

CS472 WAP

Lecture 2: CSS

Maharishi International University - Fairfield, Iowa

All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi International University.



Wholeness Statement

The basics of CSS give different visual styles to HTML elements, changing their preset default appearance.

CSS was a natural evolution to HTML that enhances the flexibility and sophistication of HTML for the specialized function of visual styling. *Nature evolves by creating encapsulated subsystems to handle specialized functionalities*

Content vs. Presentation


- HTML is for content, the information on the page
- CSS is for presentation, how to display the page
- Keeping content separate from presentation is a very important web design principle
- If the HTML contains no styles, its entire appearance can be changed by swapping `.css` files
- <http://csszengarden.com/>

Ways to Add CSS

External CSS: This is the preferred method and involves creating a separate CSS file with a .css extension.

Style Tag in <head>: Include CSS styling directly within the <head> section of your HTML document using the <style> tag

Inline Style Attribute: Adding CSS styling directly to an HTML element using the style attribute.



Method	Pros	Cons
External CSS	Most maintainable, reusable	Requires extra file
Style Tag in Head	No extra files	Less maintainable for complex sites
Inline Style Attribute	Very specific	Makes HTML code messy, hard to maintain





The bad way to produce styles

- Tags such as **strong**, **em**, **u**, and **font** are discouraged in strict HTML.

```
<p>  
  <font face="Arial">Welcome to Greasy Joe's.</font>  
  You will <strong>never</strong>, <em>ever</em>,  
  <u>EVER</u> beat  
  <font size="+4" color="red">OUR</font> prices!  
</p>
```

Welcome to Greasy Joe's. You will never, ever, EVER beat **OUR** prices!

Bad Practices, why?

Internal/Embedding style sheets



```
<head>
  <style type="text/css">
    p { font-family: sans-serif;
        color:red; }
    h2 { background-color: yellow; }
  </style>
</head>
```

Inline styles: the style attribute



```
<p style="font-family: sans-serif; color: red;">This
is a paragraph</p>
```

Note: It has higher precedence than embedded or linked styles

Cascading Style Sheets (CSS): `<link>`



- CSS describes the appearance and layout of information on a web page
 - (as opposed to HTML, which describes the content of the page)
- Can be embedded in HTML or placed into separate **.css** file (preferred)

```
<head>
```

```
  <link href="style.css" type="text/css" rel="stylesheet"/>
```

```
</head>
```


Basic CSS rule syntax

- A **CSS** file consists of one or more rules
- A rule's selector specifies HTML element(s) and applies style properties.
- To add a comment, we use: `/* */`
- The `*` **universal** selector, selects all elements

Syntax:

```
selector {  
    property: value;  
    property: value; ...  
}
```

Example:

```
p {  
    font-family: sans-serif;  
    color: red;  
}
```



CSS properties for colors

```
p {  
    color: white;  
    background-color: blue;  
}
```

This paragraph uses the style above.

property	description
color	color of the element's text
background-color	color that will appear behind the element

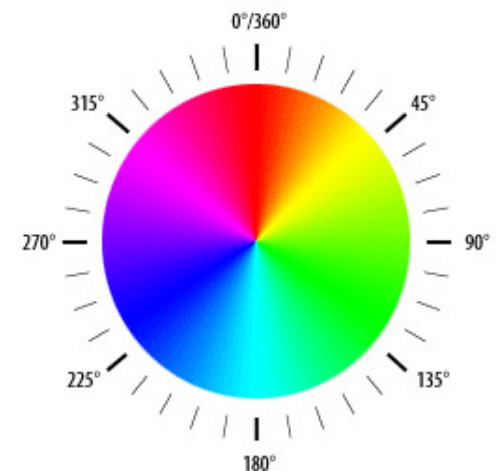
Specifying colors

- **Color names:** aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow
- **RGB & RGBA codes:** red, green, and blue values from 0 to 255
- **HEX codes:** RGB values in base-16 from 00 (none) to FF (full)
- **HSL & HSLA codes:** HSL stands for hue, saturation, and lightness
- Hue is degree on color wheel (from 0 to 360) - 0 (or 360) is red, 120 is green, 240 is blue.

Hsla demo <https://codepen.io/kman/pen/KwapPZ>

Google css color picker

```
h1 { color: red; }
h2 { color: rgb(128, 0, 196); }
h3 { color: rgba(128, 0, 196, 0.5); }
h4 { color: #FF8800; }
h5 { color: #F80; }
h6 { color: hsla(120, 60%, 70%, 0.3); }
```





CSS properties for fonts

property	description	Values
font-family	which font will be used	serif or “Courier New”
font-size	how large the letters will be drawn	A unit value, percentage, or named value
font-style	used to enable/disable italic style	normal(default), italic, ...
font-weight	used to enable/disable bold style	normal(default), bold, bolder,...
font	Sets all font properties	style weigh size family
Complete list of font properties		

CSS properties for fonts

```
h1{ /* which font will be used */
    font-family: Georgia;
}
h2 { /* enclose multi-word font names in quotes */
    font-family: "Courier New";
}
h3 { /* can specify multiple fonts from highest to
lowest priority */
    font-family: Garamond, "Times New Roman", serif;
}
```

- If the first font is not found on the user's computer, the next is tried
 - Placing a generic font name at the end of your font-family value ensures that every computer will use a valid font
 - CSS generic font names: serif, sans-serif, cursive, fantasy, monospace
-
- ▶ Serifed fonts easier to read on printed pages, hard to read on computer screens

Size Units



- **Units:** pixels (**px**), point (**pt**), m-size (**em**), vw, vh
 - **px** specifies a number of pixels on the screen
 - **pt** specifies number of points, where a point is 1/72 of an inch on screen
 - **em** relative to the font-size of the element
(2em means 2 times the size of the current font)
 - **vw** Equal to 1% of the width of the viewport's initial containing block
 - **vh** Equal to 1% of the height of the viewport's initial containing block.
- **Vague font sizes:** xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger
- <https://webflow.com/blog/how-and-why-to-use-vh-and-vw-in-webflow>

The `list-style-type` property



- **`none`**: No marker
- **`disc`** (default), **`circle`**, **`square`**
- **`decimal`**: 1, 2, 3, etc.
- **`decimal-leading-zero`**: 01, 02, 03, etc.
- **`lower-roman`**: i, ii, iii, iv, v, etc.
- **`upper-roman`**: I, II, III, IV, V, etc.
- **`lower-alpha`**: a, b, c, d, e, etc.
- **`upper-alpha`**: A, B, C, D, E, etc.
- **`lower-greek`**: alpha, beta, gamma, etc.

```
ol { list-style-type: lower-roman; }
```

```
i. first item  
ii. second item  
iii. third item
```

Explore the CSS Rules

- [CSS Colors](#)
- [CSS Units](#)
- [Text](#)
- [Fonts](#)
- [Dimensions](#), [Padding](#), [Borders](#), [Margins](#)
- [Align](#)



CSS Selectors

Element Selectors

p {} targets all paragraphs on the page <p>

Class Selectors

.big{} targets all elements with a class attribute of class="big"

ID Selectors

#main{} targets one element with id="main"

Attribute Selectors

[lang=en]{} targets all elements with an attribute of lang="en"

Pseudo-Classes/Elements

a:hover{} targets all anchor elements in hover state

p::first-line {} targets the first line of the text in all <p> elements



css Combinators

Multiple elements

div, p {} targets all <div> and <p> elements

Descendent combinator (space)

div p {} targets all paragraphs that are inside a <div> element

Direct child combinator

div > p {} targets all paragraphs that are direct children to a <div> element



The HTML **class** and **id** attribute

- **id attribute** allows you to give a unique ID to any element on a page
 - Each ID must be unique; can only be used once in the page
- **class attribute** is used to group some elements and give a style to only that group
 - unlike an id, a class can be reused as much as you like on the page

class vs id examples



```
<p id="mission">Our mission is to provide the most</p>
<p class="special">See our spectacular spatula specials</p>
<p class="special shout">Today only, satisfaction guaranteed</p>
```

```
#mission {
  font-style: italic;
  color: #000000;
}
.special { /* any element with class="special" */
  background-color: yellow;
  font-weight: bold;
}
.shout { /* only p elements with class="shout" */
  color: red;
  font-family: cursive;
}
```

Our mission is to provide the most

See our spectacular spatula specials

Today only, satisfaction guaranteed



id/class naming

- focus on the semantics and meaning of the content vs appearance
- Good example:
 - warningMsg
 - errorMsg
- Bad example: redtext, bigfont
 - if change style later, it doesn't make sense to be called redtext.

CSS Combinators/contextual selectors

```
<div>
  <p>Paragraph 1 in the div.</p>
  <p>Paragraph 2 in the div.</p>
  <section>
    <p>Paragraph 3 in the div.</p>
  </section>
</div>
```

```
<p>Paragraph 4. Not in a div.</p>
<p>Paragraph 5. Not in a div.</p>
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3 in the div.

Paragraph 4. Not in a div.

Paragraph 5. Not in a div.

```
div p {
  background-color: red;
}
```

```
div>p {
  background-color: yellow;
}
```

Body styles

- To apply a style to the entire body of your page, write a selector for the body element
- Saves you from manually applying a style to each element

```
body {  
  font-size: 16px;  
}
```



Inheriting styles

- Styles get inherited from containing elements
- **Not all properties are inherited** (notice link's color below)
 - E.g., margin

```
body {  
    font-family: sans-serif;  
    background-color: pink;  
}  
p {  
    color: green;  
}  
a {  
    text-decoration: underline;  
}  
h2 {  
    font-weight: bold;  
    text-align: center;  
}
```

CS472 Web Programming

This [course](#) provides a systematic introduction to programming interactive and dynamic web applications.

Styles that Conflict



```
/* select multiple elements separated by commas */
```

```
p, h1, h2 {
```

```
  color: green;
```

```
  background-color: grey;
```

```
}
```

```
/* when two styles set conflicting values for the same  
property, the latter style takes precedence */
```

```
h2 {
```

```
  background-color: blue;
```

```
}
```

```
<p> This paragraph will use background color grey! </p>
```

```
<h2> This heading will use background color blue! </h2>
```

This paragraph will use background color grey!

This heading will use background color blue!



Override Rules

```
<p class="RedColor BlueColor">
```

Lorem Ipsum

```
</p>
```

```
#YellowColor {
```

```
  color: yellow;
```

```
}
```

```
.BlueColor {
```


```
  color: blue;
```

```
}
```

```
.RedColor {
```

```
  color: red;
```

```
}
```



Lorem Ipsum



Style Specificity

- When multiple styles apply to an element and have the same origin precedence.
- The most specific one applies. If they have the same specificity, then the later one will be used.



```
<aside>  
  <p><em id="recent" class="awesome">Which awesome color?</em></p>  
</aside>
```

```
aside { color: gray; }  
p { color: green; }  
em { color: yellow; }  
.awesome { color: blue; }  
em.awesome { color: red; }  
#recent { color: black; }  
em#recent.awesome { color: orange; }
```



CSS Specificity

When multiple styles apply to an element, the most specific one applies. If they have the same specificity, then the latter one will be used.

1. Inline Styles `<div style="color: blue;"></div>`
2. ID Selectors `#header { color: green; }`
-  3. Class Selectors `.menu { color: red; }`
-  4. Attribute Selectors `[type="text"] { color: black; }`
5. Pseudo-Element `::first-letter { color: blue; }`
6. Pseudo-Classes `:hover { color: purple; }`
7. Element Type Selectors `p { color: orange; }`

When you use the [!important](#) rule, it will override the specificity rules.



Specificity and Conflicts

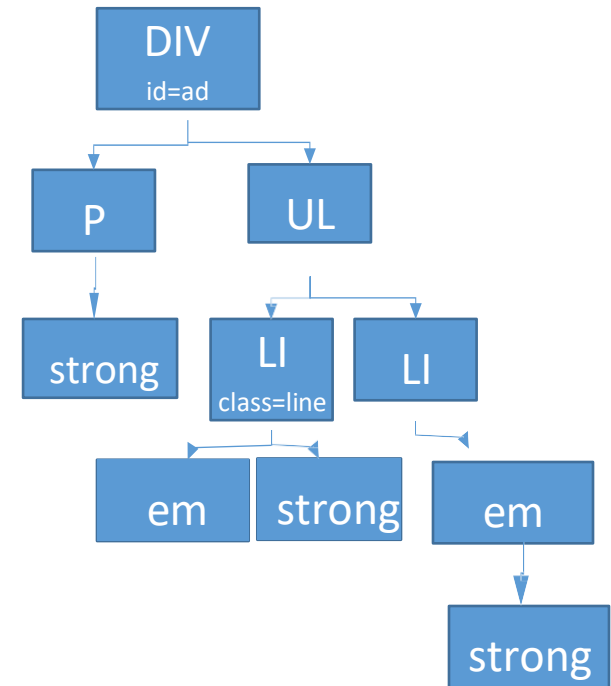
- Specificity- decide which one should win when two or more rules conflict.
- Rules: each rule's overall selector is given a score based upon approximately the following rules. The rule with the highest score wins if there's a conflict.
 - Any HTML element mentioned in the rule scores 1 point
 - Any class mentioned in the rule scores 10 points
 - Any ID mentioned in the rule scores 100 points
- Examples:
 - p.banner - 11
 - div.box > p - 12
 - body #logo .box p.banner - 122



Example

```
<div id="ad">
  <p>Shop at <strong>Hardwick's Hardware</strong></p>
  <ul>
    <li class="line">
      <em>The </em>
      <strong>best</strong>
      prices!
    </li>
    <li>
      <em>
        <strong>Act while supplies last!</strong>
      </em>
    </li>
  </ul>
</div>
```

```
ul > li { background-color: blue; }
li strong { color: red; }
li > strong { color: green; }
#ad li.line strong { text-decoration: underline; }
```



pseudo-classes

pseudo-elements







- A **pseudo-class** is used to define a special state of an element
 - Style an element when a user mouse's over it
 - Style visited and unvisited links differently
 - Style an element when it gets focus
- A CSS **pseudo-element** is used to style specified parts of an element
 - Style the first letter, or line, of an element
 - `::first-line`, `::first-letter`
 - Insert content (pseudo element) before, or after, the content of an element
 - `::before`, `::after`

```
selector:pseudo-class { property:value; }
```

```
selector::pseudo-element { property:value; }
```

CSS pseudo-classes pseudo-elements

class		description
:active		an activated or selected element
:focus		an element that has the keyboard focus
:hover		an element that has the mouse over it
:link		a link that has not been visited
:visited		a link that has already been visited
:nth-child(expr)		targets specific children of a given element
:first-child, :last-child		
:not(selector)		all elements that do not match the given CSS selector
::first-line		the first line of text inside an element
::first-letter		the first letter of text inside an element

[Complete list](#)

Examples pseudo-classes



```
/* unvisited link */
```

```
a:link { color: #FF0000; }
```

```
/* visited link */
```

```
a:visited { color: #00FF00; }
```

```
/* mouse over link */
```

```
a:hover { color: #FF00FF; }
```

```
/* click on a link */
```

```
a:active { color: #0000FF; }
```

More info and examples: [Pseudo-classes](#) and [Pseudo-elements](#)

Nesting Selectors

When the browser parses the nested selectors, it automatically adds **whitespace** between the selectors to create a new CSS selector rule.

```
parent-selector {  
    /* parent rule style properties */  
    child-selector {  
        /* child rule style properties */  
    }  
}
```



```
parent-selector {  
    /* parent rule style properties */  
}  
  
parent-selector child-selector {  
    /* child rule style properties */  
}
```



Main Point

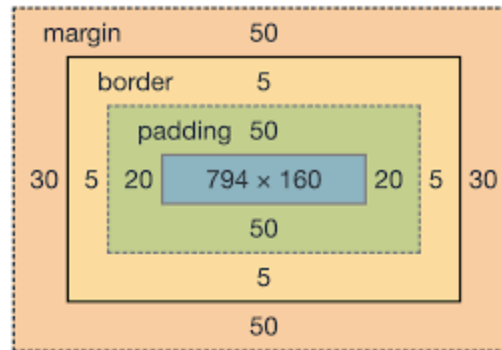
- The Cascading in CSS indicates that there are multiple levels of style sheets. More specific styles overwrite more general styles. We can be more specific by using Class selectors (can apply to multiple elements) and even more so with id selectors (individual elements) and context selectors.

Life is found in layers.

The CSS Box Model

Every HTML element is composed of:

- The actual element's content or **width/height**
- **Padding** between the content and the border
- **Border** around the element
- **Margin** between the border and other content



Total width = content width + L/R padding + L/R border + L/R margin

Total height = content height + T/B padding + T/B border + T/B margin



display

inline

do not start on a new line and only take up as much width as necessary.

- Cannot have width and height set explicitly.
- Margins and padding can only be applied horizontally, not vertically.

block

start on a new line and take up the full width available by default.

- Can have width and height set explicitly.
- Margins and padding are applied both vertically and horizontally.

inline-block

similar to inline elements in that they do not start on a new line, but they can have width and height set explicitly like block elements.

- Can have width and height set explicitly.
- Margins and padding are applied both vertically and horizontally.



position

static

This is the **default** value. Elements are positioned according to the normal flow of the document. The top, right, bottom, and left properties have no effect.

relative

The element is positioned relative to its normal position in the document flow. The top, right, bottom, and left properties can offset the element **from its normal position**.

absolute

The element is **removed** from the normal document flow. The top, right, bottom, and left properties specify the offsets **from its nearest positioned ancestor** (an ancestor with a position value other than static). If no such ancestor exists, it is positioned relative to the viewport.

fixed

The element is **removed** from the normal document flow. The top, right, bottom, and left properties specify the offsets **from the viewport**. It stays in the same place even when the page is scrolled.

sticky

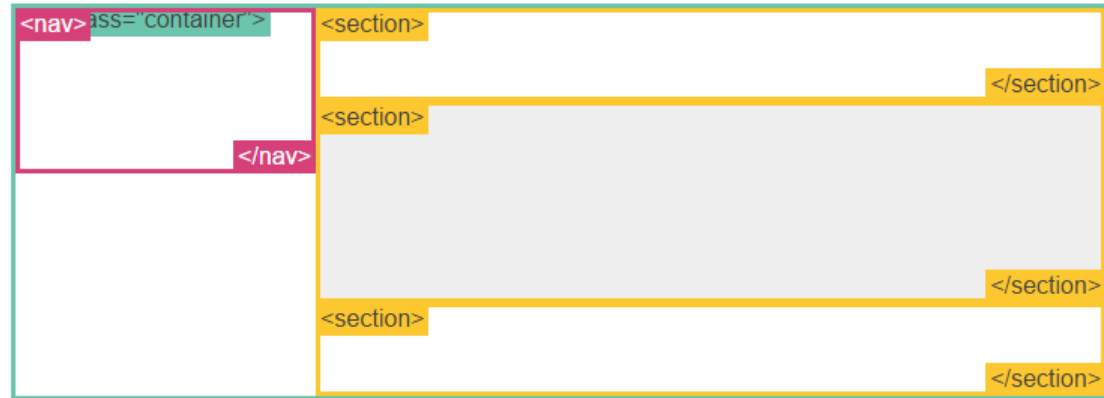


Position Layout Example

```
.container {  
  position: relative;  
}
```

```
nav {  
  position: absolute;  
  left: 0px;  
  width: 200px; }
```

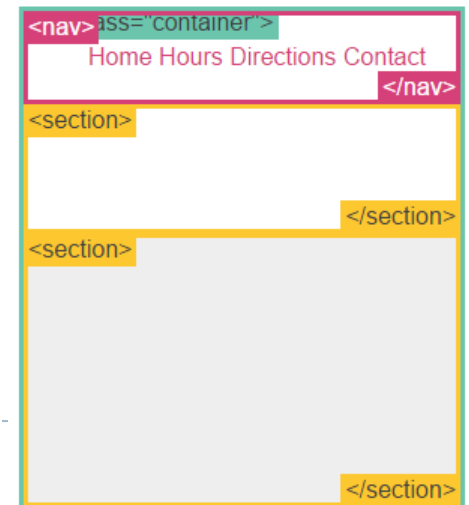
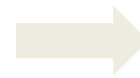
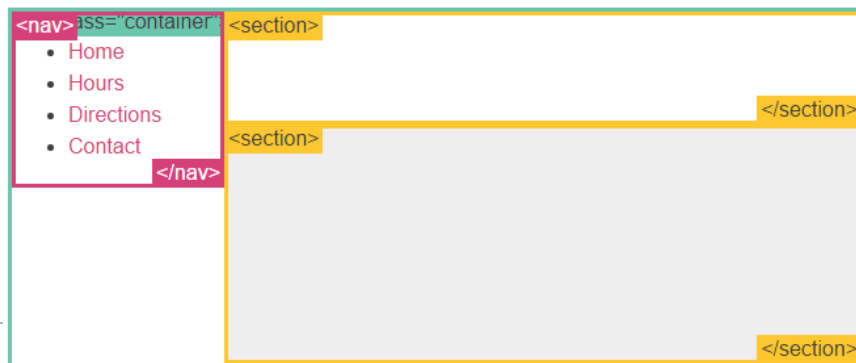
```
section {  
  margin-left: 200px;  
}
```



media queries

Responsive Design is the strategy of making a site that responds to the browser and device width.

```
@media screen and (min-width:600px) {  
  nav { position: absolute; left: 0px; width: 25%; }  
  section { margin-left: 25%; padding-top: 10px; }  
}  
@media screen and (max-width:599px) {  
  nav { position: static; }  
  nav li { display: inline-block; }  
}
```



Grid vs. Flexbox

CSS Grid creates a two-dimensional layout with rows and columns

CSS Flexbox to align elements on one dimension



Flexbox Layout

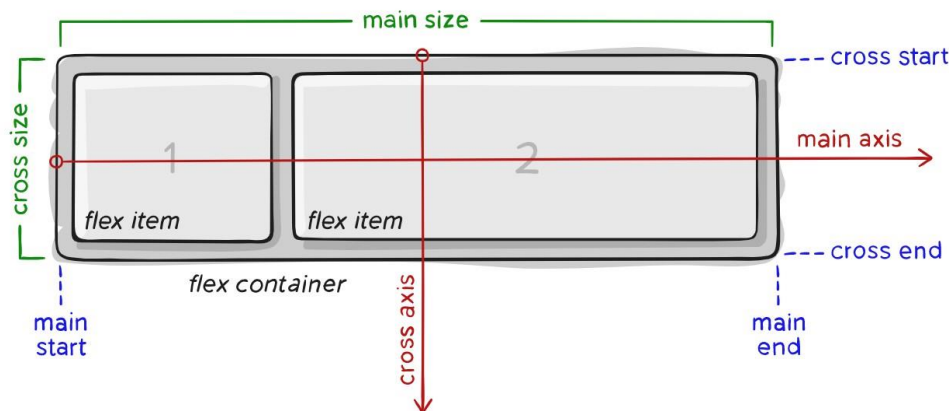
- responsive layout without float or positioning
- set *display* property to *flex* on the containing element
- direct child elements are automatically flexible items

display: flex;

[simple example](#)

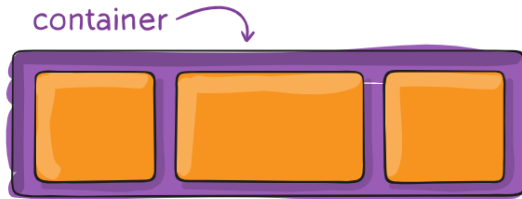
Flexbox Layout

- The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space.
- Flexbox is for one dimensional layout (row or column).
- set `display` property to `flex` on the containing element
`display: flex;`
- direct child elements are automatically flexible items



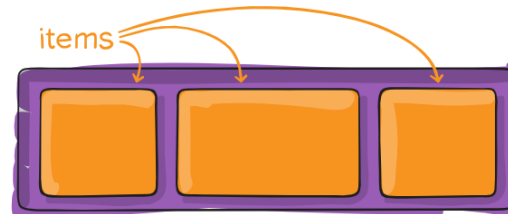
Flexbox properties

Properties for the Parent (flex container)



- `display: flex;`
- `flex-direction: row | row-reverse | column | column-reverse;`
- `flex-wrap: nowrap | wrap | wrap-reverse;`
- `flex-flow: <'flex-direction'> || <'flex-wrap'>`
- `justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;`
- `align-items: stretch | flex-start | flex-end | center | baseline;`
- `align-content: flex-start | flex-end | center | space-between | space-around | stretch;`

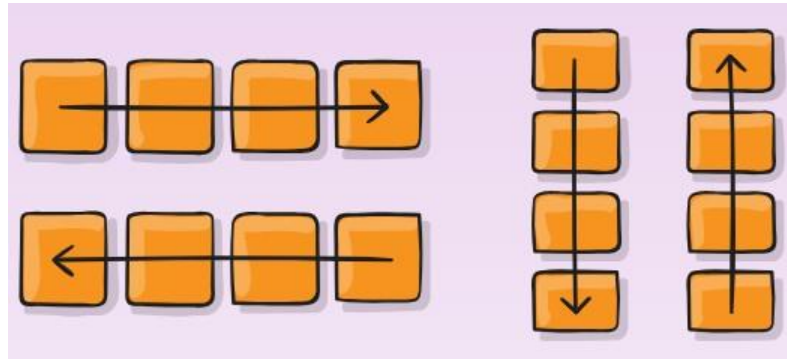
Properties for the Children (flex items)



- `order: <integer>; /* default is 0 */`
- `flex-grow: <number>; /* default 0 */`
- `flex-shrink: <number>; /* default 1 */`
- `flex-basis: <length> | auto; /* default auto */`
- `flex: none | [<'flex-grow'> <'flex-shrink'>? || <'flex-basis'>]`
- `align-self: auto | flex-start | flex-end | center | baseline | stretch;`

Flex container: flex-direction

- Specifies the direction of the flexible items inside a flex container
 - `row` (default): left to right in ltr; right to left in rtl
 - `row-reverse`: right to left in ltr; left to right in rtl
 - `column`: same as row but top to bottom
 - `column-reverse`: same as row-reverse but bottom to top



Flexbox Demo

[Home](#) [About us](#) [Products](#) [Shopping](#) [Contact us](#)

```
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About us</a></li>
    <li><a href="#">Products</a></li>
    <li><a href="#">Shopping</a></li>
    <li><a href="#">Contact us</a></li>
  </ul>
</nav>
```

```
ul {
  display: flex;
  /* flex-direction: row; */
  flex-wrap: wrap;
  justify-content: flex-end;
  li {
    list-style-type: none;
    margin-right: 10px;
  }
}
```

 [Try Flexbox Playground](#)



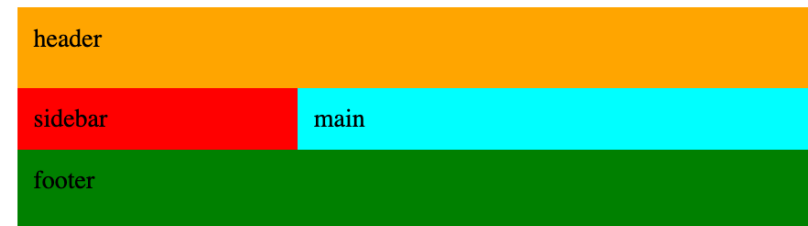
Grid Layout

- CSS Grid offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.
- A Grid Layout must have a parent element with the display property set to grid
- Direct child element(s) of the grid container automatically becomes grid items.

Grid Demo

```
.container { display: grid;  
  grid-template-columns: 150px auto auto auto;  
  grid-template-rows: 50px auto 50px;  
  grid-template-areas:  
    "header header header header"  
    "sidebar main main main"  
    "footer footer footer footer"; }
```

```
.cell-header { grid-area: header; }  
.cell-sidebar { grid-area: sidebar; }  
.cell-main { grid-area: main; }  
.cell-footer { grid-area: footer; }
```



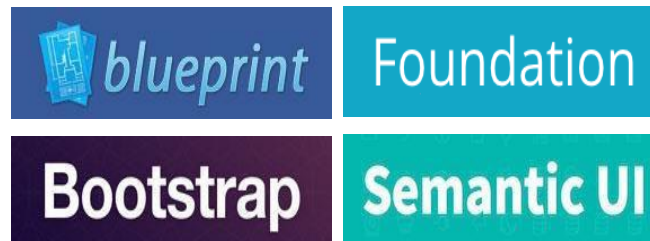
```
<div class="container">  
  <div class="cell-header">header</div>  
  <div class="cell-sidebar">sidebar</div>  
  <div class="cell-main">main</div>  
  <div class="cell-footer">footer</div>  
</div>
```

CSS Frameworks

Because CSS layout is so tricky, there are CSS frameworks out there to help make it easier. Here are a few if you want to check them out. Using a framework is only a good idea if the framework really does what you need your site to do.

They're no replacement for knowing how CSS works.

- [Blueprint](#)
- [Bootstrap](#)
- [Foundation](#)
- [SemanticUI](#)
- [Tailwind ...](#)



VS Code Extensions

HTML CSS Support

HTML to CSS autocompletion

Color Highlight



CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Changing Appearances

1. How a page is displayed is affected by both the HTML and the CSS
 2. Although every HTML tag has a default way of displaying, it can easily be changed with CSS and should never be the basis for using it. Instead use HTML tags based on meaning.
-

3. **Transcendental consciousness** is the field that underlies all differences.
4. **Impulses within the Transcendental field:** pure consciousness is a field of infinite possibilities that gives rise to the great diversity of the relative world, the same underlying content can appear as many expressions.
5. **Wholeness moving within itself:** In Unity Consciousness, one experiences that this unbounded diversity is nothing but the self.

