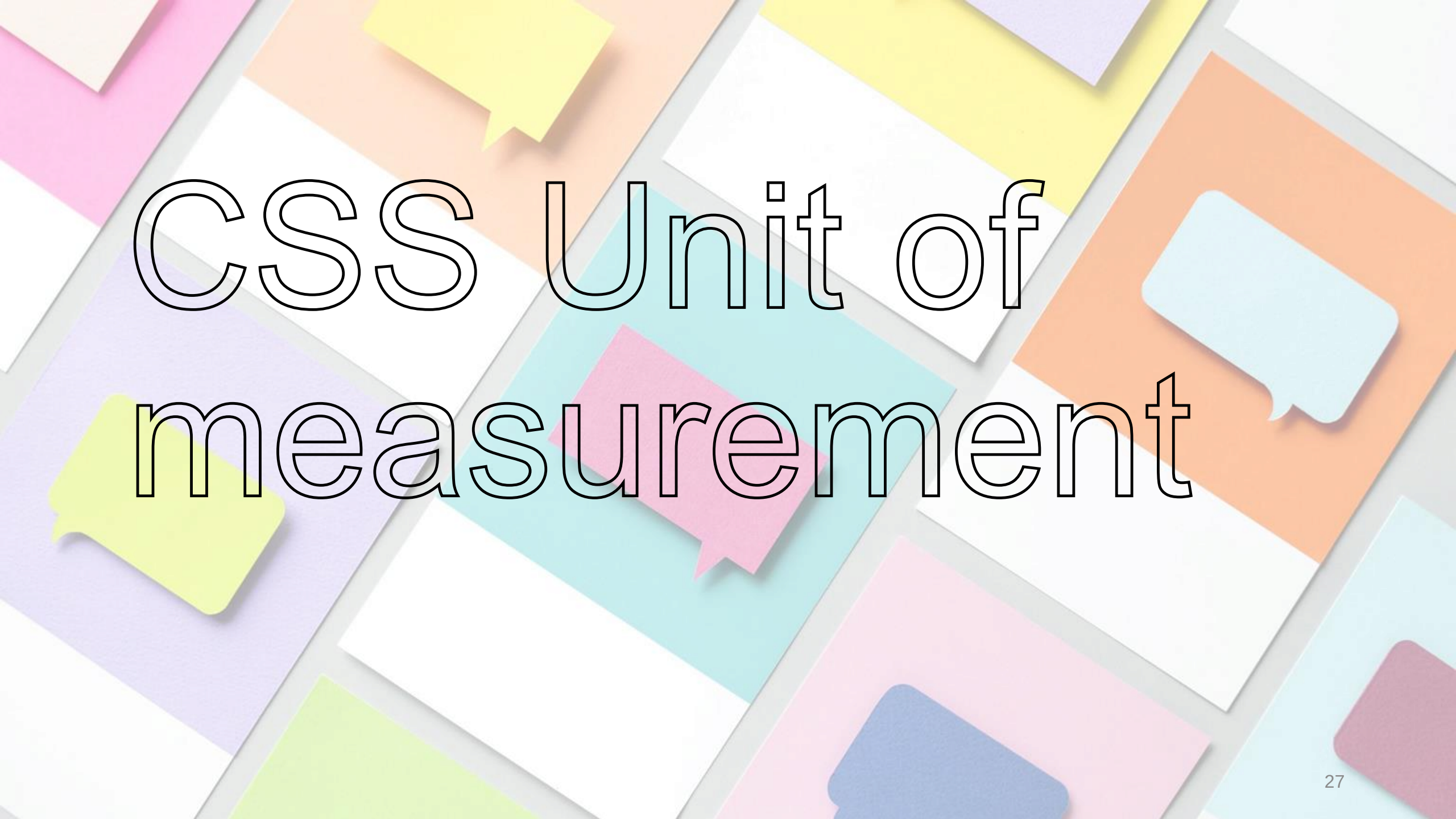
The Cascade: Rule Order

- When two rules have equal weight, rule order is used. **Whichever rule appears last "wins."**

```
<style>
  p {color: red;}
  p {color: blue;}
  p {color: green;}
</style>
```

In this example, paragraphs would be green.

- Styles may come in from external style sheets, embedded style rules, and inline styles. The style rule that gets parsed last (the one closest to the content) will apply.


CSS Unit of measurement

Full details on CSS Values and Units Module
http://www.w3.org/TR/css3-values/

CSS Units of Measurement

- CSS provides a variety ways to specify measurements:

Absolute units

- Have predefined meanings or real-world equivalents

Relative units

- Based on the size of something else, such as the default text size or the size of the parent element

Percentages

- Calculated relative to another value, such as the size of the parent element

Absolute Units

- With the exception of pixels, absolute units are not appropriate for web design:

`px`	pixel
`in`	inches
`mm`	millimeters
`cm`	centimeters
`q`	1/4 millimeter
`pt`	points (1/72 inch)
`pc`	pica (1 pica = 12 points = 1/6 inch)

Relative Units

Good for most web measurements

- Relative units are based on the size of something else:

`em`	a unit equal to the current font size
`ex`	x-height, equal to the height of a lowercase *x*
`rem`	root em, equal to the font size of the `html` element
`ch`	zero width, equal to the width of a zero (0)
`vw`	viewport width unit (equal to 1/100 of viewport width)
`vh`	viewport height unit (1/100 of viewport height)
`vmin`	viewport minimum unit (value of `vh` or `vw`, whichever is smaller)
`vmax`	viewport maximum unit (value of `vh` or `vw`, whichever is larger)

RELATIVE UNITS

The rem Unit

- The **`rem`** **(root em)** unit is based on the font size of the root **`html`** element, whatever that happens to be.

- **Default** in modern browsers: Root font size is 16 pixels, so a **`rem`** **= a 16-pixel unit,** **`10rem`** **= 160 px (for default)**

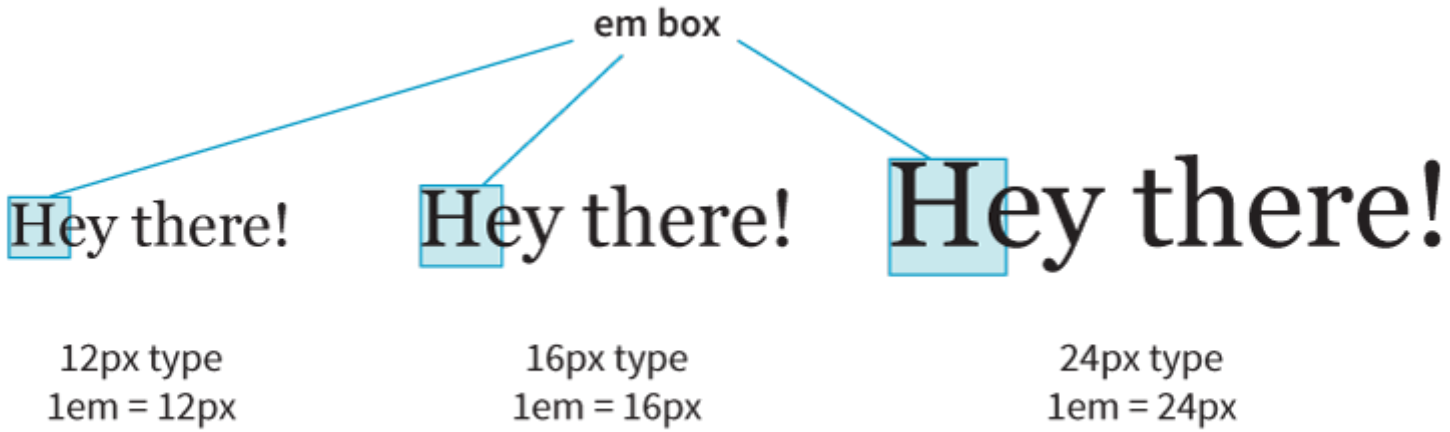
- If the root font size of the document changes, so does the size of a rem (and that's good for keeping elements proportional).

- EX: if a user changes the base font size (in his browser) to 24 px, **`10rem`** would be 240px

RELATIVE UNITS

The em Unit

- The `em` unit is traditionally based on the width of a capital letter *M* in the font.
- When the font size is 16 pixels,1em = 16 pixels, 2em = 32 pixels, and so on.



NOTE: Because they're based on the font size of the current element, the size of an em may not be consistent across a page.

RELATIVE UNITS

Viewport Percentage Lengths (vw/vh)

- Viewport width (**vw**) and viewport height (**vh**) units are relative to the size of the viewport (browser window):
  - **vh** = 1/100th width of viewport
  - **vh** = 1/100th height of viewport

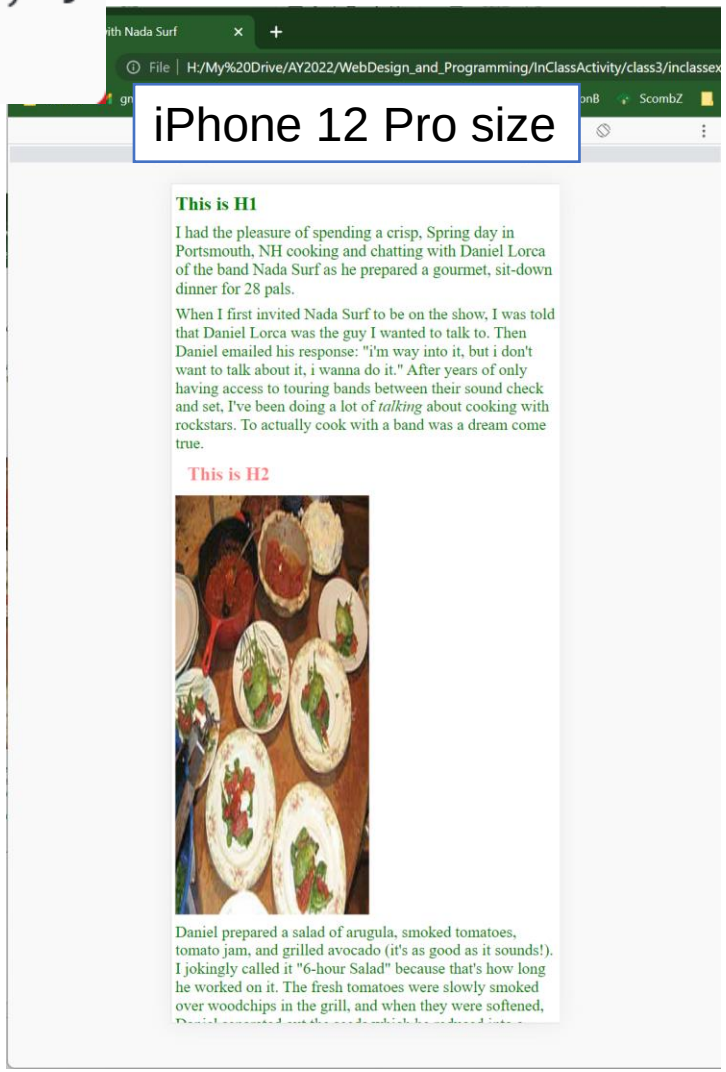
- They’re useful for making an element fill the viewport or a specified percentage of it. This image will be 50% the width and height of the viewport:

```
img { width: 50vw; height: 50vh; }
```

```
img { width: 50vw; height: 50vh; }
```

Regular browser in PC screen

iPhone 12 Pro size



Formatting text

Designing Text

- Styling text on the web is tricky because you don’t have control over how the text displays for the user:
  - They may not have the font you specify.
  - They may have their text set larger or smaller than you designed it.

- Best practices allow for flexibility in text specification.

Typesetting Terminology

- A **typeface** is a set of characters with a single design (example: `Garamond`).

- A **font** is a particular variation of the typeface with a specific weight, slant, or ornamentation (example: `Garamond Bold Italic`).

- In traditional metal type, each size was a separate font (example: `12-point Garamond Bold Italic`).

- On a computer, fonts are generally stored in individual font files.

CSS Basic Font Properties

- CSS font properties deal with specifying the shapes of the characters themselves:
  - `font-family`
  - `font-size`
  - `font-weight`
  - `font-style`
  - `font-variant`
  - `font` (a shorthand that includes settings for all of the above)

See more detail in fullreference file

Text Decoration Tips

- If you turn off underlines under links, be sure there is another visual cue to compensate.

- Underlining text that is not a link may be misleading. Consider italics instead.

- Don’t use `blink`. Browsers don’t support it anyway.

More Selector Types

- Descendent selectors
- ID selectors
- Class selectors
- Universal selector

Descendent Selectors

- A **descendent selector** targets elements contained in another element.
- It's a kind of **contextual selector** (it selects based on relationship to another element).
- It's indicated in **a list separated by a character space**.

```
ol a {font-weight: bold;}
```

(only the links (**a**) in ordered lists (**ol**) would be bold)

```
h1 em {color: red;}
```

(only emphasized text in h1s would be red)

A space between element names means that the 2nd element must be contained within the first

Descendent Selectors (cont'd)

- They can appear as part of a grouped selector:

```
h1 em, h2 em, h3 em {color: red;}
```

(only emphasized text in **h1**, **h2**, and **h3** elements)

- They can be several layers deep:

```
ol a em {font-variant: small-caps;}
```

(only emphasized text in links in ordered lists)

ID Selectors

- **ID selectors** (indicated by a # symbol) target elements based on the value of their ID attributes:

```
<li id="primary">Primary color t-shirt</li>
```

- To target just that item:

```
li#primary {color: olive;}
```

- To omit the element name:

```
#primary {color: olive;}
```

- It can be used as part of a compound or contextual selector:

```
#intro a { text-decoration: none;}
```

Class Selectors

- **Class selectors** (indicated by a **.** symbol) select elements based on the value of their class attributes:

```
p.special { color: orange;}
```

(All paragraphs with the class name "special" would be orange.)

- To target *all* element types that share a class name, omit the element name in the selector:

```
.hilight { background-color: yellow;}
```

(All elements with the class “hilight” would have a yellow background.)

Universal Selector

- The **universal element selector** (*****) matches any element, like a wildcard in programming languages:

```
* {border: 1px solid gray;}
```

(puts a 1-pixel gray border around every element in the document)

- Can be used as part of contextual selectors:

```
#intro * {border: 1px solid gray;}
```

(selects all elements contained within an element with the ID **`intro`**)

Specificity Basics

- **Specificity** refers to a system for sorting out which selectors have more weight when resolving style rule conflicts.
- **More specific selectors have more weight.**
- In simplified terms, it works like this:
  - **Inline styles** with the `style` attribute are more specific than (and will override…)
  - **ID selectors**, which are more specific than (and will override…)
  - **Class selectors**, which are more specific than (and will override…)
  - **Individual element selectors**

Calculating Specificity

- There is a system used to calculate specificity. Start by drawing three boxes:

[ ] [ ] [ ]

For each style rule:

1. Count the **IDs** in the selector and put that number in the first box.
2. Count the **class** and pseudo-class selectors and put the number in the second box.
3. Count the **element** names and put the number in the third box

[ID] [class] [elements]

4. **The first box that is not a tie determines which selector wins.**

Calculating Specificity (cont'd)

Example:

The second one has a class selector and the first one doesn't, therefore the second one is more specific and has more weight.

The lime color applies to `h1`s when they have the class name "special."

Using Specificity

Use specificity strategically to take advantage of overrides:

```
p { line-height: 1.2em; }                    [0] [0] [1]
```

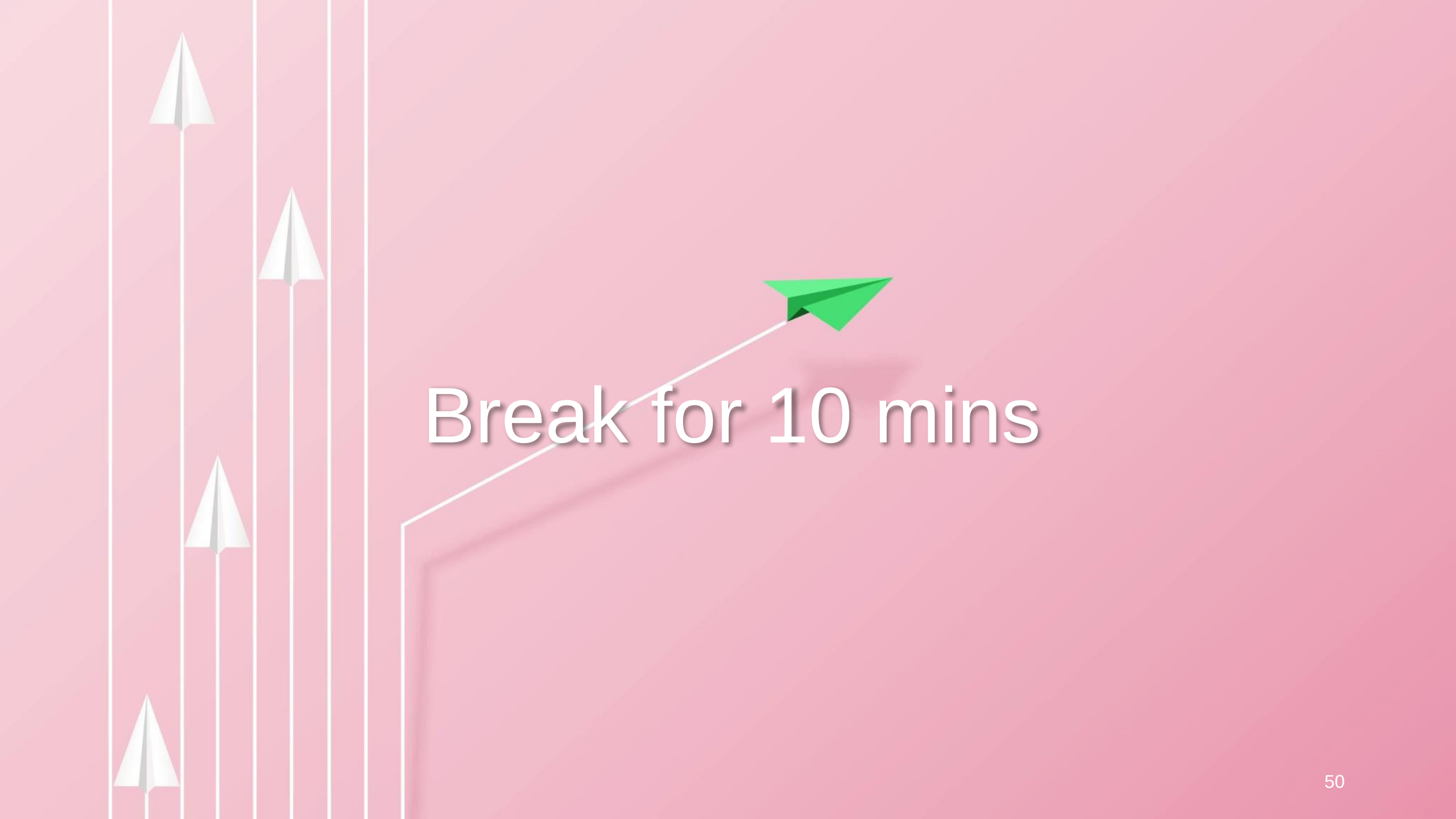
(sets the line-height for all paragraphs)

```
blockquote p { line-height: 1em; }           [0] [0] [2]
```

(more specific selector changes line-height when the paragraph is in a blockquote)

```
p.intro { line-height: 2em; }                [0] [1] [1]
```

(paragraphs with the class “intro” have a line-height of 2em, even when they’re in a blockquote. A class name in the selector has more weight than two element names.)


Break for 10 mins

Colors and Background

Ways to specify color

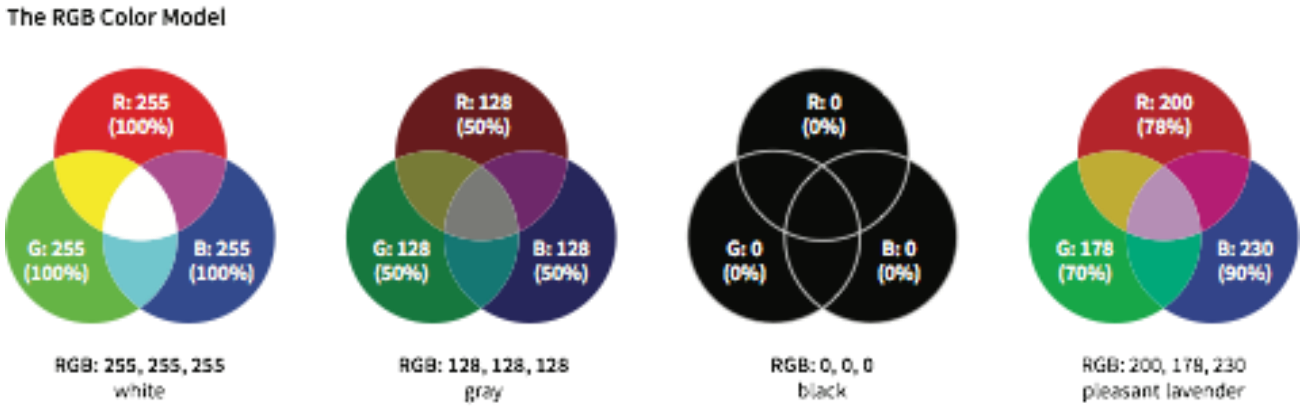
- By color names

```
h1 { color: red; }
h2 { color: darkviolet; }
body { background-color: papayawhip; }
```

- By numeric values
  - RGB (Red Green Blue)
  - RGBa + Alpha (opacity-transparency)
  - HSL (Hue, Saturation, Lightness)
  - HSLa + Alpha (opacity-transparency)

RGB Color

- The **RGB color model** mixes color with red, green, and blue light.
- Each channel can have 256 shades, for millions of color options.

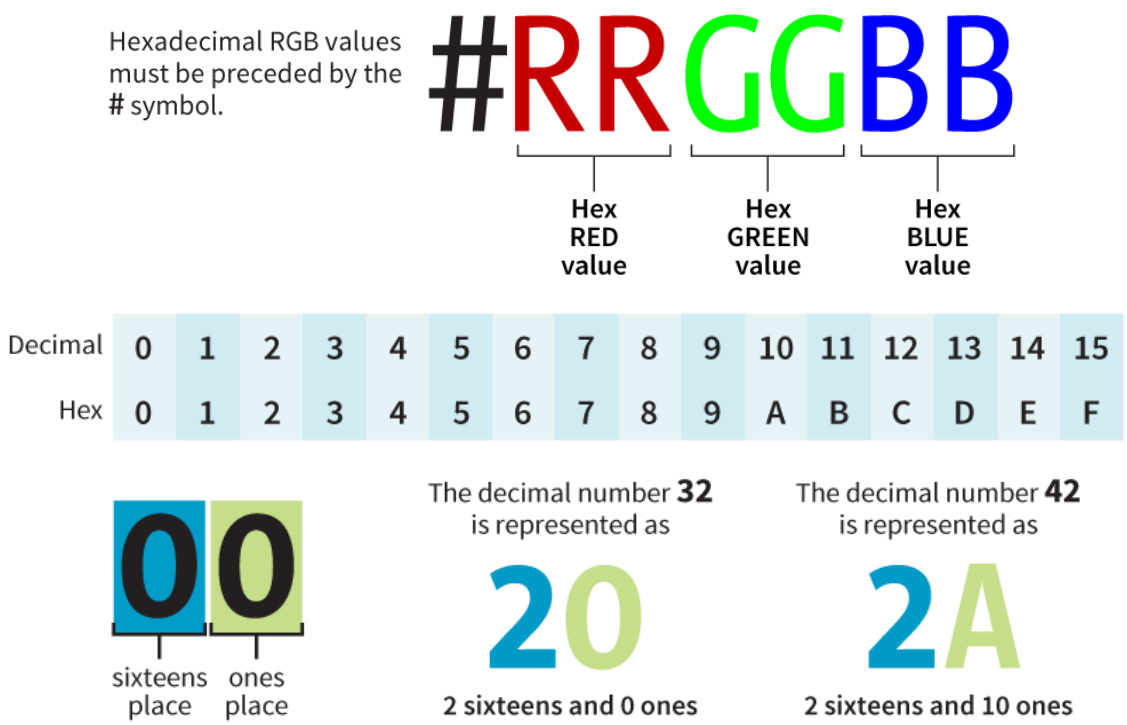


4 Styles of RGB specification

- RGB values (0 to 255):
  `rgb(200,178,230)`
- Percentage values:
  `rgb(78%,70%,90%)`
- Hexadecimal values:
  `#C8B2E6`
- Condensed hexadecimal values (for double-digits only):
  `#F06` is the same as `#FF0066`

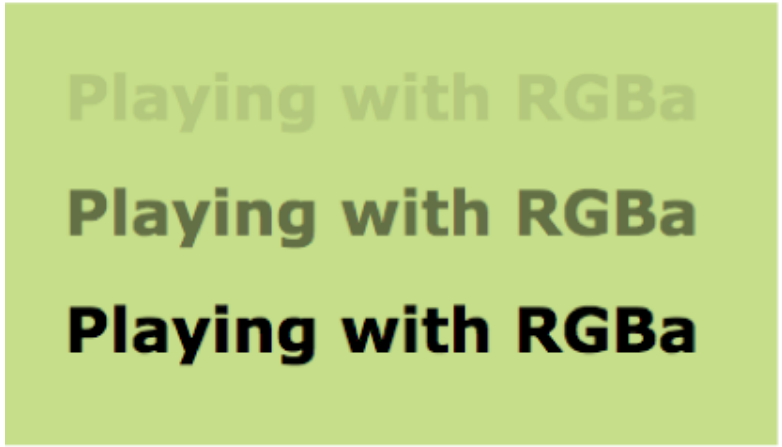
Hexadecimal RGB Values

Red, green, and blue values converted to **hexadecimal** and preceded by the # symbol.



RGBa Color

- **RGB + an alpha channel** for transparency
- The first three values are RGB. The fourth is the transparency level from 0 (transparent) to 1 (opaque).



HSL and HSLa

- Colors described by values for **hue** (°), **saturation** (%), and **luminosity** (%):

```
hsl(180,50%,75%)
```

- Hue specifies the position on a color wheel (in degrees) that has red at 0°, green at 120°, and blue at 240°.
- HSL is less commonly used than RGB, but some find it more intuitive.
- **HSLa** adds an alpha value for transparency.

Opacity

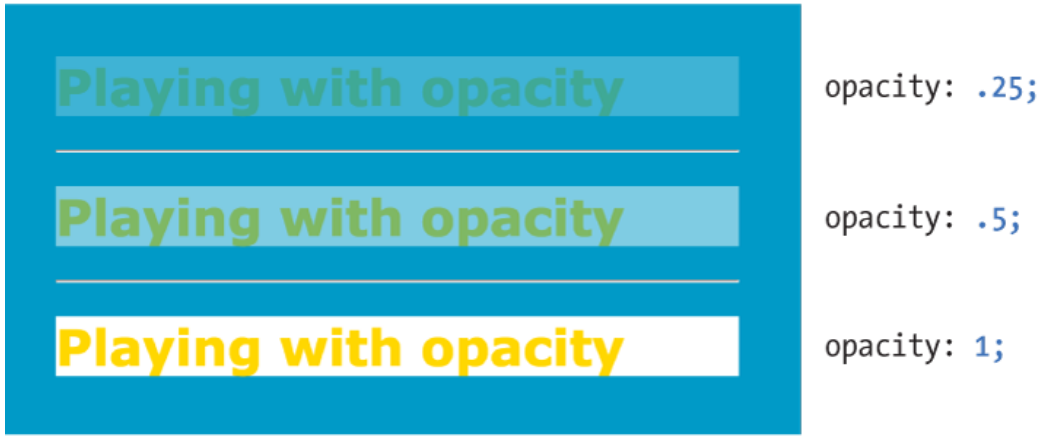
`opacity`

Values: *`number`* (0 to 1)

Example:

```
h1 {color: gold; background: white; opacity: .25;}
```

Specifies the transparency level from 0 (transparent) to 1 (opaque):



Shorthand background Property

`background`

Values:
`background-color background-image background-repeat background-attachment background-position background-clip background-origin background-size`

- Specifies all background properties in one declaration

```
background: white url(star.png) no-repeat top center fixed;
```

- Properties are optional and may appear in any order
- Properties not represented reset to their defaults—be careful it doesn't overwrite previous background settings.

Multiple Background Images

- You can place more than one background image in a single image (separated by commas):

```
body {
  background:
    url(image1.png) left top no-repeat,
    url(image2.png) center center no-repeat,
    url(image3.png) right bottom no-repeat;
  }
```

See more detail in fullreference file

More Selector Types

Pseudo-class selectors

Pseudo-element selectors

Attribute selectors

Pseudo-Class Selectors

Treat elements in a certain state as belonging to the same class

Link Pseudo-classes

`:link`	Applies style to unvisited (unclicked) links
`:visited`	Applies style to visited links

User Action Pseudo-classes

`:focus`	Applies when element is selected for input
`:hover`	Applies when the mouse pointer is over the element
`:active`	Applies when the element (such as a link or button) is in the process of being clicked or tapped

Pseudo-classes (cont'd)

Pseudo-classes must appear in the following order:

```
a { text-decoration: none; }  /* turns underlines off for all links */
a:link { color: maroon; }
a:visited { color: gray; }
a:focus { color: maroon; background-color: #ffd9d9; }
a:hover { color: maroon; background-color: #ffd9d9; }
a:active { color: red; background-color: #ffd9d9; }
```

a:link

Links are maroon and not underlined.

a:focus
a:hover

While the mouse is over the link or when the link has focus, the pink background color appears.

a:active

As the mouse button is being pressed, the link turns bright red.

a:visited

After that link has been visited, the link is gray.

Attribute Selectors

Targets elements based on attribute names or values. There are eight types:

Simple attribute selector
Matches an element with a given attribute:

E[*attribute*]

```
img[title] { border: 3px solid;}
```

(Matches every `img` *element that has a* title *attribute)*

Attribute Selectors (cont.)

```
img[title="first grade"] {border: 3px solid;}
```
Select any image that has title equal to “first grade”

```
img[title~="grade"] {border: 3px solid;}
```
Select any image that has title *with* word “grade”

```
[hreflang|="en"] {border: 3px solid;}
```
Match any link that points to a doc. written in English

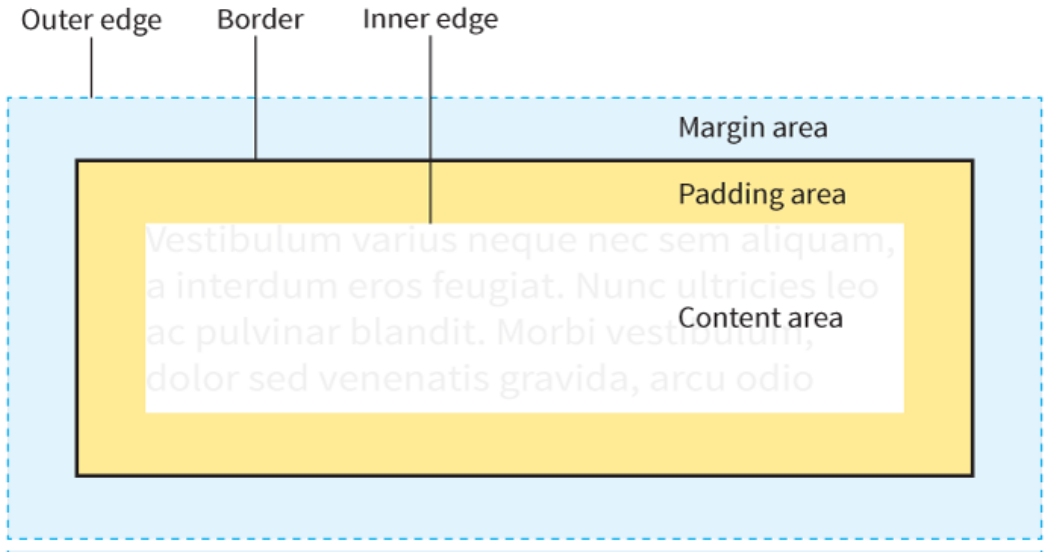
```
img[src^=“/images/icons”]{border: 3px solid;}
```
Apply the style only in images path beginning with /images/icons

```
a[href$=”.pdf”]{border-bottom: 3px solid;}
```
Apply the style to a element that only link to pdf file

```
img[title*=“February”]{border: 3px solid;}
```
Look for any image with ”February” in the title

Box model

- Fundamental concepts of CSS
- Every element in a document generates a box



The parts of an Element Box

NOTE: The margin is indicated with a blue shade and outline, but is invisible in the layout.

Borders

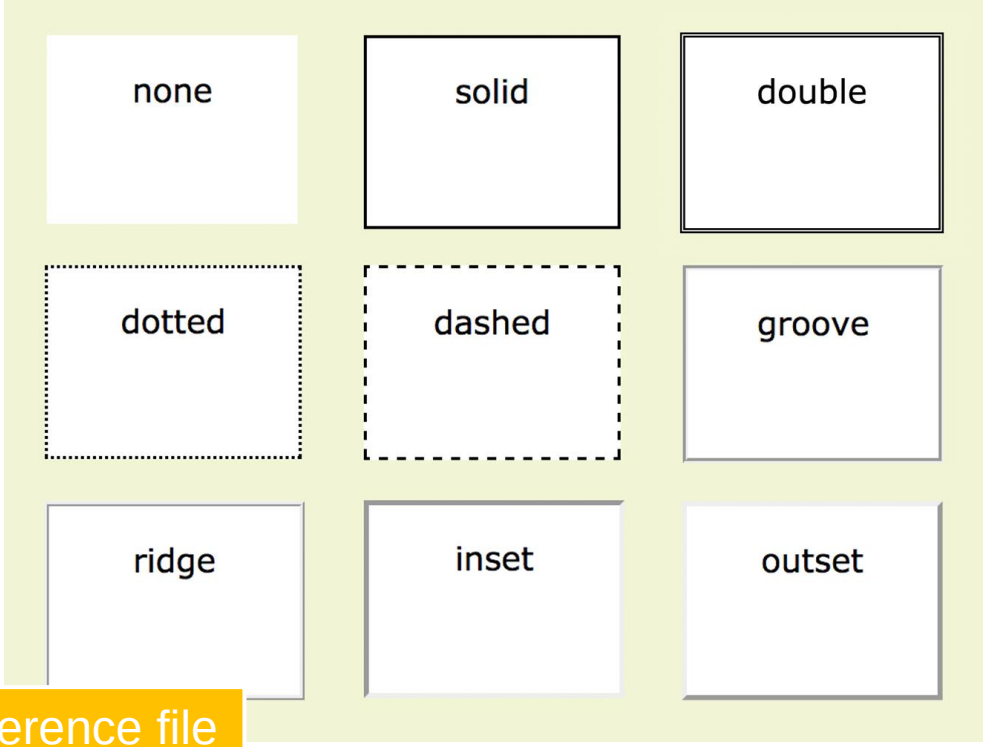
- A **border** is a line drawn around the content area and its (optional) padding.
- The thickness of the border is included in the dimensions of the element box.
- You define **style**, **width** (thickness), and **color** for borders.
- Borders can be applied to single sides or all around

Border Style

`border-style`,
`border-top-style`, `border-right-style`,
`border-bottom-style`, `border-left-style`

Values: `none`, `solid`, `hidden`, `dotted`, `dashed`, `double`, `groove`, `ridge`, `inset`, `outset`

NOTE: The default is `none`, so if you don't define a border style, it won't appear.



See more detail in fullreference file

Margins

margin,
margin-top,
margin-right,
margin-bottom,
margin-left

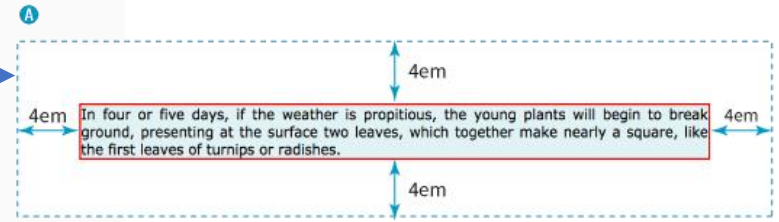
Values: *Length*, *percentage*

Margin is

- An amount of space added on the outside of the border.
- Prevent elements from bumping into one another or the edge of the viewport.

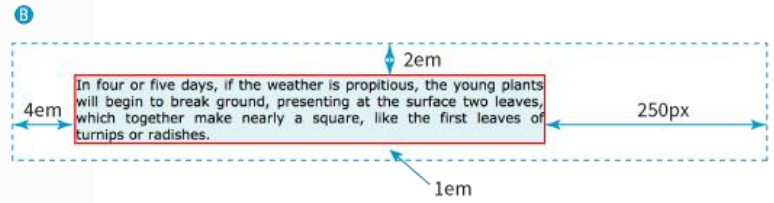
A

```
p#A {
    margin: 4em;
    border: 2px solid red;
    background: #e2f3f5;
}
```



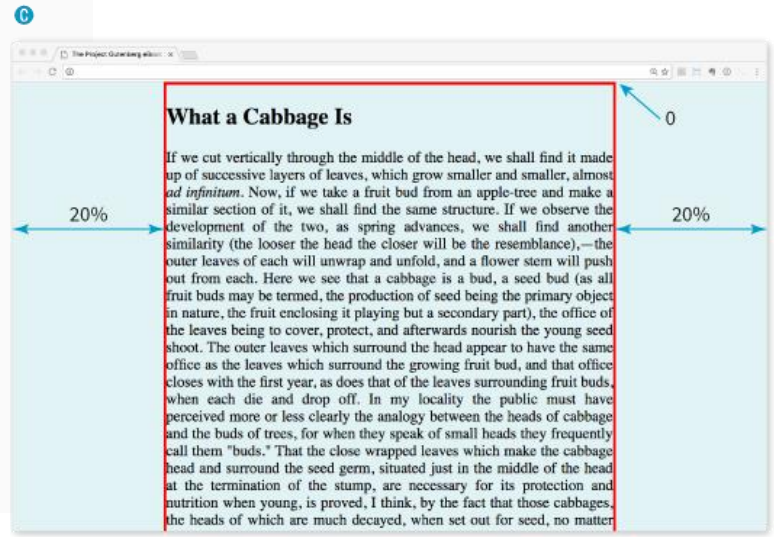
B

```
p#B {
    margin-top: 2em;
    margin-right: 250px;
    margin-bottom: 1em;
    margin-left: 4em;
    border: 2px solid red;
    background: #e2f3f5;
}
```



C

```
body {
    margin: 0 20%;
    border: 3px solid red;
    background-color: #e2f3f5;
}
```



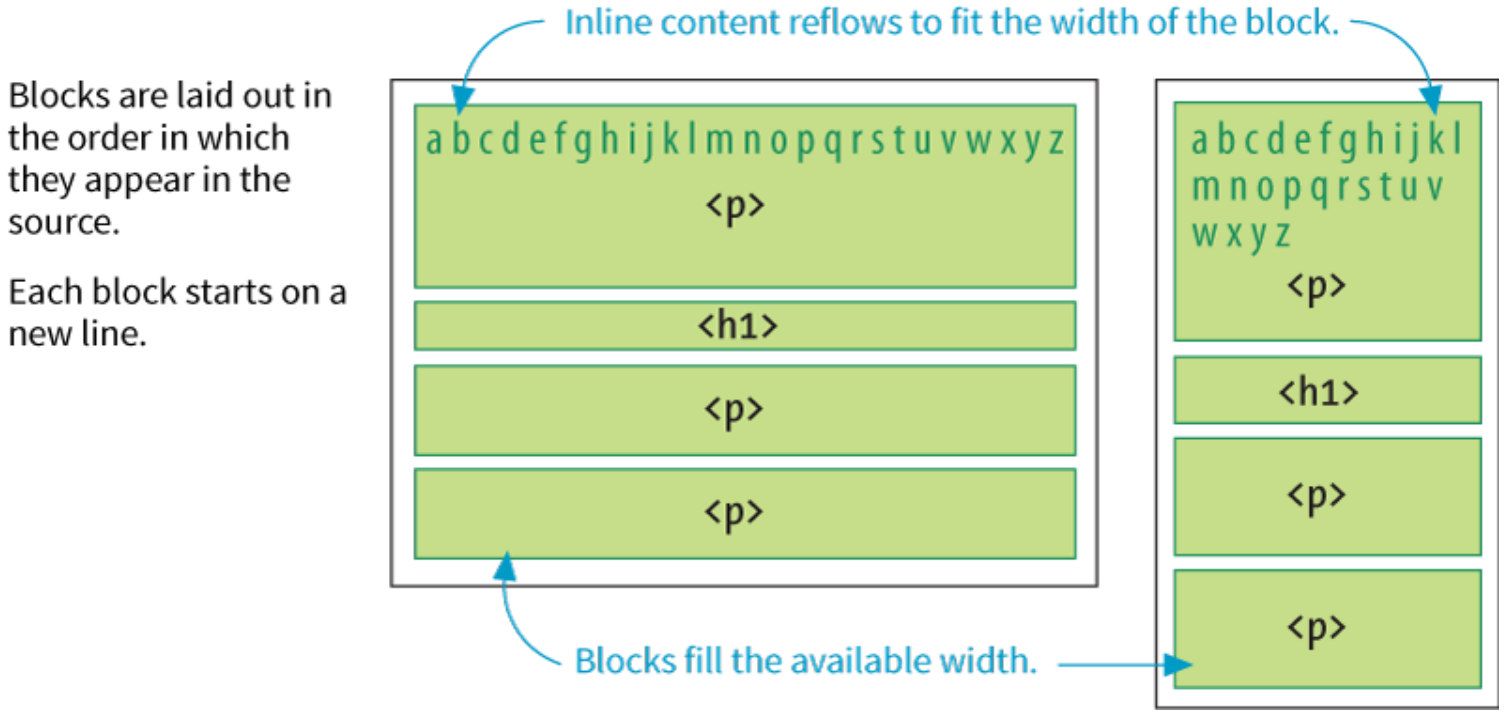
See more detail in fullreference file

Floating and positioning

- Floating elements to the left and right
- Clearing floated elements
- Containing floated elements
- Creating text-wrap shapes
- Relative Positioning
- Absolute positioning and containing blocks
- Fixed positioning

Normal Flow

- In the **normal flow**, elements are laid out from **top to bottom** in the order in which they appear in the source and from **left to right** (in left-to-right reading languages).



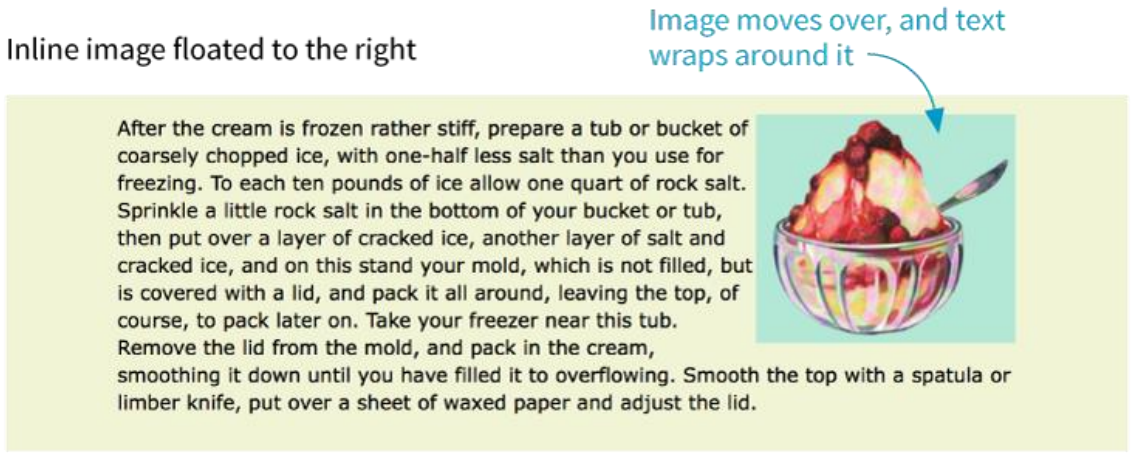
Floating

`float`

Values: `left`, `right`, `none`

Moves an element as far as possible to the left or right and allows the following content to wrap around it:

```
img { float: right; }
```



Note:

- Floated elements are removed from the normal flow but **influence** the surrounding content.
- Floated elements stay **within** the content area of the element that contains it.
- Margins are **always maintained** (they don't collapse) on all sides of floated elements.
- You must provide a **width** for a floated text element (because default width is `auto`).
- Elements **don't float higher** than their reference in the source.

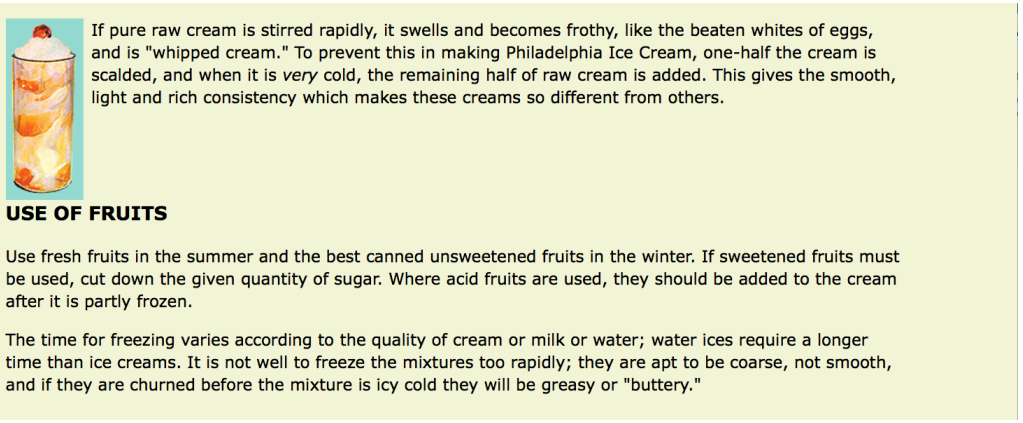
Clearing Floated Elements

`clear`

Values: `left`, `right`, `both`, `none`

Prevents an element from appearing next to a floated element and forces it to start against the next available “clear” space

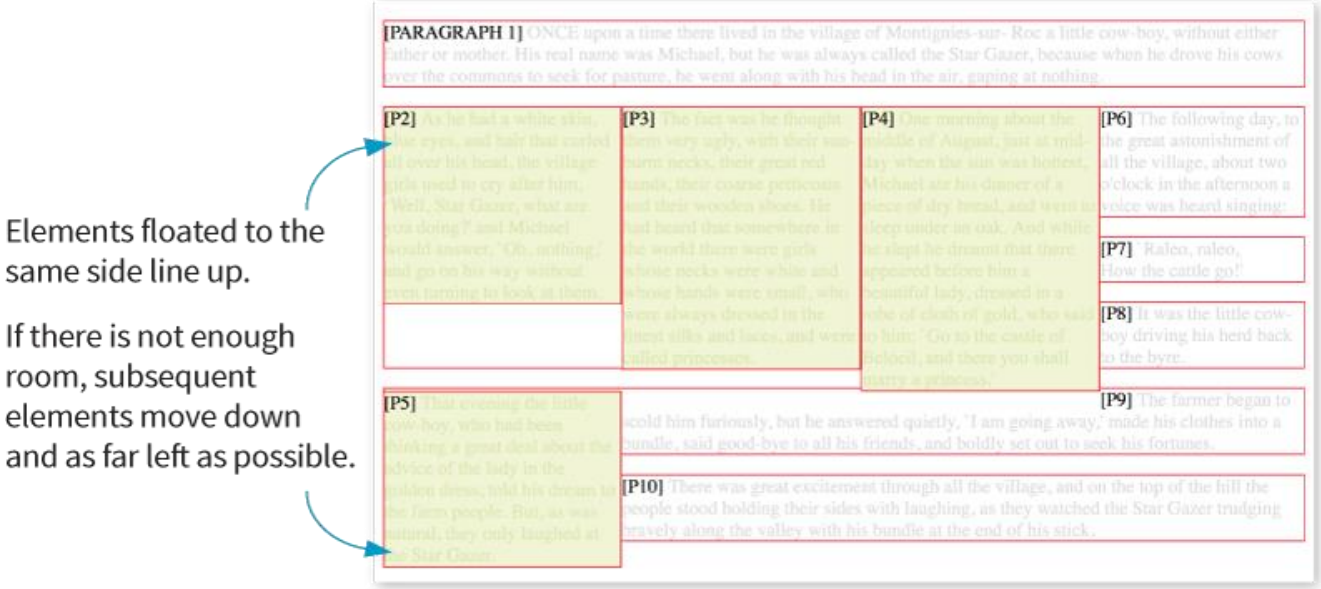
```
img {
  float: left;
  margin-right: .5em;
}
h2 {
  clear: left;
  margin-top: 2em;
}
```



(The `h2` is “cleared” and starts below the floated element.)

Floating Multiple Elements

- When you float multiple elements, browsers follow rules in the spec to ensure they don't overlap.
- Floated elements will be placed as far left or right (as specified) and as high up as space allows.

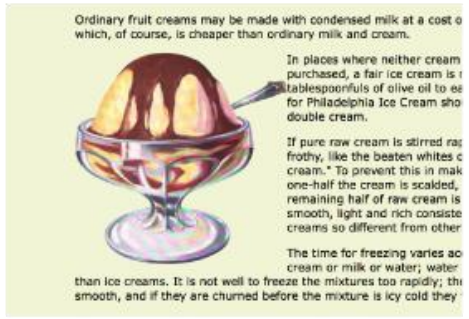


CSS Shapes (Text Wrap)

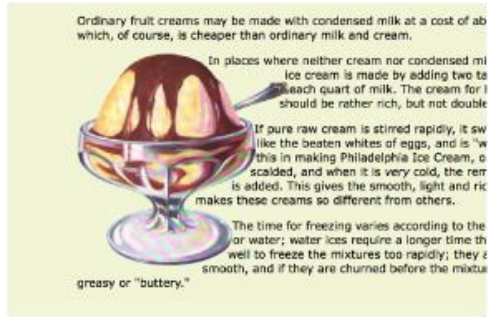
shape-outside

Values: `none, circle(), ellipse(), polygon(), url(), [margin-box | padding-box | content-box ]`

Changes the shape of the text wrap to a circle or ellipse, a complex path, or based on transparent areas in an image

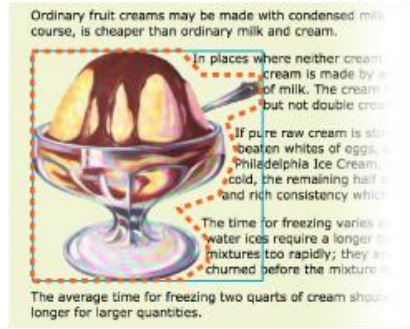


Default text wrap



Using the transparent areas of the image as a guide:
`shape-outside: url(sundae.png);`

Using circle() notation:
`shape-outside: circle(200px);`



The edges of the image (blue) and polygon path (dotted orange) revealed.

Using a path:

```
shape-outside: polygon(0px 0px, 186px 0px,
225px 34px, 300px 34px, 300px 66px, 255px 88px,
267px 127px, 246px 178px, 192px 211px, 226px
236px, 226px 273px, 209px 300px, 0px 300px);
```

shape-margin

Values: *length, percentage*

Specifies an amount of space to hold between the image and the wrapped text

Positioning

`position`

Values: `static`, `relative`, `absolute`, `fixed`, `sticky`

Indicates that an element is to be positioned and specifies which positioning method to use

- **Static:** The default position in the flow
- **Relative:** Moves the element relative to its original position
- **Absolute:** Removes the element from the flow and places it relative to the viewport or other containing element
- **Fixed:** Element stays in one position relative to the viewport
- **Sticky:** Element starts in the flow but stays fixed once it scrolls to a particular position relative to the viewport

`top, right, bottom, left`

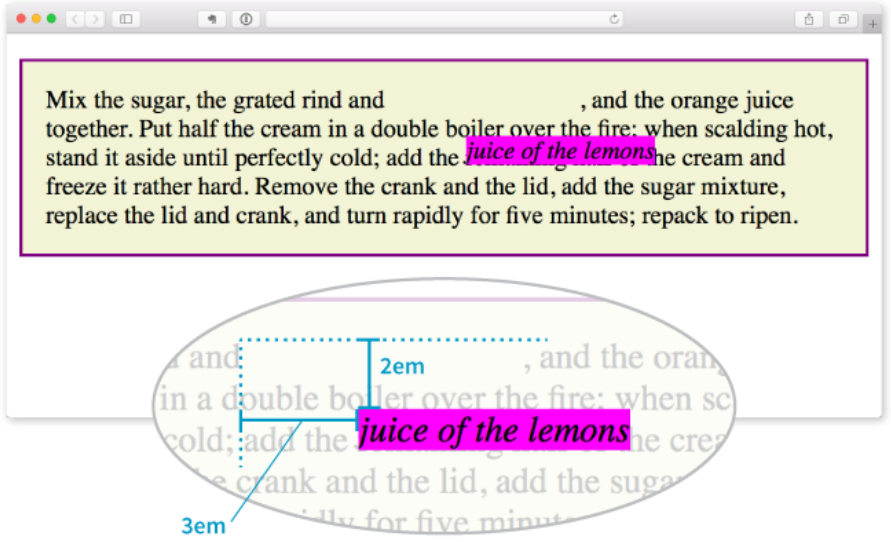
Values: *`length`*, *`percentage`*, `auto`

Offset properties that provide the distance the element should be moved *away* from that respective edge

Relative Positioning

- Moves the element **relative** to its original position
- The space it originally occupied is preserved.

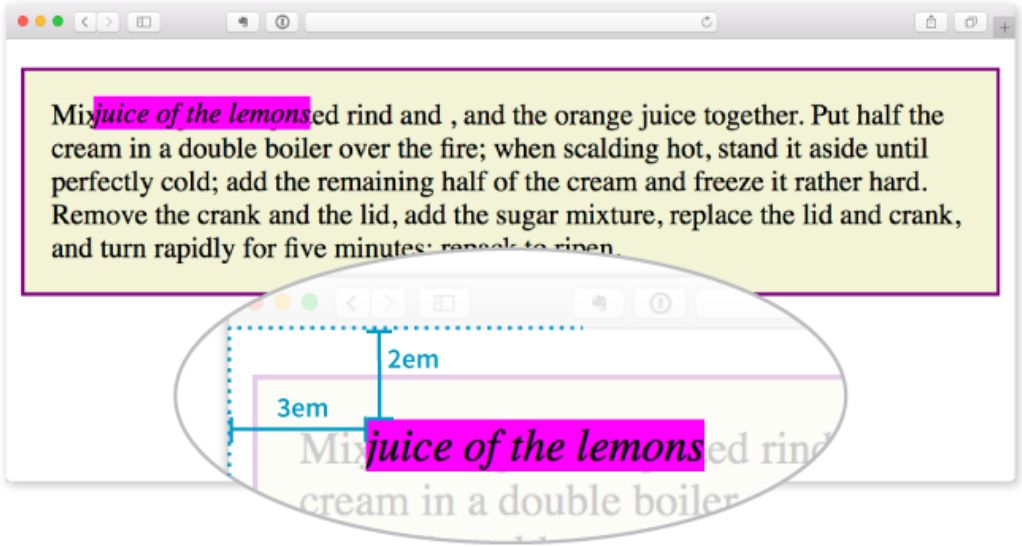
```
em {
  position: relative;
  top: 2em; /* moves it down */
  left: 3em; /* moves it right */
  background-color: fuchsia;
}
```



Absolute Positioning

- Moves the element relative to the **viewport** or **containing block** element
- The space it originally occupied is closed up.

```
em {
  position: absolute;
  top: 2em;
  left: 3em;
  background-color: fuchsia;
}
```



Break for 10 mins

CSS Layout with FLEXBOX

Flexbox Container
Rows and Columns (Direction)
Flex Flow (Direction + Wrap)
Aligning on the Main Axis
Aligning on the Cross Axis
Aligning with Margins
Specifying How Items “Flex”
Changing Item Order
Browser Support for Flexbox

About Flexbox

- **Flexbox** is a display mode that lays out elements along **one axis** (horizontal or vertical).
- Useful for menu options, galleries, product listings, etc.
- Items in a flexbox can expand, shrink, and/or wrap onto multiple lines, making it a great tool for responsive layouts.
- Items can be reordered, so they aren't tied to the source order.
- Flexbox can be used for individual components on a page or the whole page layout.

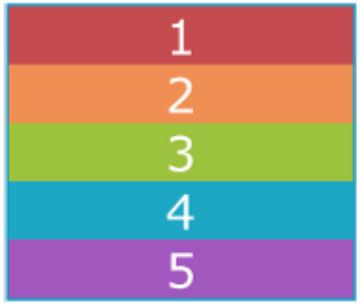
Flexbox Container

```
display: flex
```

- To turn on Flexbox mode, set the element's `display` to `flex`.
- This makes the element a **flexbox container**.
- All of its direct children become **flex items** in that container.
- By default, items line up in the writing direction of the document (left to right rows in left-to-right reading languages).

Flexbox Container (cont'd)

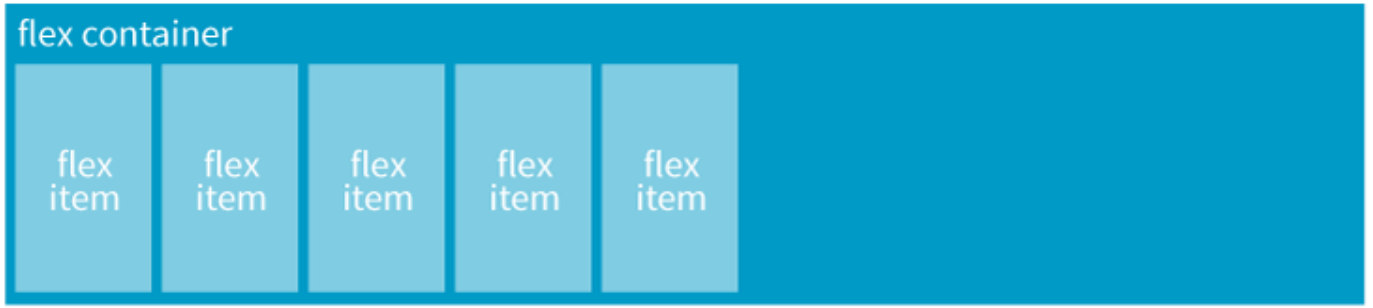
By default, the `divs` display as block elements, stacking up vertically. Turning on flexbox mode makes them line up in a row.



block layout mode

```
display: flex;
```

flexbox layout mode

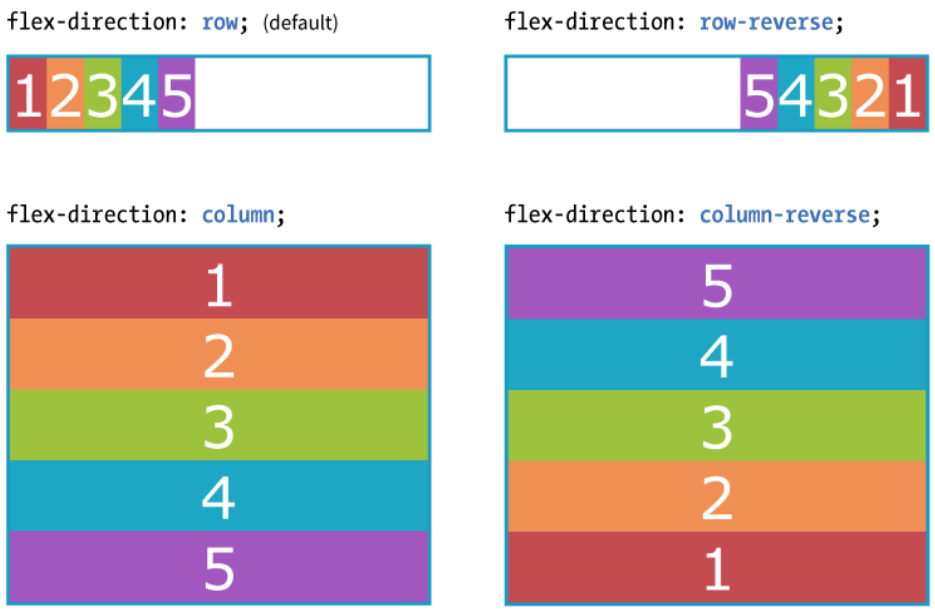


Rows and Columns (Direction)

`flex-direction`

Values: `row`, `column`, `row-reverse`, `column-reverse`

The default value is `row` (for L-to-R languages), but you can change the direction so items flow in columns or in reverse order:



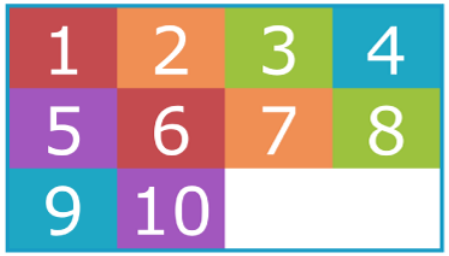
Wrapping Flex Lines

`flex-wrap`

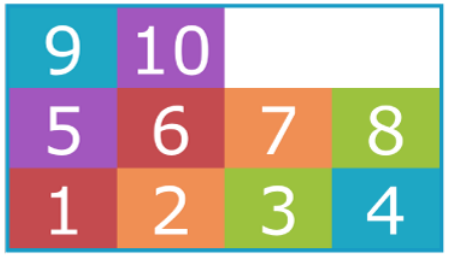
Values: `wrap`, `nowrap`, `wrap-reverse`

Flex items line up on one axis, but you can allow that axis to wrap onto multiple lines with the `flex-wrap` property:

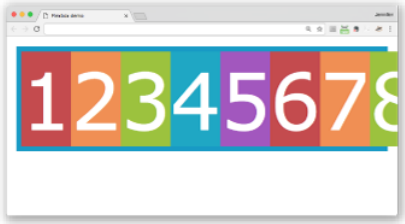
`flex-wrap: wrap;`



`flex-wrap: wrap-reverse;`



`flex-wrap: nowrap;` (default)



When wrapping is disabled, flex items squish if there is not enough room, and if they can't squish any further, may get cut off if there is not enough room in the viewport.

Flex Flow (Direction + Wrap)

`flex-flow`

Values: *`Flex-direction flex-flow`*

The shorthand `flex-flow` property specifies both direction and wrap in one declaration.

Example

```
#container {
  display: flex;
  height: 350px;
  flex-flow: column wrap;
}
```

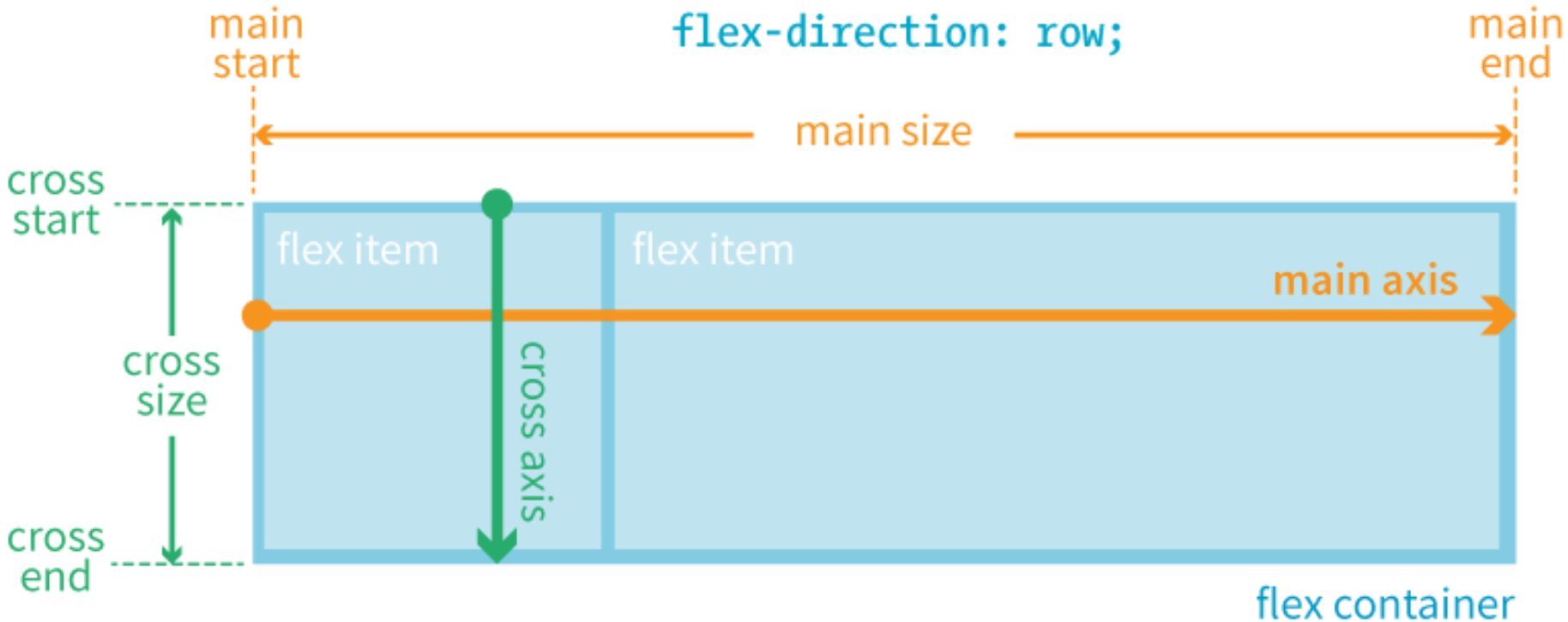
Flexbox Alignment Terminology

- Flexbox is **“direction-agnostic”** so we talk in terms of *main axis* and *cross axis* instead of rows and columns.
- The **main axis** runs in whatever direction the flow has been set.
- The **cross axis** runs perpendicular to the main axis.
- Both axes have a **start**, **end**, and **size**.

ROW: Main and Cross Axes

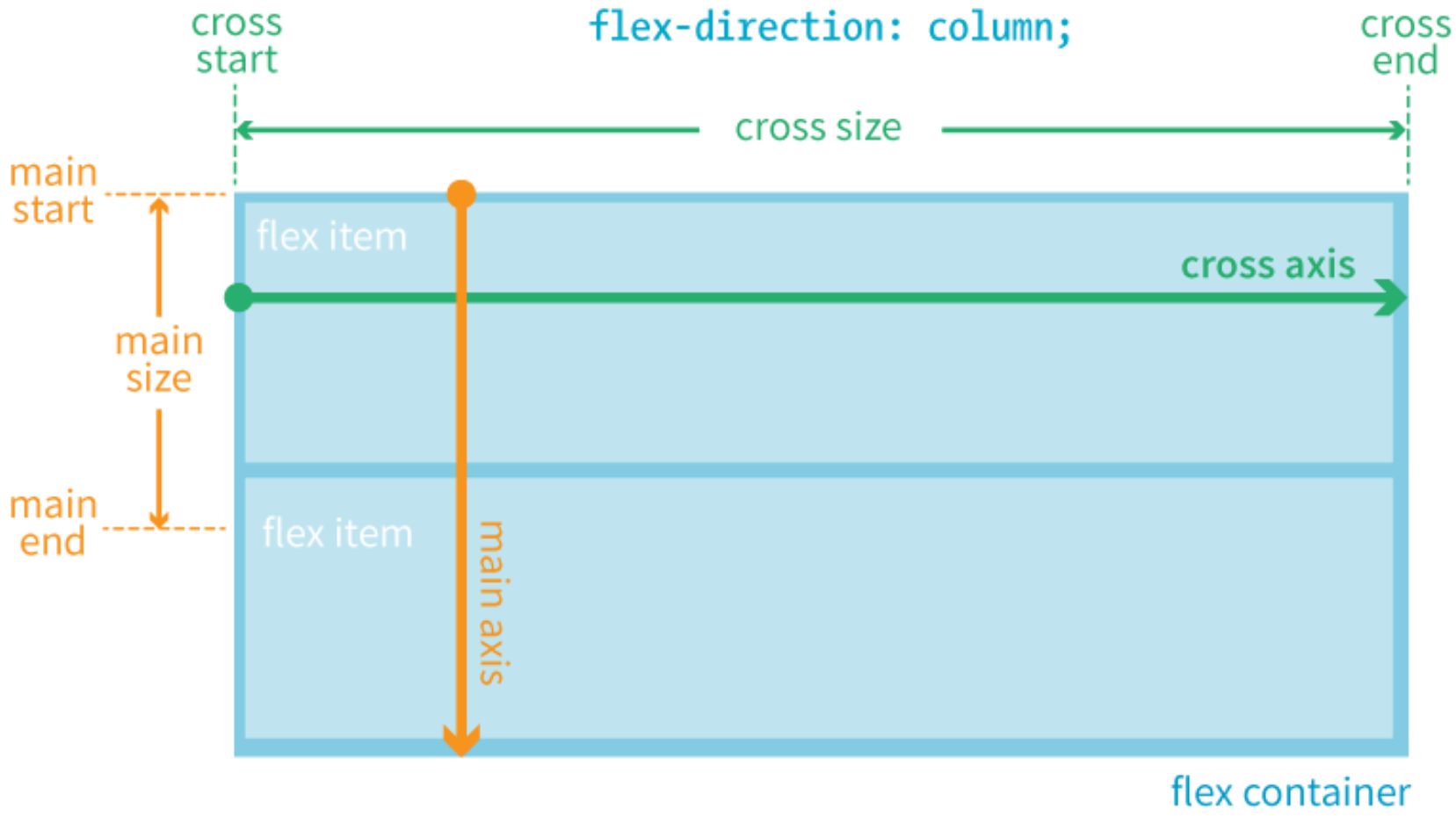
FOR LANGUAGES THAT READ HORIZONTALLY FROM LEFT TO RIGHT:

When `flex-direction` is set to `row`, the main axis is horizontal and the cross axis is vertical.



COLUMN: Main and Cross Axes

When **flex-direction** is set to **column**, the main axis is vertical and the cross axis is horizontal.



Aligning on the Main Axis

`justify-content`

Values: `flex-start`, `flex-end`, `center`, `space-between`, `space-around`

When there is space left over on the **main axis**, you can specify how the items align with the `justify-content` property (notice we say *start* and *end* instead of left/right or top/bottom).

The `justify-content` property applies to the **flex container**.

Example:

```
#container {
  display: flex;
  justify-content: flex-start;
}
```

When the direction is **row**, and the **main axis is horizontal**

justify-content: flex-start; (default)

justify-content: flex-end;

justify-content: center;

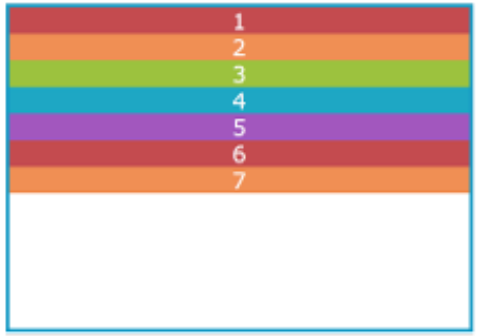
justify-content: space-between;

justify-content: space-around;

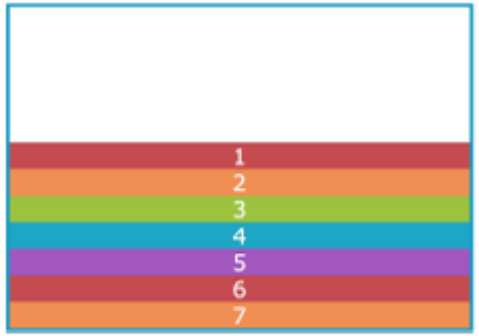
Aligning on the Main Axis (cont'd.)

When the direction is **column**, and the **main axis is vertical**

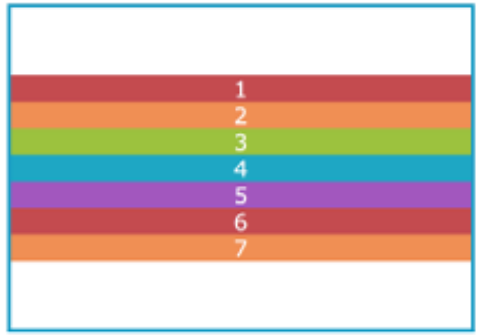
`justify-content: flex-start;` (default)



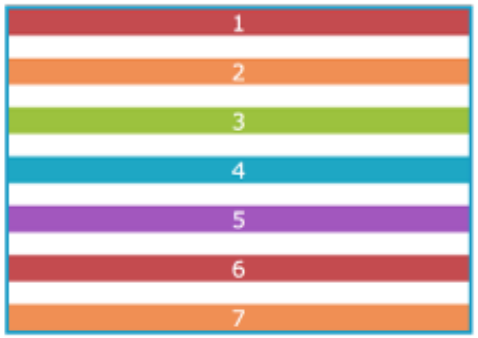
`justify-content: flex-end;`



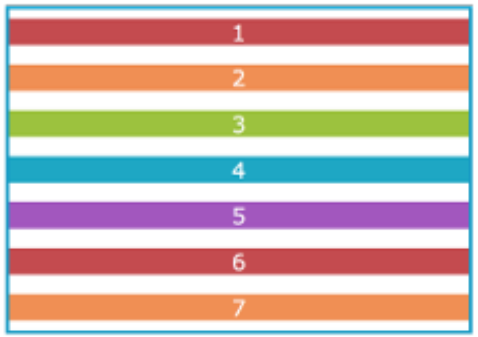
`justify-content: center;`



`justify-content: space-between;`



`justify-content: space-around;`



NOTE: I needed to specify a height on the container to create **extra space** on the main axis. By default, it's just high enough to contain the content.

A WORD FROM THE AUTHOR

“Keeping the main and cross axes straight in your mind when changing between rows and columns is one of the trickiest parts of using Flexbox.

Once you master that, you’ve got it!”

—Jennifer Robbins

Aligning on the Cross Axis

`align-items`

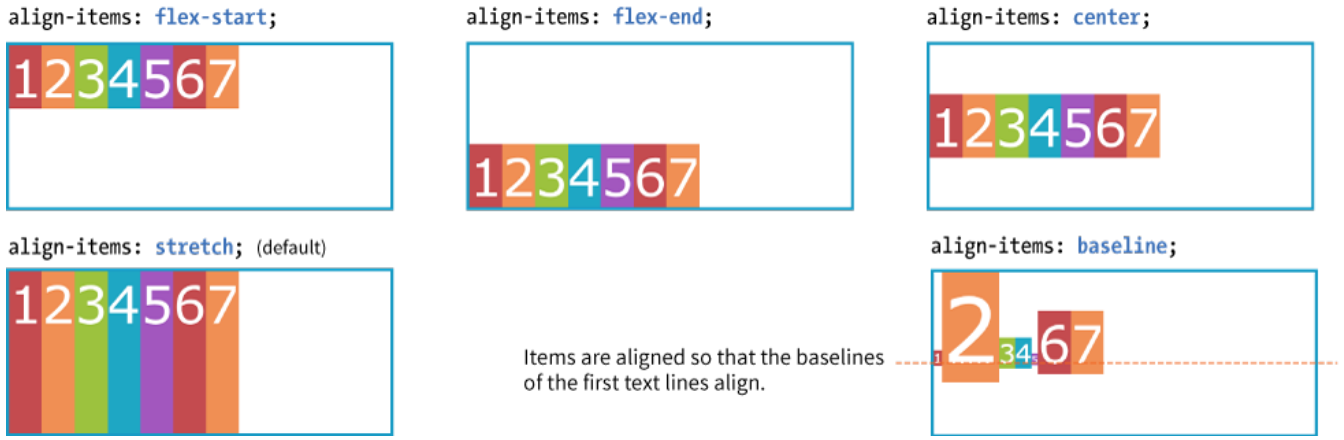
Values: `flex-start`, `flex-end`, `center`, `baseline`, `stretch`

When there is space left over on the **cross axis**, you can specify how the items align with the **`align-items`** property.

The **`align-items`** property applies to the **flex container**.

Example:

```
#container {
  display: flex;
  flex-direction: row;
  height: 200px;
  align-items: flex-start;
}
```



When the direction is **row**, the main axis is horizontal, and the **cross axis is vertical**. (need to specify extra height in the container)

Aligning on the CROSS Axis (cont'd)

`align-self`

Values: `flex-start`, `flex-end`, `center`, `baseline`, `stretch`

Aligns an **individual item** on the cross axis. This is useful if one or more items should override the `align-items` setting for the container.

The `align-self` property applies to the **flex item**.

Example:

```
.box4 {
  align-self: flex-end;
}
```

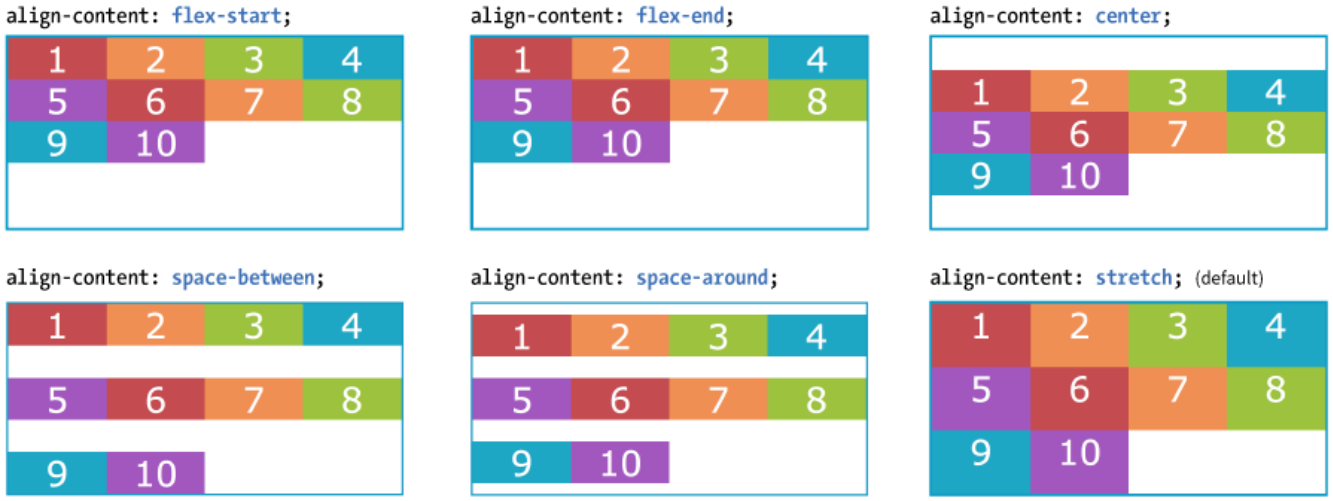
Aligning on the CROSS Axis (cont'd)

`align-content`

Values: `flex-start`, `flex-end`, `center`, `space-around`, `space-between`, `stretch`

When lines are set to **wrap** and there is extra space on the cross axis, use `align-content` to **align the lines of content**.

The `align-content` property applies to the **flex container**.



CSS Layout with GRID

- Grid terminology
- Grid display type
- Creating the grid template
- Naming grid areas
- Placing grid items
- Implicit grid behavior
- Grid spacing and alignment

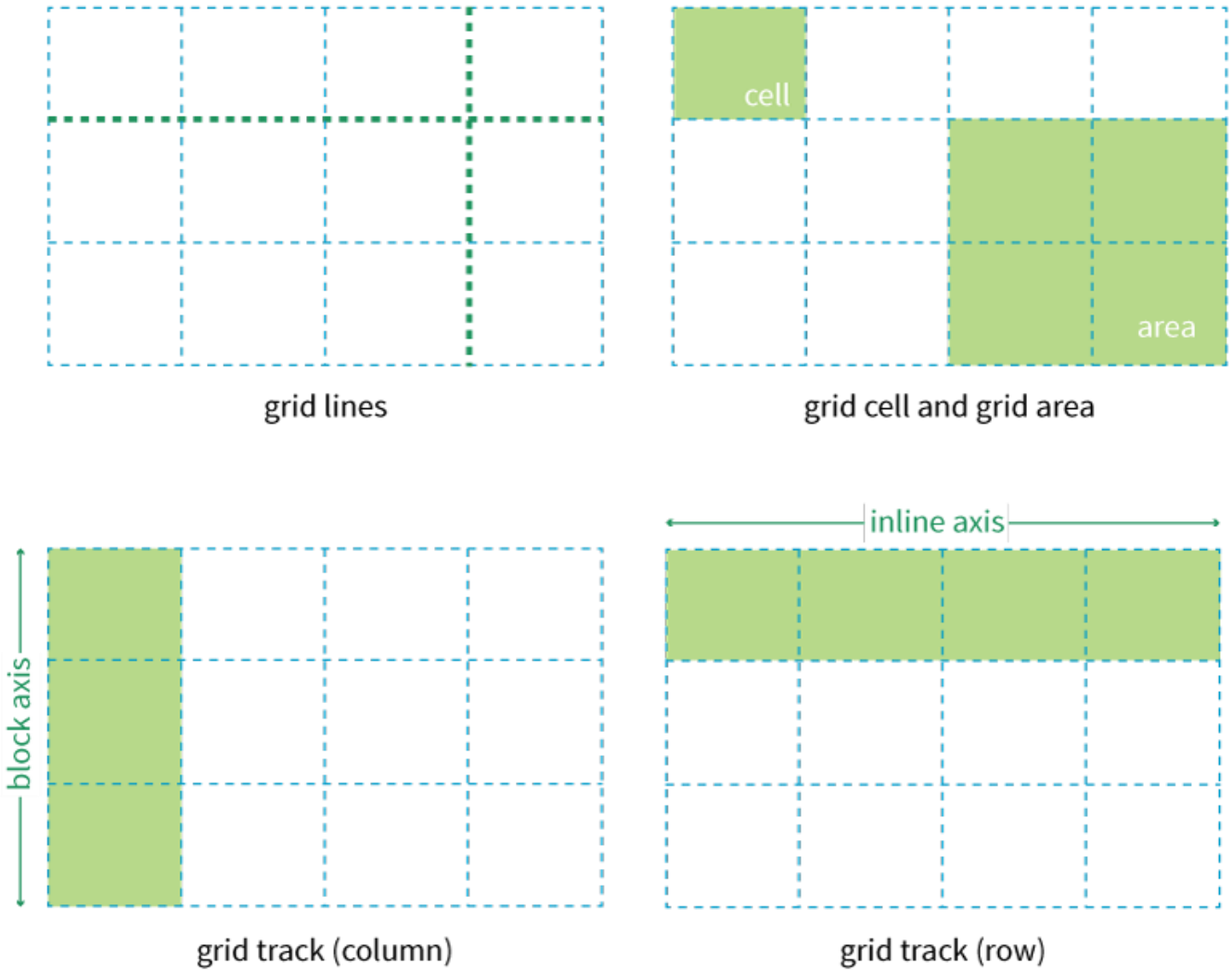
Grid vs. Flexbox

- Flexbox lays out elements on **one axis only**
- Grid lay out elements in rows and column – completely flexible to fit a variety of screen size or mimic a print page layout
- Grid Layout Module is one of the more **complex** specs in CSS
- Here is for a good head start to Grid

How CSS Grids Work

1. Set an element's `display` to `grid` to establish a **grid container**. Its children become **grid items**.
2. Set up the columns and rows for the grid (explicitly or with rules for how they are created on the fly).
3. Assign each grid item to an area on the grid (or let them flow in automatically in sequential order).

Grid Terminology



Creating a Grid Container

- To make an element a **grid container**, set its **`display`** property to **`grid`**.

- All of its children automatically become **grid items**.

The markup

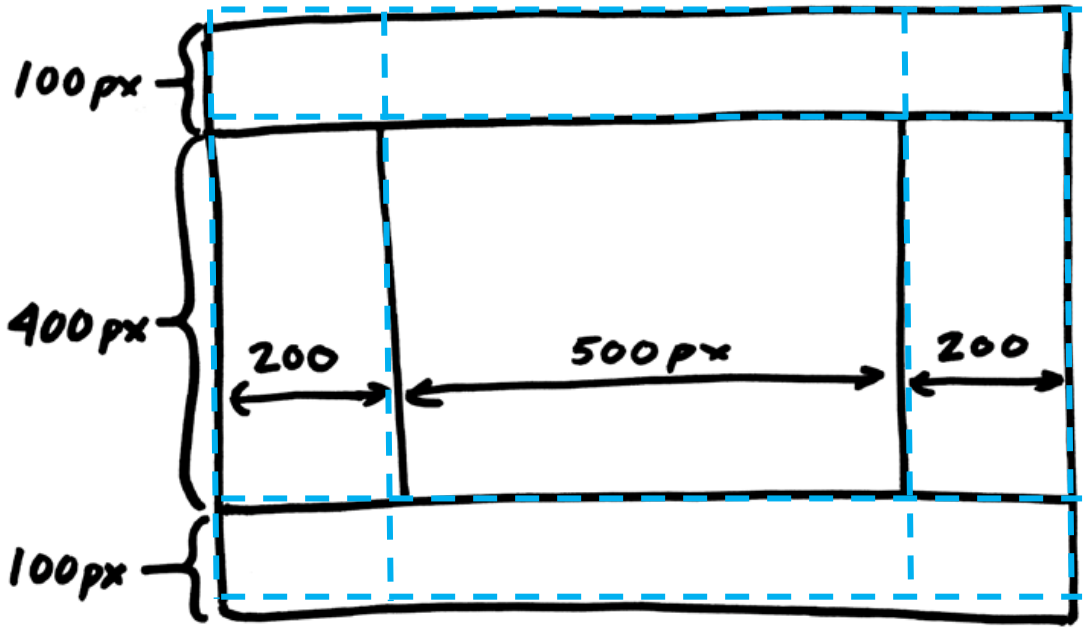
```
<div id="layout">
  <div id="one">One</div>
  <div id="two">Two</div>
  <div id="three">Three</div>
  <div id="four">Four</div>
  <div id="five">Five</div>
</div>
```

The styles

```
#layout {
  display: grid;
}
```

Setting up the grid

- Create a sketch of how you like your final grid to look



The blue dotted lines shows how many rows and column the grid requires to create this structure

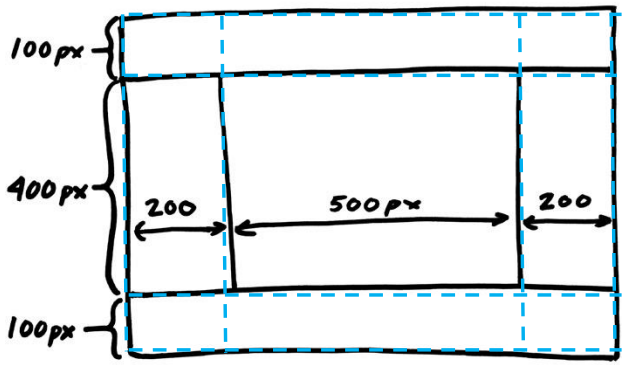
Defining Row and Column Tracks

`grid-template-rows`
`grid-template-columns`

Values: `none`, *list of* **track** *sizes and optional line names*

- The value of `grid-tempate-rows` is a list of the *heights* of each row track in the grid.
- The value of `grid-template-columns` is a list of the *widths* of each column track in the grid.

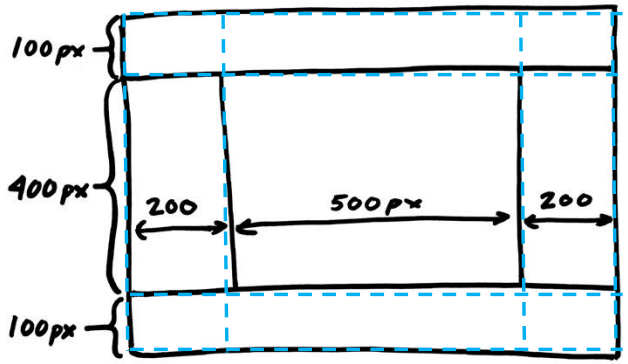
```
#layout {
  display: grid;
  grid-template-rows: 100px 400px 100px;
  grid-template-columns: 200px 500px 200px;
}
```

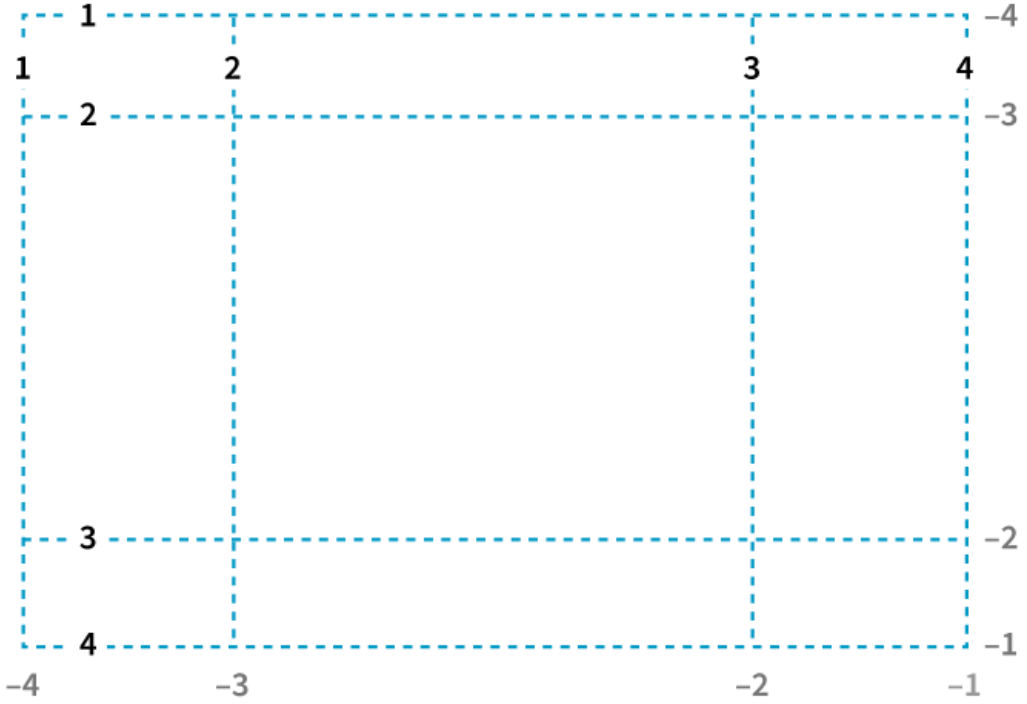


- The number of sizes provided determines the number of rows/columns in the grid.
- This grid in the example above has 3 rows and 3 columns.

Grid Line Numbers

Browsers assign a number to every grid line automatically, starting with **1** from the beginning of each row and column track and also starting with **–1** from the end.

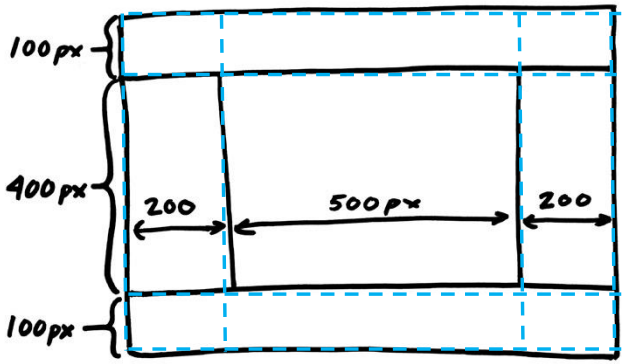




Grid Line Names

- **You can also assign names to lines** to make them more intuitive to reference later.
- Grid line names are added in square brackets in the position they appear relative to the tracks.
- To give a line more than one name, include all the names in brackets, separated by spaces.

```
#layout {
  display: grid;
  grid-template-rows: [header-start] 100px [header-end content-start]
400px [content-end footer-start] 100px;
  grid-template-columns: [ads] 200px [main] 500px [links] 200px;
}
```



Track Size Values

The CSS Grid spec provides a *lot* of ways to specify the width and height of a track. Some of these ways allow tracks to adapt to available space and/or to the content they contain:

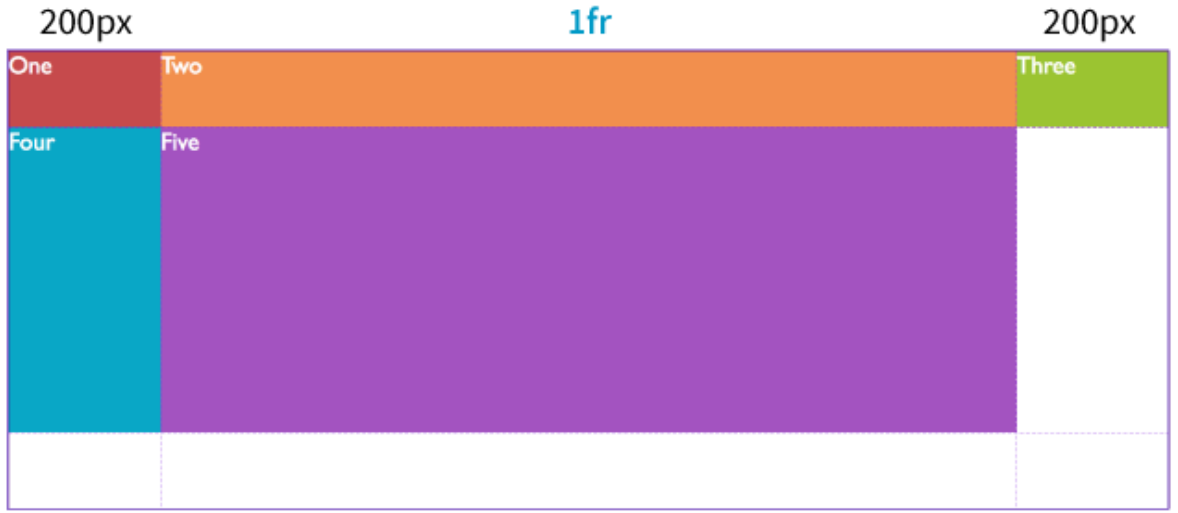
- Lengths (such as `px` or `em`)
- Percentage values (`%`)
- **Fractional units (`fr`)**
- `minmax()`
- `min-content`, `max-content`
- `auto`
- `fit-content()`

Tips: You can try to use Firefox's CSS Grid Inspector and Layout Panel, or some extension of Chrome (eg: Gridman) to see Grid layout of a specific web page

Fractional Units (fr)

The Grid-specific fractional unit (**fr**) expands and contracts based on available space:

```
#layout {
  display: grid;
  grid-template-rows: 100px 400px 100px;
  grid-template-columns: 200px 1fr 200px;
}
```



Size Range with minmax()

- The `minmax()` function constricts the size range for the track by setting a minimum and maximum dimension.
- It’s used in place of a specific track size.
- This rule sets the middle column to at least 15em but never more than 45em:

```
grid-template-columns: 200px minmax(15em, 45em) 200px;
```

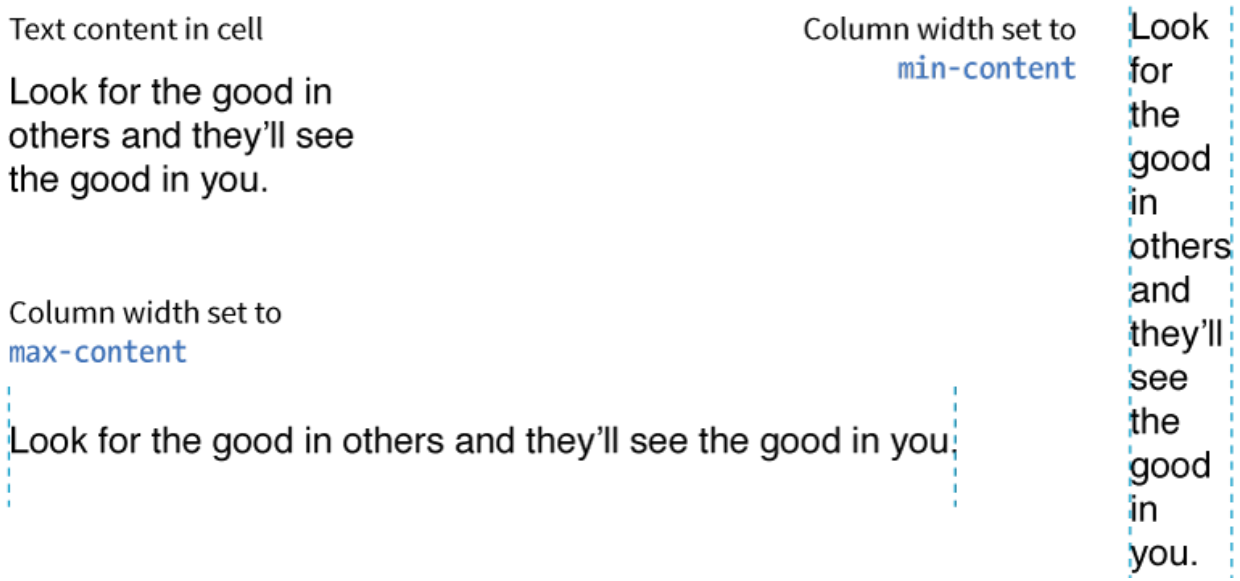
Content-based sizing

`min-content` is the smallest that a track can be depending on the content.

`max-content` allots the maximum amount of space needed for a content.

`auto` lets the browser take care of it.

** If you are not sure which to use, start with `auto` for content-based sizing.



Repeating Track Sizes

The shortcut **`repeat()`** function lets you repeat patterns in track sizes:

`repeat(#,` *`track pattern`*`)`

The first number is the number of repetitions. The track sizes after the comma provide the pattern:

This

```
grid-template-columns: 200px 20px 1fr 20px 1fr 20px 1fr
20px 1fr 20px 1fr 20px 1fr 200px;
```

Is equivalent to

```
grid-template-columns: 200px repeat(5, 20px 1fr) 200px;
```

(Here repeat() is used in a longer sequence of track sizes.
It repeats the track sizes 20px 1fr 5 times.)

Repeating Track Sizes (cont'd.)

You can let the browser figure out how many times a repeated pattern will fit with **`auto-fill`** and **`auto-fit`** values instead of a number:

```
grid-template-rows: repeat(auto-fill, 15em);
```

`auto-fill` creates as many tracks as will fit in the available space, even if there's not enough content to fill all the tracks. (Fills the row with as many columns

`auto-fit` creates as many tracks as will fit, dropping empty tracks from the layout. (Fits the currently available rows into the space by expanding them)

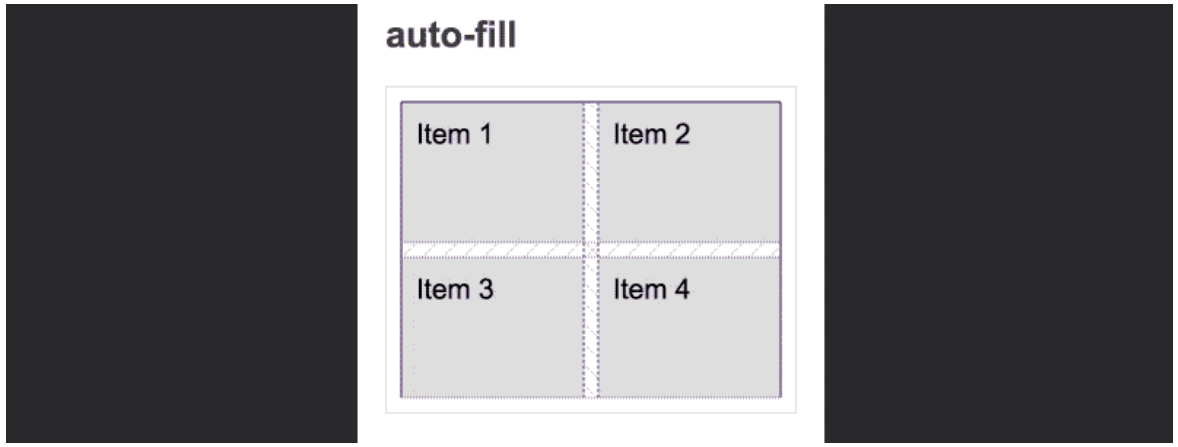
NOTE: If there's leftover space in the container, it's distributed according to the provided vertical and horizontal alignment values.

auto-fill and auto-fit demo

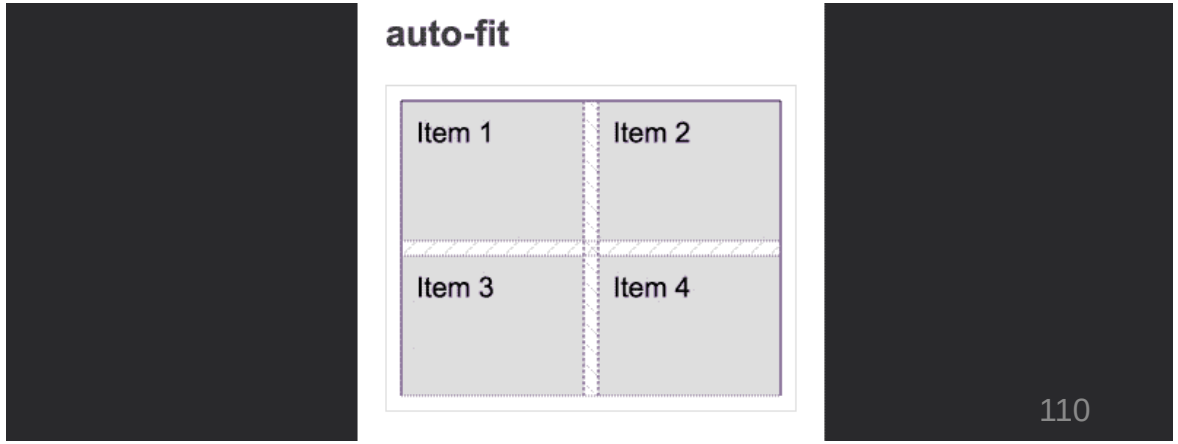
- Auto-fill will fill the extra space with a new empty grid
  So if the width of the parent element is widen, there is extra space, a new grid will be populated

- When using auto-fit, the extra space in the parent element is eliminated to be `0px`.
  Therefore, a grid item width of `1fr` will fill the extra space

```
grid-template-columns: repeat(auto-fill,
minmax(100px, 1fr));
```




```
grid-template-columns: repeat(auto-fit,
minmax(100px, 1fr));
```



Giving Names to Grid Areas

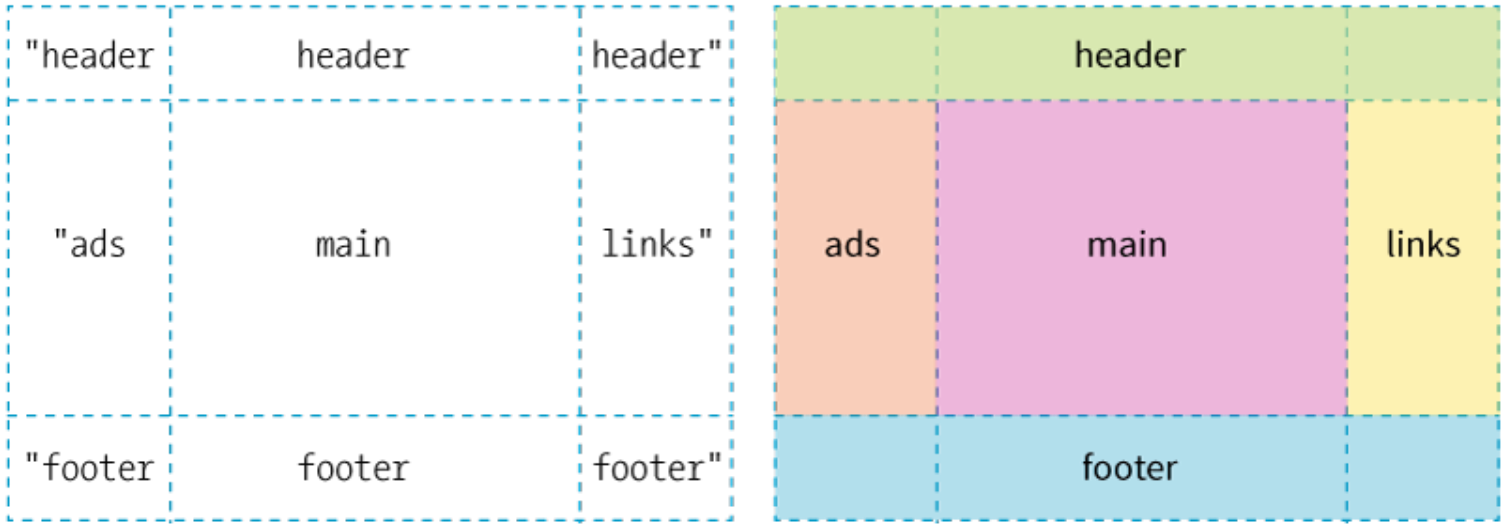
`grid-template-areas`

Values: `none`, *series of area names by row*

- `grid-template-areas` lets you assign names to areas in the grid to make it easier to place items in that area later.
- The value is a list of names for every cell in the grid, **listed by row**.
- When neighboring cells share a name, they **form a grid area** with that name.

Giving Names to Grid Areas (cont'd)

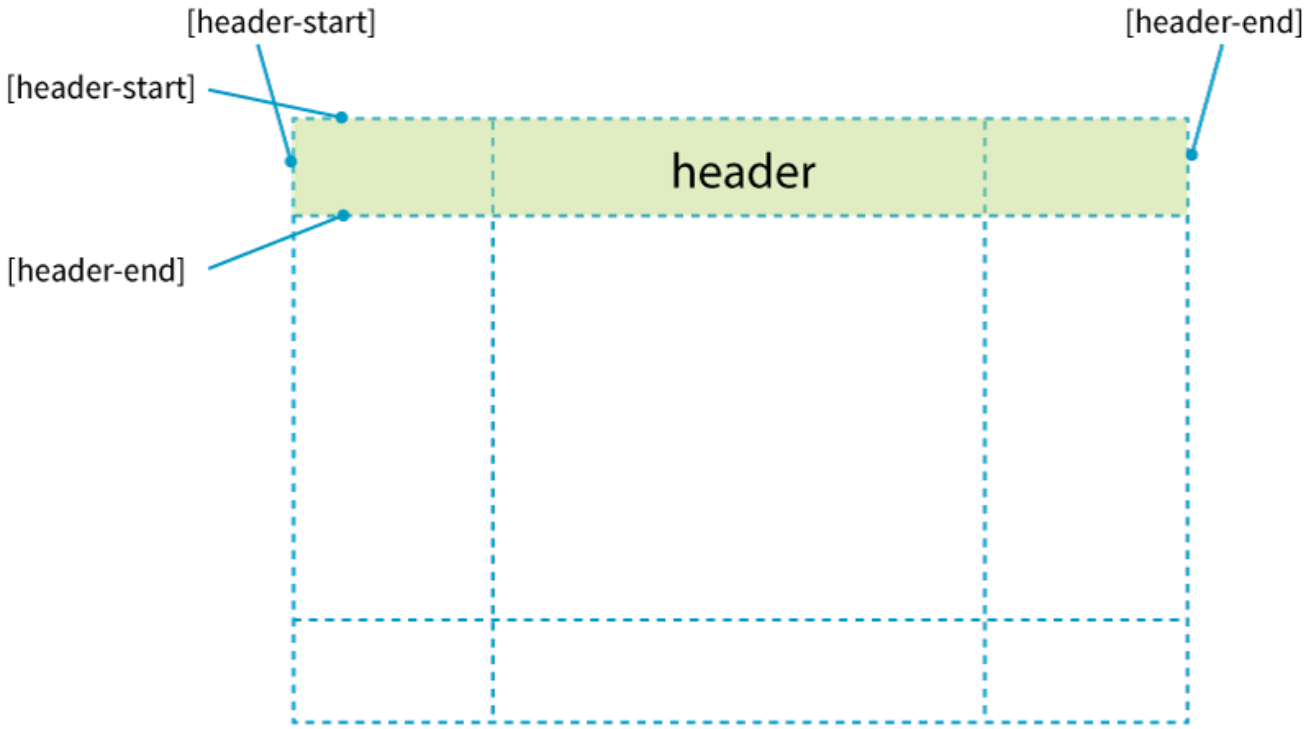
```
#layout {
  display: grid;
  grid-template-rows: [header-start] 100px [content-start] 400px
[footer-start] 100px;
  grid-template-columns: [ads] 200px [main] 1fr [links] 200px;
  grid-template-areas:
    "header  header  header"
    "ads     main    links"
    "footer  footer  footer"
}
```



Giving Names to Grid Areas (cont'd)

- Assigning names to lines with -start and -end suffixes creates an area name **implicitly**.

- Similarly, when you specify an area name with **`grid-template-areas`**, line names with -start and -end suffixes are **implicitly generated**.



The grid Shorthand Property

grid

Values: `none`, *`row info/column info`*

The `grid` shorthand sets values for `grid-template-rows`, `grid-template-columns`, and `grid-template-areas`.

NOTE: The `grid` shorthand is available, but the word on the street is that it's more difficult to use than separate template properties.

Example grid: rows / columns

```
#layout {
  display: grid;
  grid: 100px 400px 100px / 200px 1fr 200px;
}
```

See more about Grid and shorthand property at /https://css-tricks.com/snippets/css/complete-guide-grid

Placing Items on the Grid Using Areas

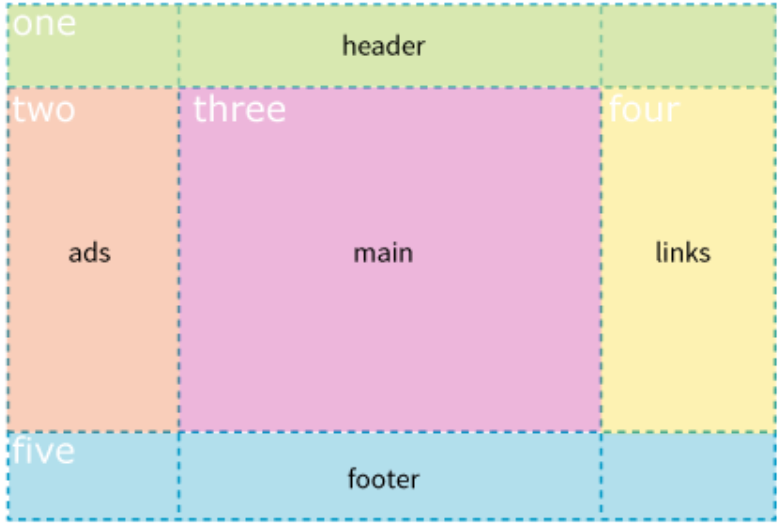
The easy method

`grid-area`

Values: *`Area name, 1 to 4 line identifiers`*

Positions an item in an area created with `grid-template-areas`:

```
#one { grid-area: header; }
#two { grid-area: ads; }
#three { grid-area: main; }
#four { grid-area: links; }
#five { grid-area: footer; }
```



Automatically Generated Tracks

`grid-auto-rows`
`grid-auto-columns`

Values: *List of track sizes*

Provide one or more track sizes for automated tracks. If you provide more than one value, it acts as a repeating pattern.

Example:

Column widths are set explicitly with a template, but columns will be generated automatically with a height of 200 pixels:

```
grid-template-columns: repeat(3, 1fr);
grid-auto-rows: 200px;
```

Bootstrap

Some of the important classes especially Grid in Bootstrap

What is Responsive Web Design?

- Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.

- Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites.

- Bootstrap is completely free to download and use!

Bootstrap at a glance

- Bootstrap is a popular front-end development framework created by Twitter. (originally name Twitter Blueprint)
- It provides pre-written HTML, CSS, and JavaScript code that developers can use to build responsive and mobile-first websites.
- Bootstrap includes a grid system that makes it easy to create layouts with different numbers of columns.
- It also includes a variety of UI components, such as buttons, forms, navigation menus, and alerts.
- Bootstrap has built-in support for popular CSS preprocessors like Sass and Less, making it easy to customize the look and feel of a website.
- Bootstrap is open-source and free to use, with an active community of developers contributing to its development and maintenance.
- Bootstrap can be used with a variety of web development frameworks, including React, Angular, and Vue.js.
- With Bootstrap, developers can save time and effort by not having to write as much code from scratch, while still creating professional-looking websites that are mobile-friendly and accessible.

Bootstrap

Here are some examples of popular websites that use Bootstrap:

1. Airbnb
2. Spotify
3. Netflix
4. Slack
5. Udemy
6. GitHub
7. Lyft
8. Fiverr
9. Udacity
10. Coursera

Bootstrap 4 vs. Bootstrap 5

Bootstrap 4:

- Released in 2018
- Widely used for building responsive and mobile-friendly websites
- Includes a grid system for creating layouts with different numbers of columns
- Offers a variety of UI components, such as buttons, forms, navigation menus, and alerts
- Supports popular CSS preprocessors like Sass and Less for easy customization
- Used by over 20% of all websites on the internet

Bootstrap 5:

- Released in 2021
- Offers a smaller file size and improved performance compared to Bootstrap 4
- No longer requires jQuery, making it easier to integrate with other JavaScript frameworks
- Includes updated utility classes for working with flexbox layouts
- Offers simplified customization options for creating a customized version of the framework
- Includes new components such as the accordion and offcanvas
- Uses a new default font called Inter for improved readability
- Provides improved documentation with more examples, code snippets, and tutorials

Adding Bootstrap to your website

https://getbootstrap.com/docs/5.3/getting-started/introduction/

1. Open your preferred text editor or IDE and create a new HTML file.

2. Go to the Bootstrap 5 official website at https://getbootstrap.com/ and click on the "Get started" button.

3. Under CDN via `jsDelivr`, copy the link to the CSS file and paste it into the head section of your HTML file.

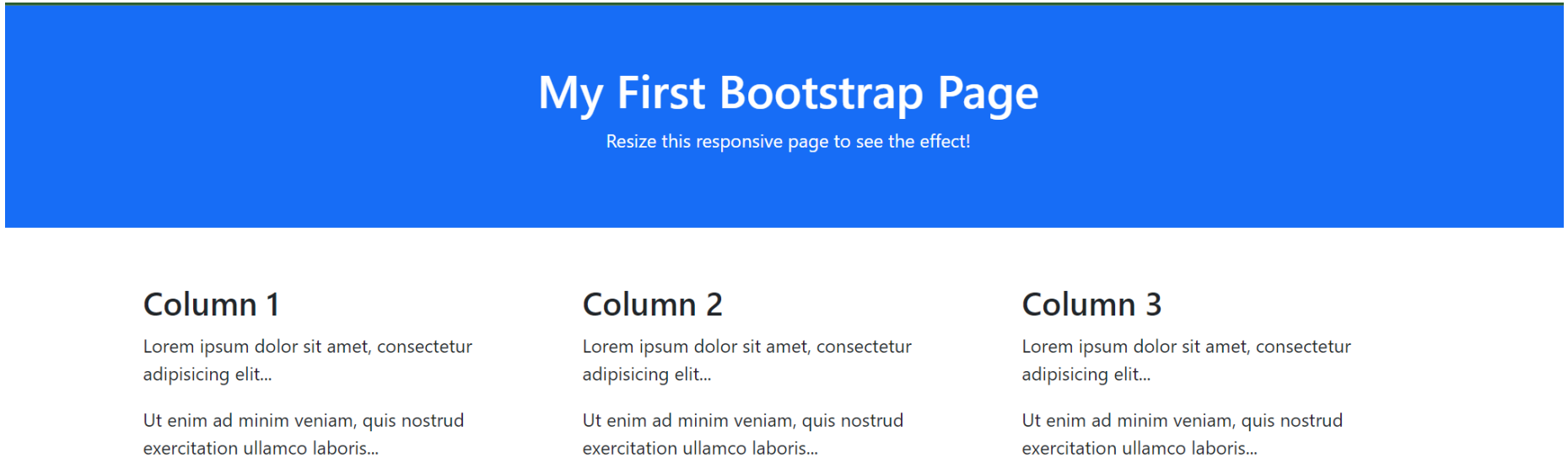
4. If you plan on using Bootstrap's JavaScript plugins, you will also need to include the link to the jQuery JavaScript library and the Popper.js library, which are required dependencies for some of the Bootstrap plugins.

5. Save your HTML file and open it in your browser to confirm that Bootstrap 5 is working correctly.

https://getbootstrap.com/docs/5.3/getting-started/download/

Adding Bootstrap to your website Example

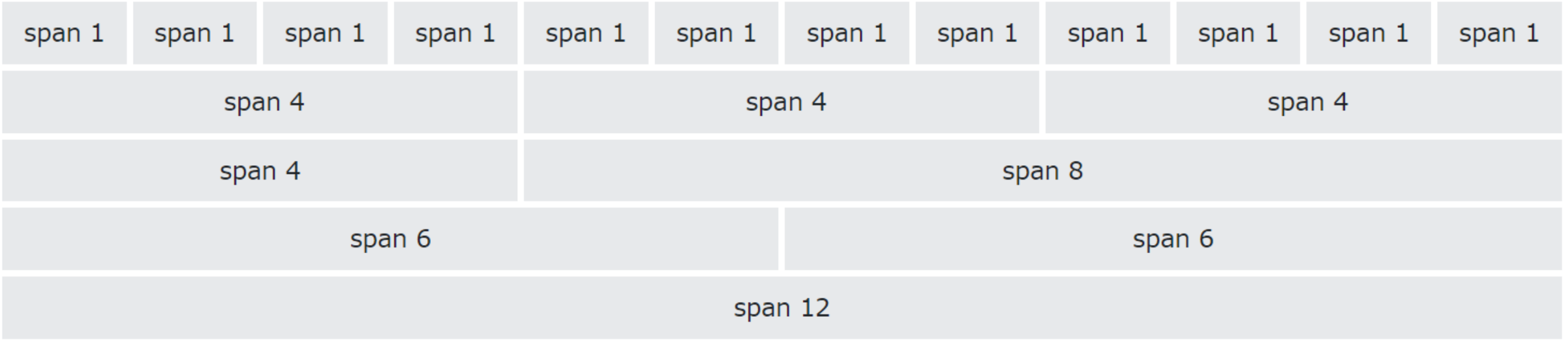
DEMO from 1-Start/index.html



https://www.w3schools.com/bootstrap5/tryit.asp?filename=trybs_default&stacked=h

Bootstrap GRID system

- Bootstrap’s grid system allows up to 12 columns across the page
  - If you don’t want to use all 12 columns individually, you can group the columns together to create wider columns
  - Bootstrap’s grid system is responsive, and the columns will re-arrange automatically depending on the screen size



Bootstrap Grid System

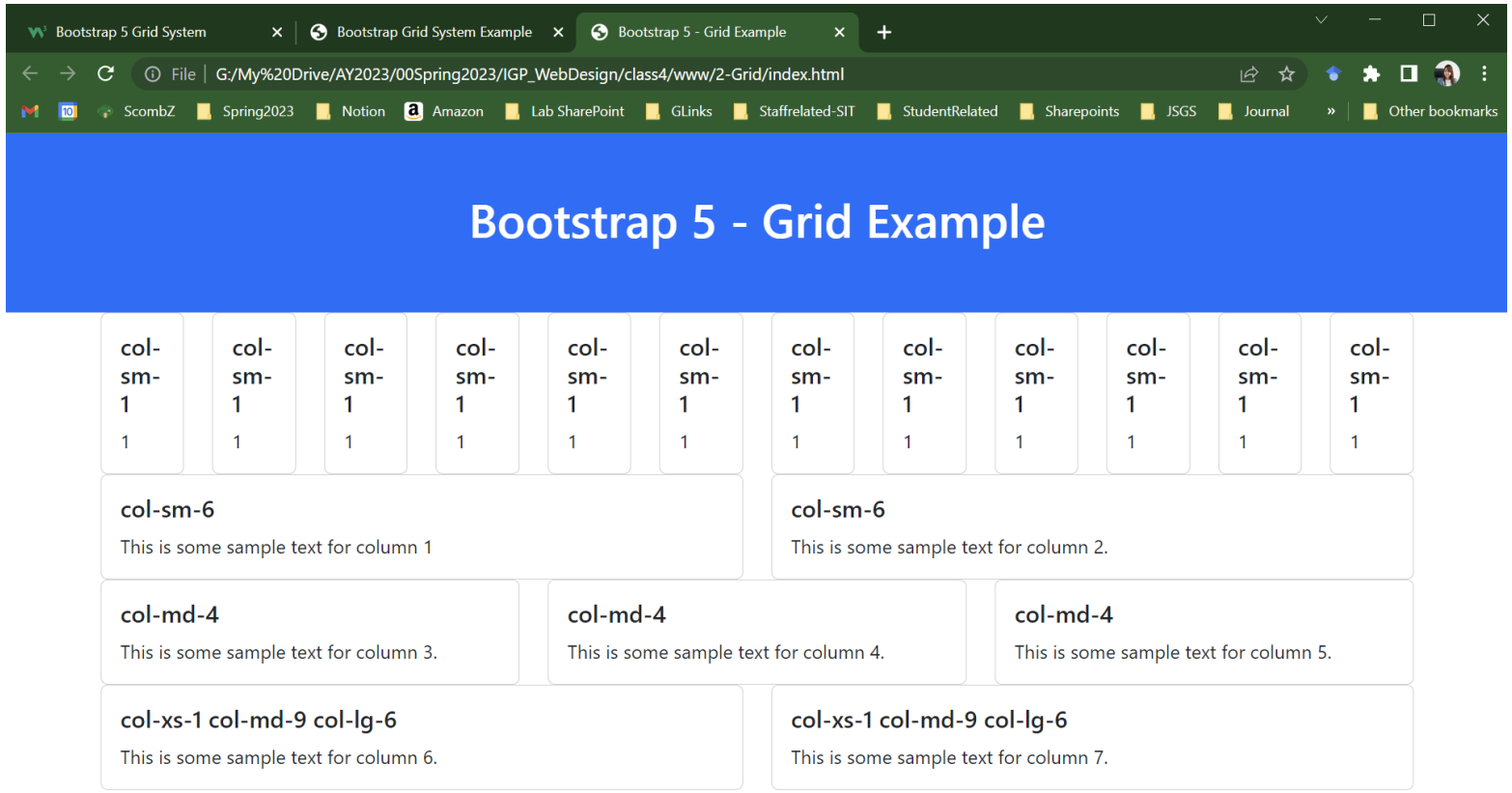
- Bootstrap grid system is based on a 12-column layout
- Each column is represented by a `div` element with a class of `col-`
- Column width can be adjusted by adding multiple classes to a column element
- Content is wrapped in a `container` element with a class of `container` or `container-fluid`
- Rows are created by adding `div` elements with a class of `row` inside the container
- Columns are added to rows to create the layout
- Breakpoint-specific column classes are used to specify how many columns a column should take up at different screen sizes.

The Bootstrap 5 grid system has six classes:

`.col-` (extra small devices - screen width less than 576px)
`.col-sm-` (small devices - screen width equal to or greater than 576px)
`.col-md-` (medium devices - screen width equal to or greater than 768px)
`.col-lg-` (large devices - screen width equal to or greater than 992px)
`.col-xl-` (xlarge devices - screen width equal to or greater than 1200px)
`.col-xxl-` (xxlarge devices - screen width equal to or greater than 1400px)

Demo Grid System

DEMO from 2/index.html



Combination of grid classes

- The classes can be combined to create more dynamic and flexible layouts

```
<div class="col-1 col-md-9 col-lg-6">  </div>
```

- The code refers to a Bootstrap grid system, which allows for flexible and responsive layouts on web pages.
- The "`col`" classes define the width of the column within the grid.
- The numbers after "`col`" indicate the number of grid columns the element should occupy.
- In this case,
  - `col-1` : the element will take up `1` column on extra-small screens
  - `col-md-9` : the element will take up `9` columns on medium screens
  - `col-lg-6` : the element will take up `6` columns on large screens.
- This provides a responsive layout that adjusts to the screen size of the device being used.

Grid across different screen size

https://www.w3schools.com/bootstrap5/bootstrap_grid_system.php

	Extra small (<576px)	Small (>=576px)	Medium (>=768px)	Large (>=992px)	Extra Large (>=1200px)	XXL (>=1400px)
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`	`.col-xxl-`
Grid behaviour	Horizontal at all times	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints
Container width	None (auto)	540px	720px	960px	1140px	1320px
Suitable for	Portrait phones	Landscape phones	Tablets	Laptops	Laptops and Desktops	Laptops and Desktops
# of columns	12	12	12	12	12	12
Gutter width	1.5rem (.75rem on each side of a column)	1.5rem (.75rem on each side of a column)	1.5rem (.75rem on each side of a column)	1.5rem (.75rem on each side of a column)	1.5rem (.75rem on each side of a column)	1.5rem (.75rem on each side of a column)
Nestable	Yes	Yes	Yes	Yes	Yes	Yes
Offsets	Yes	Yes	Yes	Yes	Yes	Yes
Column ordering	Yes	Yes	Yes	Yes	Yes	Yes

Bootstrap Tables

- Table classes: Bootstrap provides a number of classes to style your tables, such as `.table` to create a basic table and `.table-striped` to add striped rows.
- Table headers: You can add table headers using the `<thead>` element, and table footers using the `<tfoot>` element.
- Table borders: You can add borders to your tables using the `.table-bordered` class.
- Table hover: You can add a hover effect to your tables using the `.table-hover` class.
- Table responsiveness: You can make your tables responsive by wrapping them in a `.table-responsive div`. This will add horizontal scrolling to the table when viewed on smaller screens.
- Table contextual classes: You can use contextual classes like `.table-primary`, `.table-secondary`, `.table-success`, `.table-danger`, `.table-warning`, `.table-info`, `.table-light`, and `.table-dark` to add different colors to your table rows.
- Table sizing: You can adjust the size of your table using the `.table-sm` class for small tables and `.table-lg` for larger tables.
- Table caption: You can add a caption to your table using the `<caption>` element.

Bootstrap Tables

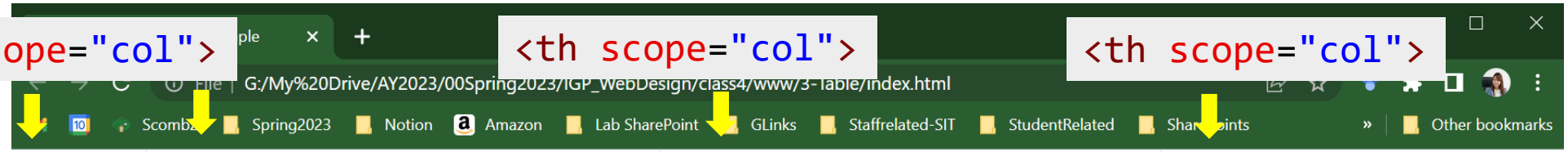
DEMO from 3 – tables/index.html

```
<table class="table table-striped table-bordered table-hover">
```

```
<th scope="col">
```

```
<th scope="col">
```

```
<th scope="col">
```




```
<th scope="row">
```

#	First Name	Last Name	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

Scope attribute

- In HTML, the `scope` attribute is used in conjunction with the `th` (table header) element to define whether the header cell applies to a row or a column of a table.
- The scope attribute can have two possible values:
- `row`: Indicates that the header cell applies to one or more cells in the same row.
- `col`: Indicates that the header cell applies to one or more cells in the same column.
- By default, if the `scope` attribute is not specified, it is assumed to be `col`.
- For example, in a table with multiple rows and columns, if you want to define the header for a specific column, you can use the `scope="col"` attribute in the `th` element. Similarly, if you want to define the header for a specific row, you can use the `scope="row"` attribute in the `th` element.
- Bootstrap uses the scope attribute to help with accessibility by indicating the scope of each header cell to assistive technology like screen readers.

Bootstrap contextual classes

- Bootstrap 5 provides contextual classes to change the color and style of an element based on the context or intent.
- The contextual classes include primary, secondary, success, danger, warning, info, and light, dark, and white.
- These classes can be applied to various HTML elements such as buttons, links, badges, alerts, and more.
- The classes can be used to emphasize or de-emphasize certain elements or convey different meanings, such as success or danger.
- The contextual classes can also be combined with other classes to create more complex styles and effects.
- Additionally, Bootstrap 5 provides utility classes to quickly apply contextual classes to various elements, such as bg-primary or text-danger.

Contextual classes

DEMO from 4 – context/index.html

`.bg-primary`: adds primary background color to an element.

`.bg-secondary`: adds secondary background color to an element.

`.bg-success`: adds a background color to show success message.

`.bg-danger`: adds a background color to show error or danger message.

`.bg-warning`: adds a background color to show warning message.

`.bg-info`: adds a background color to show informational message.

`.text-primary`: adds primary color to the text of an element.

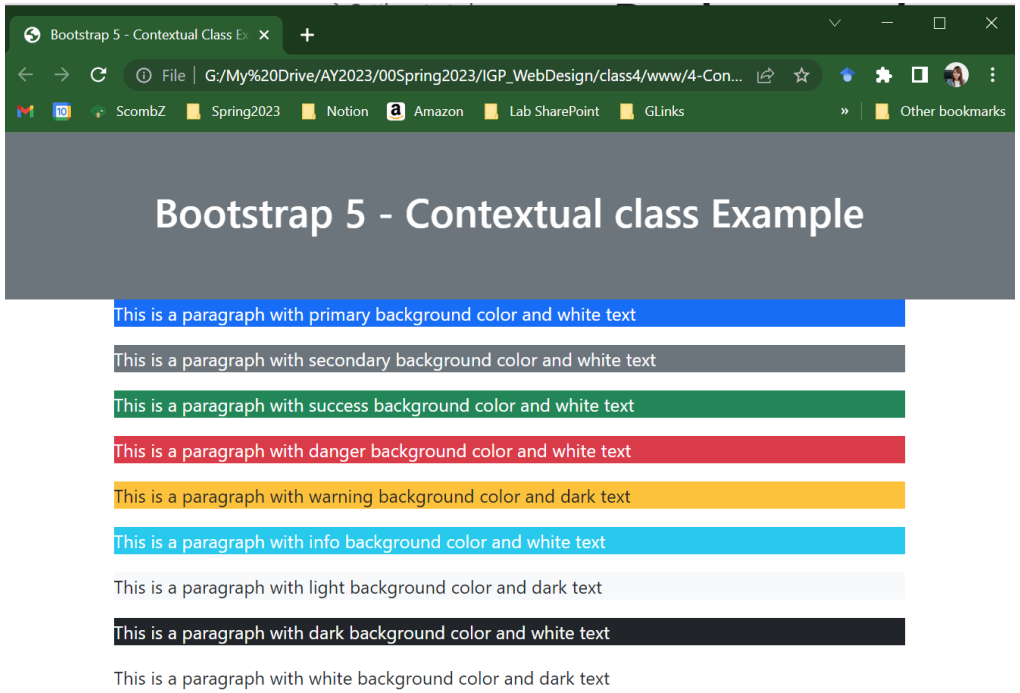
`.text-secondary`: adds secondary color to the text of an element.

`.text-success`: adds a color to show success message in the text of an element.

`.text-danger`: adds a color to show error or danger message in the text of an element.

`.text-warning`: adds a color to show warning message in the text of an element.

`.text-info`: adds a color to show informational message in the text of an element.



Bootstrap Images

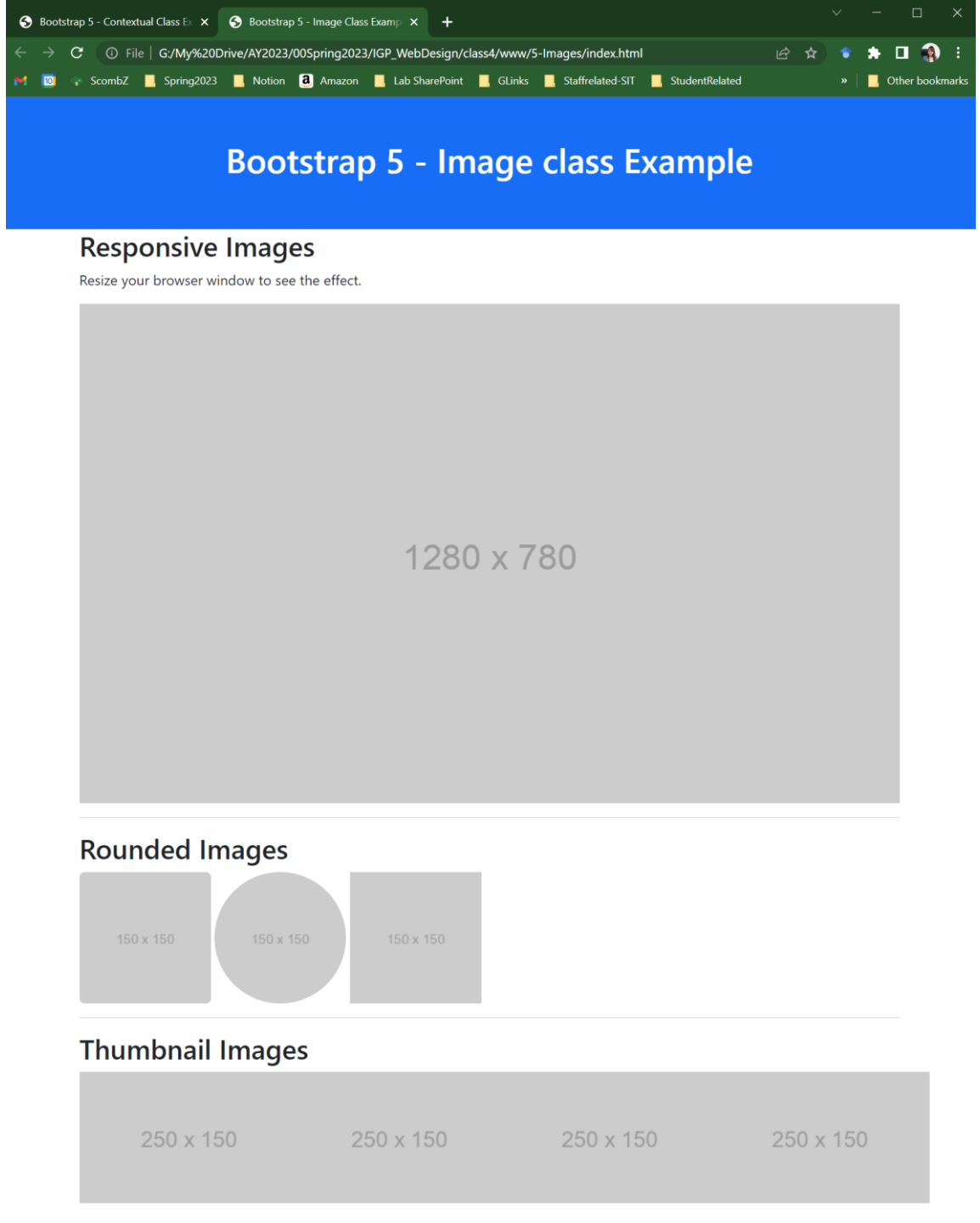
- Bootstrap 5 provides several classes to help you style images on your webpage. These classes are:

1. `img-fluid`: This class makes the image responsive, meaning it will adjust its size based on the size of the screen or container it's displayed in.
2. `rounded`: This class adds rounded corners to the image.
3. `rounded-circle`: This class creates a circular image.
4. `rounded-0`: This class removes any rounded corners from the image.
5. `float-start` or `float-end`: These classes float the image to the left or right, respectively, allowing text to wrap around it.
6. `mx-auto`: This class centers the image horizontally within its container.

- You can use these classes individually or in combination to achieve the desired look for your images.

Bootstrap Images

DEMO from 5 – image/index.html



Bootstrap Carousel

- Bootstrap 5 carousel is a slideshow component that allows you to display images, text, or custom content in a rotating manner. The carousel is fully responsive and supports swipe gestures on touch-enabled devices.

- To use Bootstrap 5 carousel, you need to include the required CSS and JavaScript files, which can be downloaded from the official Bootstrap website or included via a CDN.

Bootstrap Carousel

Code explanation

DEMO from 6 – Carousel/index.html

- The div with the ID `carouselExampleControls` represents the carousel container. It has the carousel class and the `data-bs-ride` attribute set to carousel to enable the automatic sliding functionality.
- Inside the container, there is a `div` with the class `carousel-inner`, which holds the carousel items (slides).
- Each slide is represented by a `div` with the class `carousel-item`. The first slide has an additional active class, indicating it is the initial slide shown when the carousel loads.
- Inside each slide, there is an `img` tag with a placeholder image URL. You can replace these URLs with actual image URLs of your choice. The class="`d-block w-100`" ensures the image fills the entire width of the slide.
- After the carousel items, there are two control buttons: `carousel-control-prev` and `carousel-control-next`. These buttons allow users to navigate to the previous and next slides.
- The `data-bs-target` attribute of the control buttons is set to the ID of the carousel container (`#carouselExampleControls`).
- The control buttons contain `span` elements with appropriate classes to display the previous and next icons.

https://www.w3schools.com/bootstrap5/bootstrap_carousel.php

Bootstrap Dropdown

- Dropdowns are a common UI element in web applications and Bootstrap 5 provides a simple and flexible way to add them to your site.

- A dropdown can be a button or a link that reveals a menu or list of options when clicked or hovered over.

https://www.w3schools.com/bootstrap5/bootstrap_dropdowns.php

Bootstrap Dropdown

`dropdown`: the container element for the entire dropdown component.

`dropdown-toggle`: the button or link that triggers the dropdown menu to appear.

`dropdown-menu`: the container for the dropdown menu items.

`dropdown-menu-start`: aligns the dropdown menu to the start (left) of the dropdown toggle button.

`dropdown-menu-end`: aligns the dropdown menu to the end (right) of the dropdown toggle button.

`dropdown-menu-lg-start` or dropdown-menu-lg-end: aligns the dropdown menu to the start or end on large screens only.

`dropdown-header`: styles the dropdown menu item as a header.

`dropdown-divider`: creates a horizontal line to divide the dropdown menu items.

`dropdown-item`: a single dropdown menu item.

`dropdown-item-text`: a non-clickable item for displaying text in the dropdown menu.

`dropdown-item-text`: a non-clickable item for displaying text in the dropdown menu.

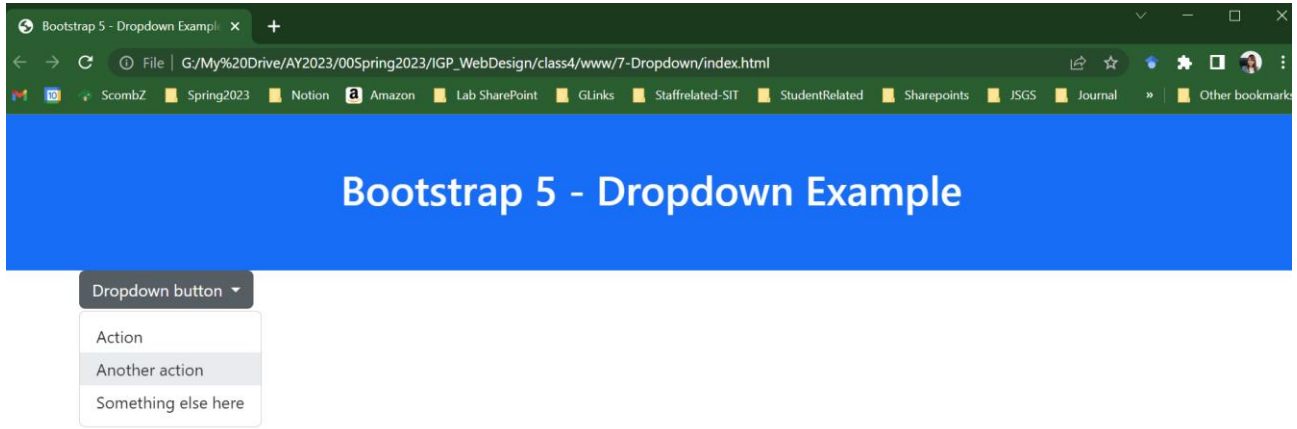
`active`: applies an active state to a dropdown menu item.

`disabled`: disables a dropdown menu item.

Bootstrap Dropdown

DEMO from 7 – Dropdown/index.html

- Add the dropdown class to a `<div>` element.
- Inside the `<div>,` add a button with the `dropdown-toggle` class and the `data-bs-toggle="dropdown"` attribute. This will serve as the trigger for the dropdown.
- Add a `<ul>` element with the `dropdown-menu` class inside the `<div>`. This will contain the dropdown items.
- Inside the `<ul>,` add `<li>` elements with the `dropdown-item` class to represent each item in the dropdown.



Bootstrap Offcanvas

- Offcanvas is a side content overlay that slides in from the edge of the viewport
- It is a flexible and versatile component that can be used for various purposes such as navigation, shopping cart, search bar, etc.
- It can be triggered by clicking a button, a link, or programmatically with JavaScript
- Offcanvas has built-in CSS classes that allow for customization and positioning of the overlay and the content inside it
- It can be easily implemented in Bootstrap 5 with just a few lines of code

Bootstrap Offcanvas

DEMO from 8 – Offcanvas/index.html

Code explanation

- A button with the class `btn btn-primary` is used to trigger the `offcanvas`. It has two attributes `data-bs-toggle` and `data-bs-target` which are used to identify and control the `offcanvas`.
- The `offcanvas` content is placed inside a `<div>` with class `offcanvas`. It also has an id attribute that is used to identify it in the `data-bs-target` attribute of the button.
- The `offcanvas-start` class is used to position the `offcanvas` on the left side of the screen. You can also use `offcanvas-end` to position it on the right side.
- The `offcanvas-header` contains the title of the offcanvas and a close button with the class `btn-close`.
- The `offcanvas-body` contains the actual content of the offcanvas. You can place any HTML content inside it.

Use the `.offcanvas-start|end|top|bottom` to position the `offcanvas` to the left, right, top or bottom:

Bootstrap Accordion

- An `accordion` is a component that displays collapsible content, with only one item visible at a time, and other items hidden.
- Each item in an accordion usually consists of a header, a body, and a toggle button to open or close the body.
- Bootstrap 5 provides classes to create an accordion component easily, without requiring any JavaScript code.
- The `accordion` class is used to create the container for the accordion.
- The `accordion-item` class is used to create an individual item in the accordion.
- The `accordion-header` class is used to create the header of an accordion item.
- The `accordion-body` class is used to create the body of an accordion item.
- The `accordion-button` class is used to create the toggle button to open or close the body of an accordion item.
- By default, only one item can be open at a time, but you can allow multiple items to be open simultaneously by adding the `allow-multiple` attribute to the `accordion` container.

Bootstrap Accordion

DEMO from 9 – Accordion/index.html

Code explanation

- Container `div` with the class `"accordion"` that has an ID of `"accordionExample”` is where all the accordion items will be placed.
- Each accordion item is represented by a div with the class `"accordion-item"`. Inside this `div`, we have an `"accordion-header"` and an `"accordion-collapse" div`.
- The `accordion-header` contains a button element with the class "accordion-button". This button has several data attributes that are used by Bootstrap to toggle the accordion item when it is clicked.
- The `"data-bs-target"` attribute of the button specifies the ID of the "`accordion-collapse`" div that is associated with this header. This is what allows Bootstrap to know which content should be toggled.
- The `"data-bs-parent"` attribute of the "`accordion-collapse`" div specifies the ID of the container div that contains all the accordion items. This allows Bootstrap to know which items are part of the same accordion.

Other popular classes

- Alert classes: .alert, .alert-primary, .alert-secondary, .alert-success, .alert-danger, .alert-warning, .alert-info, .alert-light, .alert-dark. Used to display different types of alerts to the user.
- Badge classes: .badge, .badge-primary, .badge-secondary, .badge-success, .badge-danger, .badge-warning, .badge-info, .badge-light, .badge-dark. Used to display a small badge with a numeric or text value.
- Button classes: .btn, .btn-primary, .btn-secondary, .btn-success, .btn-danger, .btn-warning, .btn-info, .btn-light, .btn-dark, .btn-link. Used to create different styles of buttons.
- Form control classes: .form-control, .form-control-lg, .form-control-sm, .form-range. Used to style form controls like inputs, select boxes, and sliders.
- Grid classes: .container, .container-fluid, .row, .col, .col-*. Used to create a responsive grid system for laying out content.
- Navbar classes: .navbar, .navbar-expand-*, .navbar-light, .navbar-dark. Used to create a navigation bar at the top of the page.
- Spinner classes: .spinner-border, .spinner-grow. Used to display loading spinners.
- You can find more information about these classes and others in the Bootstrap 5 documentation or on various online resources.

Midterm project

Web Design and Programming

Spring 2024

About me website

- Create a website that introduce yourself using only HTML and CSS
- **Requirements**
  - The website will acts as your **portfolio**, so please include the detail about your photos, previous works, showcase (selection of class assignment).
  - **Responsive** for mobile, tablet, and regular desktop
  - Accessible for public, therefore, do not use copyright files
- **References**
  - Plenty of references about HTML tags, CSS out there. The good one is W3school. Keep on studying!

Rubric

Design 10%	**Color**	Consistent across pages
	Fonts	Consistent across pages, easy to read, point size varies appropriately
	Graphic	Graphics are related to the theme / content, high quality, enhance reader interest or understanding
	Responsiveness	Responsive for at least 3 breakpoints (mobile, tablet, PC)
	Layout	The Web site has an exceptionally attractive and usable layout. It is easy to locate all-important elements. White space, graphic elements and/or alignment are used effectively to organize material.
	Navigation	Links for navigation are clearly labelled, consistently places, easy to find (easy to move forward and backward) and do not become lost inside the page
	Image	All images, especially those that are used for navigation, have an ALT tag that describes the image and its link so people who are visually impaired can use the Web site well.
	Copyright	No material is included from websites that stated that permission is required
Code 9%	**Code**	Appropriate comments for all codes, HTML codes are used for semantic meaning, CSS are for decoration, appropriate JS is included
Presentation 5%	**Overall**	You will need to do a quick presentation (60sec) of your website. Clearly explain all important codes with confident + use pointer to explain Appropriately answer questions when asked
Participation 1%	**Participation**	Listening to other people presentation & ask question at least to 2 presentations

Vote for the best design: Gold, Silver, Bronze
Additional score for Gold (+1%), Silver (+0.5%), Bronze (+0.25%)

Work on homework
Work on midterm project