

The background of the slide features a complex, stylized graphic. It consists of a network of black lines that resemble circuit traces or a web of connections. These lines are set against a light gray background that contains faint, circular patterns resembling gears or concentric circles. The overall aesthetic is technical and modern.

# CSS

2020/21 COMP3322 Modern Technologies on WWW

# Contents

- About CSS
- CSS Syntax
- The concept of Cascade
- The box model
- More controls for developers
- CSS Layout
- Responsive web design

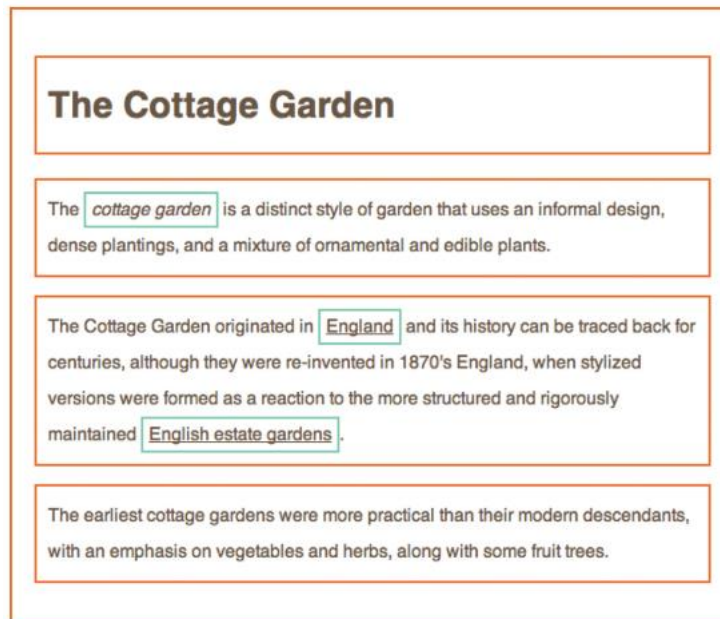
About CSS

# What is CSS?

- Cascading Style Sheet
  - CSS is a **W3C standard** for describing the presentation (or appearance) of HTML elements.
  - CSS allows us to **create rules** that specify how the content of an element should appear.
- CSS is a language in that it has its own syntax rules.
- With CSS, we can assign font properties, colors, sizes, borders, background images, and **even the position** of elements.

# What is CSS?

- The key to understanding how CSS works is to imagine that there is an **invisible box** around **every** HTML element.
  - CSS allows you to **create rules** that **control the way** that each individual box (and the contents of that box) is **presented**.



In this example, block level elements are shown with red borders, and inline elements have green borders.

The `<body>` element creates the first box, then the `<h1>`, `<h2>`, `<p>`, `<i>`, and `<a>` elements each create their own boxes within it.

Using CSS, you could add a border around any of the boxes, specify its width and height, or add a background color. You could also control text inside a box — for example, its color, size, and the typeface used.

## CSS Versions

- CSS was first proposed by Håkon Wium Lie on October 10, 1994.
- W3C published the CSS Level 1 Recommendation in Dec 1996.
- Two years later, the CSS Level 2 Recommendation was published.
- CSS Level 2.1 has gone through a decade of development, and only became official W3C Recommendation until June 2011.
- And to complicate matters even more, all through the last decade, during the same time the CSS2.1 standard was being worked on, a different group at the W3C was working on a CSS3 draft.
  - CSS3 is divided into several separate documents called "modules".
  - Different modules are in different statuses; some get to Recommendation status and some are still in Working Draft status

# Three Locations of Placing Styling

- CSS style rules can be located in three different locations.
  - Inline
  - Embedded
  - External
  - You can combine all 3!

# Inline Styles

```
<h1 style="color:blue;margin-left:30px;">This is a heading</h1>  
<p>This is a paragraph.</p>
```

**This is a heading**

This is a paragraph.

- Style rules placed **within an HTML element** via the “**style**” attribute
- An inline style only affects the element it is defined within and will **override** any other style definitions for the properties used in the inline style.
- Using inline styles is **generally discouraged** since they increase bandwidth and decrease maintainability.
- Inline styles can however be handy for quickly testing out a style change.



# Embedded Style Sheet

```
<head lang="en">
  <meta charset="utf-8">
  <title>Share Your Travels -- New York - Central Park</title>
  <style>
    h1 { font-size: 24pt; }
    h2 {
      font-size: 18pt;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <h1>Share Your Travels</h1>
  <h2>New York - Central Park</h2>
  ...

```

- Style rules placed within the **<style>** element inside the **<head>** element
- While better than inline styles, using embedded styles is also by and large discouraged.
- Since each HTML document has its own <style> element, **it is more difficult to consistently style multiple documents** when using embedded styles.

# External Style Sheet

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

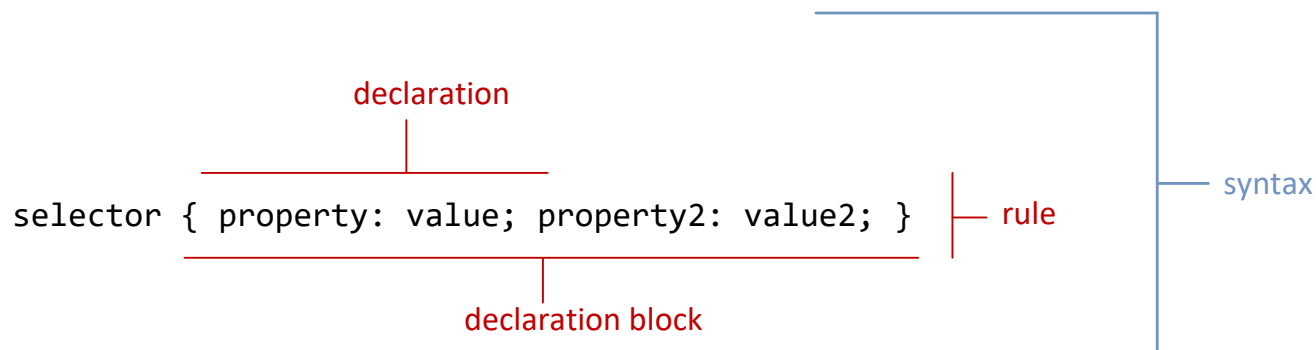
- Style rules placed in an external text file with the .css extension and the document references it from the **<link>** element.
  - The **href** attribute - This specifies the path to the CSS file
  - The **type** attribute - This specifies the type of document being linked to. The value should be **text/css**.
  - The **rel** attribute - This specifies the **relationship** between the HTML page and the file it is linked to. The value should be **stylesheet** when linking to a CSS file.
- This is by far the most common place to locate style rules because it provides **the best maintainability**.
  - When you make a change to an external style sheet, all HTML documents that reference that style sheet will automatically use the updated version.

# CSS Syntax

Rules, properties, values, declarations

# How To Specify Styling Rules

- A rule consists of a **selector** that identifies the **HTML element(s)** that will be affected, followed by a series of **property and value pairs** (each pair is also called a **declaration** and terminates with a semicolon).
- A **declaration block** can be together on a single line, or spread across multiple lines and are surrounded by **curly braces**.



```
body {  
    font-family: Arial, Verdana, sans-serif;}  
h1, h2 {  
    color: #ee3e80;}  
p {  
    color: #665544;}
```

# Properties and Values

- Properties indicate **the aspects of the element** we want to change.
- Values specify the settings we want to use for the chosen properties.
- Property names are **predefined by the CSS standard** for different types.

# Common Properties

Property Type	Property
Fonts	font font-family font-size font-style font-weight @font-face
Text	letter-spacing line-height text-align text-decoration text-indent
Color and background	background background-color background-image background-position background-repeat color
Borders	border border-color border-width border-style border-top border-top-color border-top-width, etc

Property Type	Property
Spacing	padding padding-bottom, padding-left, padding-right, padding-top margin margin-bottom, margin-left, margin-right, margin-top
Sizing	height max-height max-width min-height min-width width
Layout	bottom, left, right, top clear display float overflow position visibility z-index
Lists	list-style list-style-image list-style-type

## Add Comments

- We can add comments to CSS between `/*` and `*/`
- They can be used on a single line, or traverse multiple lines.

# Selectors

- Tell the browser which **element(s)** will be affected by this declaration block.
- There are a number of different selector types.
- CSS selectors are **case sensitive**, so they **must** match **element names** and **attribute values** exactly.



# Most Commonly Used CSS Selectors

Selector	Meaning
Universal Selector	Applies to <b>all</b> elements in the document
Element Selector	Matches <b>element names</b>
Class Selector	Matches an element whose <b>class attribute</b> has a value that matches the one specified after the period (or <b>full stop</b> ) symbol
Id Selector	Matches an element whose <b>id attribute</b> has a value that matches the one specified after the pound or <b>hash</b> symbol
Child Selector	Matches an element that is a <b>direct child of another</b>
Descendant Selector	Matches an element that is a <b>descendent of another</b> specified element (not just a direct child of that element)
Attribute Selector	Matches an element that has an <b>attribute</b> or an attribute with a specific value.

# Element Selectors

- The element (or type) selector works on the type of HTML element specified by the selector.
  - For example, the following rule will ensure that all text within <p>...</p> tags is fully justified:

```
p { text-align: justify; }
```

- Group selectors
  - We can select **a group of elements** by separating the different element names with commas.

```
p, div, aside { margin: 0; padding: 0; }
```

# Universal Selector

- You can select all elements by using the **universal element selector**, which is the \* (asterisk) character.

```
* { background-color: yellow; }
```

[https://www.w3schools.com/cssref/tryit.asp?filename=trycss\\_sel\\_all](https://www.w3schools.com/cssref/tryit.asp?filename=trycss_sel_all)

- We seldom use it on its own, but we can use it *as part of a compound rule*.

```
div * {  
    background-color: yellow;  
}
```

[https://www.w3schools.com/cssref/tryit.asp?filename=trycss\\_sel\\_all\\_div](https://www.w3schools.com/cssref/tryit.asp?filename=trycss_sel_all_div)

# Class Selectors and ID Selectors

- There are two attributes supported by **all HTML tags**: class and id.
  - The class attribute is used to **classify** elements.
  - The id attribute is for assigning identifiers to **unique** elements.
- Class selectors
  - Allow you to simultaneously target different HTML elements regardless of their position in the document, so far as they are being labeled with the same class attribute value.
- Id selectors
  - Allow you to target a specific element by its id attribute.

# Class Selectors

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

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
    text-align: center;
    color: red;
}
</style>
</head>
<body>

<h1 class="center">Red and center-aligned heading</h1>
<p class="center">Red and center-aligned paragraph.</p>

</body>
</html>
```

**Red and center-aligned heading**

Red and center-aligned paragraph.

# Id Selectors

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

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>

</body>
</html>
```

Hello World!

This paragraph is not affected by the style.

[https://www.w3schools.com/css/tryit.asp?filename=trycss\\_syntax\\_id](https://www.w3schools.com/css/tryit.asp?filename=trycss_syntax_id)

# Combinators

- A combinator is something that explains the relationship between the selectors.

Selector	Matches	Example
Descendant	A specified element that is contained somewhere within another specified element	<b>div p</b> Selects a <p> element that is <b>contained somewhere within</b> a <div> element. That is, the <p> can be any descendant, <b>not just a child</b> .
Child	A specified element that is a direct child of the specified element	<b>div&gt;h2</b> Selects an <h2> element that is a child of a <div> element.
Adjacent Sibling	A specified element that is the next sibling (i.e., comes directly after) of the specified element.	<b>h3+p</b> Selects the first <p> after any <h3>.
General Sibling	A specified element that follows and shares the same parent as the specified element.	<b>h3~p</b> Selects all the <p> elements that follow and share the same parent as the <h3>.

We can find more information on combinators at [https://www.w3schools.com/css/css\\_combinators.asp](https://www.w3schools.com/css/css_combinators.asp)

# Example: General Sibling Selector

```
<!DOCTYPE html>
<html>
<head>
<style>
div ~ p {
  background-color: yellow;
}
</style>
</head>
<body>

<p>Paragraph 1.</p>

<div>
  <p>Paragraph 2.</p>
</div>

<p>Paragraph 3.</p>
<code>Some code.</code>
<p>Paragraph 4.</p>

</body>
</html>
```

---

Paragraph 1.

Paragraph 2.

Paragraph 3.

Some code.

Paragraph 4.

[https://www.w3schools.com/css/tryit.asp?filename=trycss\\_sel\\_element\\_tilde](https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_tilde)



# Attribute Selectors

- Selecting via presence of element attribute or by the value of an attribute

- Examples:

```
a[target] { ... }
```

- Selects all <a> elements with a *target* attribute.

```
a[target="_blank"] { ... }
```

- Selects all <a> elements with a *target="\_blank"* attribute.

```
[class^="top"] { ... }
```

- Selects all elements with a *class* attribute value that begins with *"top"*.

We can find more information on attribute selectors at [https://www.w3schools.com/css/css\\_attribute\\_selectors.asp](https://www.w3schools.com/css/css_attribute_selectors.asp)

# Pseudo Selectors

- Two types of pseudo selectors:

- Pseudo-element selector

- A pseudo-element acts like an extra but **fictional HTML** element in the code, e.g., a specific part of the element(s)

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

In CSS3, pseudo-elements are indicated by a double colon (::) symbol, but for backward compatibility, systems accept a single colon (:) as they were defined in CSS2.

- Pseudo-class selector

- A pseudo-class is a class (**implicitly added** by the system) which allows us to select the elements which are **at certain states or circumstances**.

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


# Pseudo-Element Selectors

- Commonly used pseudo-elements:
  - `::first-line`
    - Select the first line of the text in the element.
  - `::first-letter`
    - Select the first letter of the text in the element.
  - `::before`
    - Can be used to insert some content before the content of the element.
  - `::after`
    - Can be used to insert some content after the content of the element.
  - `::selection`
    - Select the part of a document that has been highlighted by the user

```
<!DOCTYPE html>
<html>
<head>
<style>
h1::before {
  content: url(smiley.gif);
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>The ::before pseudo-element inserts content
before the content of an element.</p>

</body>
</html>
```

 **This is a heading**

The `::before` pseudo-element inserts content before the content of an element.

[https://www.w3schools.com/css/tryit.asp?filename=trycss\\_before](https://www.w3schools.com/css/tryit.asp?filename=trycss_before)

# Pseudo-Class Selectors

- Commonly used pseudo-classes:

- :link | :visited
  - Select unvisited | visited link(s)
- :hover
  - Select an element when your mouse is over it
- :active
  - Select the active element
- :first-child
  - Select the first child of the target element
- :nth-child(*i*) e.g., p:nth-child(2)
  - Select all <p> elements which are the 2nd child of its parents

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    background-color: green;
    color: white;
    padding: 25px;
    text-align: center;
}
```

```
div:hover {
    background-color: blue;
}
</style>
</head>
<body>
```

<p>Mouse over the div element below to change its background color:</p>

<div>Mouse Over Me</div>

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

Mouse over the div element below to change its background color:



[https://www.w3schools.com/css/tryit.asp?filename=trycss\\_pseudo-class\\_hover\\_div](https://www.w3schools.com/css/tryit.asp?filename=trycss_pseudo-class_hover_div)

# Units of Measurement

- There are multiple ways of specifying a unit of measurement in CSS.
- Some of these are **relative units**, in that they are based on the value of something else, such as the size of a parent element.
- Others are **absolute units**, in that they have a real-world size.
  - Unless you are defining a style sheet for printing, it is recommended to avoid using absolute units.

# Absolute Units

Unit	Description	Type
in	Inches	Absolute
cm	Centimeters	Absolute
mm	Millimeters	Absolute
pt	Points (equal to 1/72 of an inch)	Absolute
pc	Pica (equal to 1/6 of an inch)	Absolute

# Relative Units

Unit	Description	Type
px	Pixel. In CSS2 this is a relative measure, while in CSS3 it is absolute (1/96 of an inch).	Relative (CSS2) <b>Absolute</b> (CSS3)
em	Represents the calculated font-size of the element. When used for font sizes, the em unit is <i>in relation to the font size of the parent</i> .	Relative
%	A measure that is always <i>relative to another value</i> . The precise meaning of % varies depending upon which property it is being used.	Relative
ex	A rarely used relative measure that expresses size <i>in relation to the x-height of an element's font</i> .	Relative
ch	Another rarely used relative measure; this one expresses size <i>in relation to the width of the zero ("o") character of an element's font</i> .	Relative (CSS3 only)
rem	Stands for root em, which is <i>the font size of the root element</i> . Unlike em, which may be different for each element, the rem is constant throughout the document.	Relative (CSS3 only)
vw, vh	Stands for <b>viewport width and viewport height</b> . Both are <b>percentage values</b> (between 0 and 100) of the viewport (browser window). This allows an item to change size when the viewport is resized.	Relative (CSS3 only)

# The concept of Cascade



# Inheritance

- Many (but not all) CSS properties **affect** not only themselves but **their descendants** as well.
- Font, color, list, and **text properties** are inheritable.
- Layout, sizing, border, background and spacing properties are not.
  - The most notable **noninherited** properties are related to **block-style**.
- It is possible to **force an element to inherit value** from its parent element by using “inherit” as the value of the properties.
  - Example: The following rule will make all paragraphs inherit the background properties from their parents:

```
p { background: inherit; }
```

# How CSS Rules Cascade

- If there are two or more rules that apply to the same element, it is important to understand which will take **precedence**.
  - Last Rule
    - If a selector has two rules, **the later** of the two will take precedence.
  - Specificity
    - If one selector is **more specific** than the others, the more specific rule will take precedence.
    - e.g., `p b { }` is more specific than `p { }`. `p#intro { }` is more specific than `p { }`.
  - Important
    - We can add **!important** after any property value to indicate that it should be considered more important than other rules that apply to the same element/selector.

# How CSS Rules Cascade

```
<h1>Potatoes</h1>
<p id="intro">There are <i>dozens</i> of different
  <b>potato</b> varieties.</p>
<p>They are usually described as early, second early
  and maincrop potatoes.</p>
```

## Potatoes

There are *dozens* of different **potato** varieties.

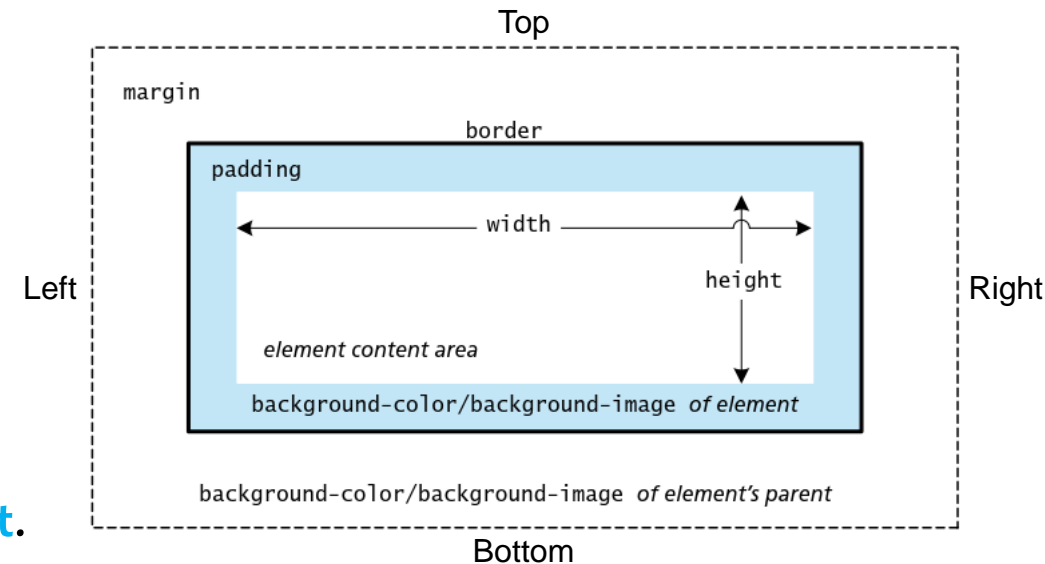
They are usually described as early, second early and maincrop potatoes.

```
* {
  font-family: Arial, Verdana, sans-serif;}
h1 {
  font-family: "Courier New", monospace;}
i {
  color: green;}
i {
  color: red;}
b {
  color: pink;}
p b {
  color: blue !important;}
p b {
  color: violet;}
p#intro {
  font-size: 100%;}
p {
  font-size: 75%;}
```

# The Box Model

# CSS Box Model

- In CSS, all HTML elements exist within an **element box**.
- There are four boxes for each element.
  - Content
    - The innermost box contains the content of the element.
  - Padding
    - A clear area around the content, which is **transparent**.
  - Border
  - Margin
    - Borders provide a way to **visually separate** elements.
  - Margin
    - A clear area outside the border, which helps differentiate one element from another. It is **transparent**.

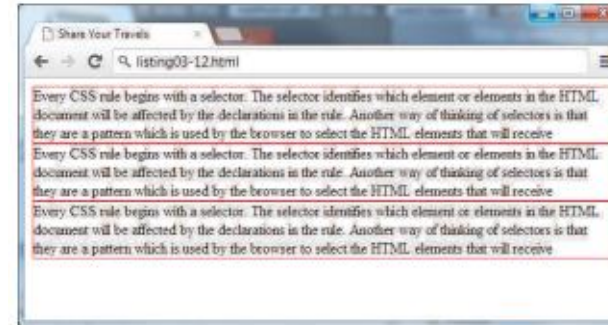


# Margins

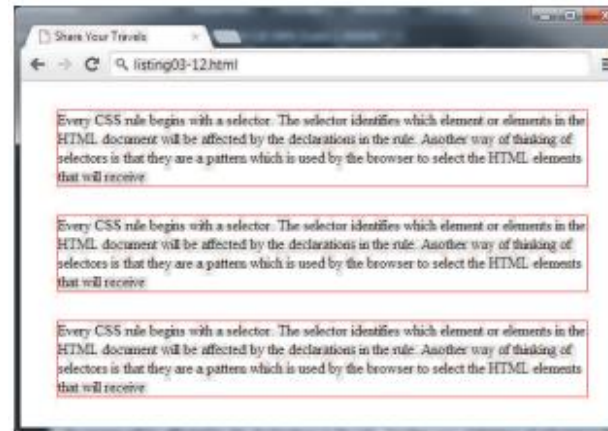
- The margin property controls the gap between boxes of adjacent elements.
- We can set the width of the 4 margins:
  - margin-bottom, margin-left, margin-right, margin-top
- Or setting the four in one declaration
  - margin: top, right, bottom, left
    - E.g., margin: 25px 50px 75px 100px;

# Margins

- Collapsing margins
  - If one box sits **on top of** another, margins are collapsed, which means **the larger of the two margins** will be used and the smaller will be disregarded.
  - However, **horizontal margins are never collapsed**.



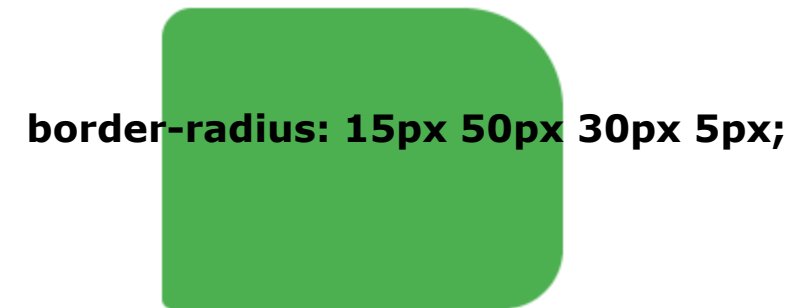
```
p {  
  border: solid 1pt red;  
  margin: 0;  
  padding: 0;  
}
```



```
p {  
  border: solid 1pt red;  
  margin: 30px;  
  padding: 0;  
}
```

# Borders

- You can put borders around all four sides of an element, or just one, two, or three of the sides.
- There are a few features we can specify for the borders
  - border-style
    - solid, dotted, dashed, double, groove, ridge, inset, outset, none, and hidden
  - border-width
    - The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick.
  - border-color
  - border-radius
    - For setting the radius of a rounded corner
  - border-image
    - Give the URL of the image to use as a border



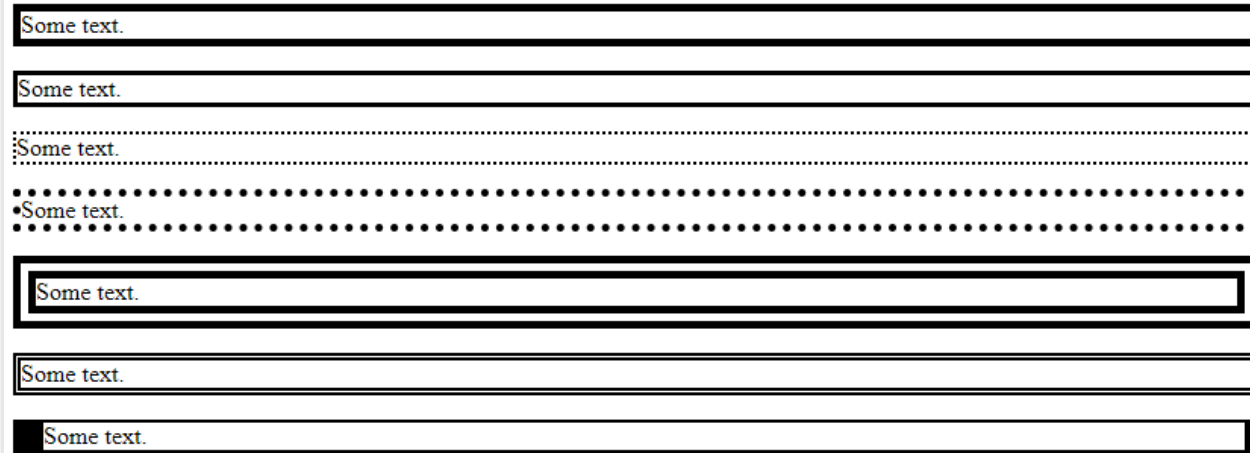
We can find more information on CSS border at [https://www.w3schools.com/css/css\\_border.asp](https://www.w3schools.com/css/css_border.asp)



# Borders

## The borders' style and width Properties

This property specifies the width of the four borders:



**Note:** The "border-width" property does not work if it is used alone. Always specify the "border-style" property to set the borders first.

We also have the long-form for setting each of the borders, e.g.,

```
p {  
  border-top-style: dotted;  
  border-right-style: solid;  
  border-bottom-style: dotted;  
  border-left-style: solid;  
}
```

```
<head>  
<style>  
p.one {  
  border-style: solid;  
  border-width: 5px;  
}  
p.two {  
  border-style: solid;  
  border-width: medium;  
}  
p.three {  
  border-style: dotted;  
  border-width: 2px;  
}  
p.four {  
  border-style: dotted;  
  border-width: thick;  
}  
p.five {  
  border-style: double;  
  border-width: 15px;  
}  
p.six {  
  border-style: double;  
  border-width: thick;  
}  
p.seven {  
  border-style: solid;  
  border-width: 2px 10px 4px 20px;  
}  
</style>  
</head>  
<body>
```

```
<h2>The borders' style and width Properties</h2>  
<p>This property specifies the width of the four borders:</p>
```

```
<p class="one">Some text.</p>  
<p class="two">Some text.</p>  
<p class="three">Some text.</p>  
<p class="four">Some text.</p>  
<p class="five">Some text.</p>  
<p class="six">Some text.</p>  
<p class="seven">Some text.</p>
```

```
<p><b>Note:</b> The "border-width" property does not work if it is used alone.  
Always specify the "border-style" property to set the borders first.</p>
```

# Padding

- The padding property allows you to specify how much space should appear between the content of an element and its border.
- Similar to margins, we can specify the padding of each side,
  - padding-top, padding-right, padding-bottom, padding-left
- Or setting all four in one declaration
  - E.g., padding: 25px 50px 75px 100px;

## Content's Width and Height

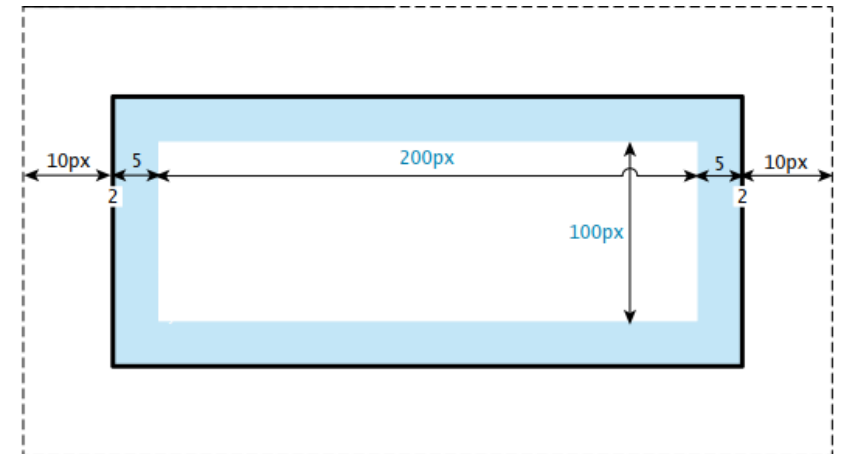
- The width and height properties specify *the size* of the element's **content area**.
- Only block-level elements and non-text inline elements such as images have a **width** and **height** that you can specify.
- By default, inline elements are only as wide as they need to be to display their contents.

# Width and Height – Issue 1

- **By default**, the width and the height refer to *the size of the content area*. So the total size of an element is equal to not only its content area, but also to the sum of its padding, borders, and margins.

- Example:

```
div {  
  width: 200px;  
  height: 100px;  
  padding: 5px;  
  border: 2px solid black;  
  margin: 10px;  
}
```

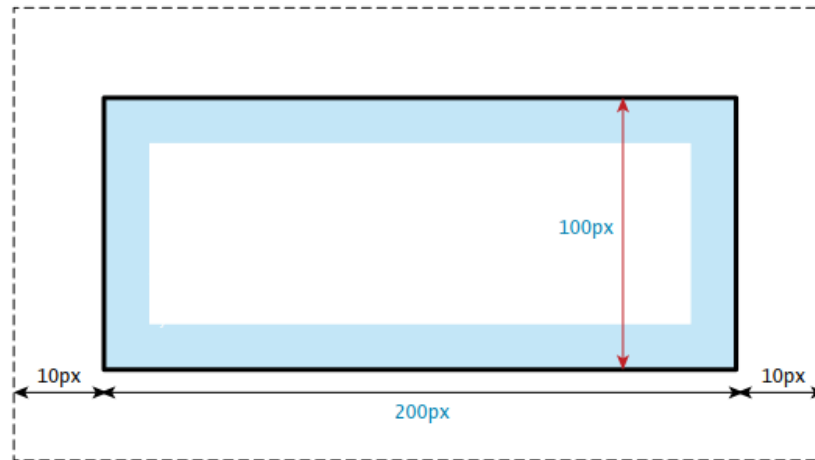


- The total width of the element is:
  - $10\text{px} + 2\text{px} + 5\text{px} + 200\text{px} + 5\text{px} + 2\text{px} + 10\text{px} = 234\text{px}$
- The total height of the element is:
  - $10\text{px} + 2\text{px} + 5\text{px} + 100\text{px} + 5\text{px} + 2\text{px} + 10\text{px} = 134\text{px}$

# Width and Height – Issue 1

- There is another way of representing the size of an element.
- CSS3 introduces the box-sizing property. By setting **box-sizing: border-box**, we can get a different measurement.

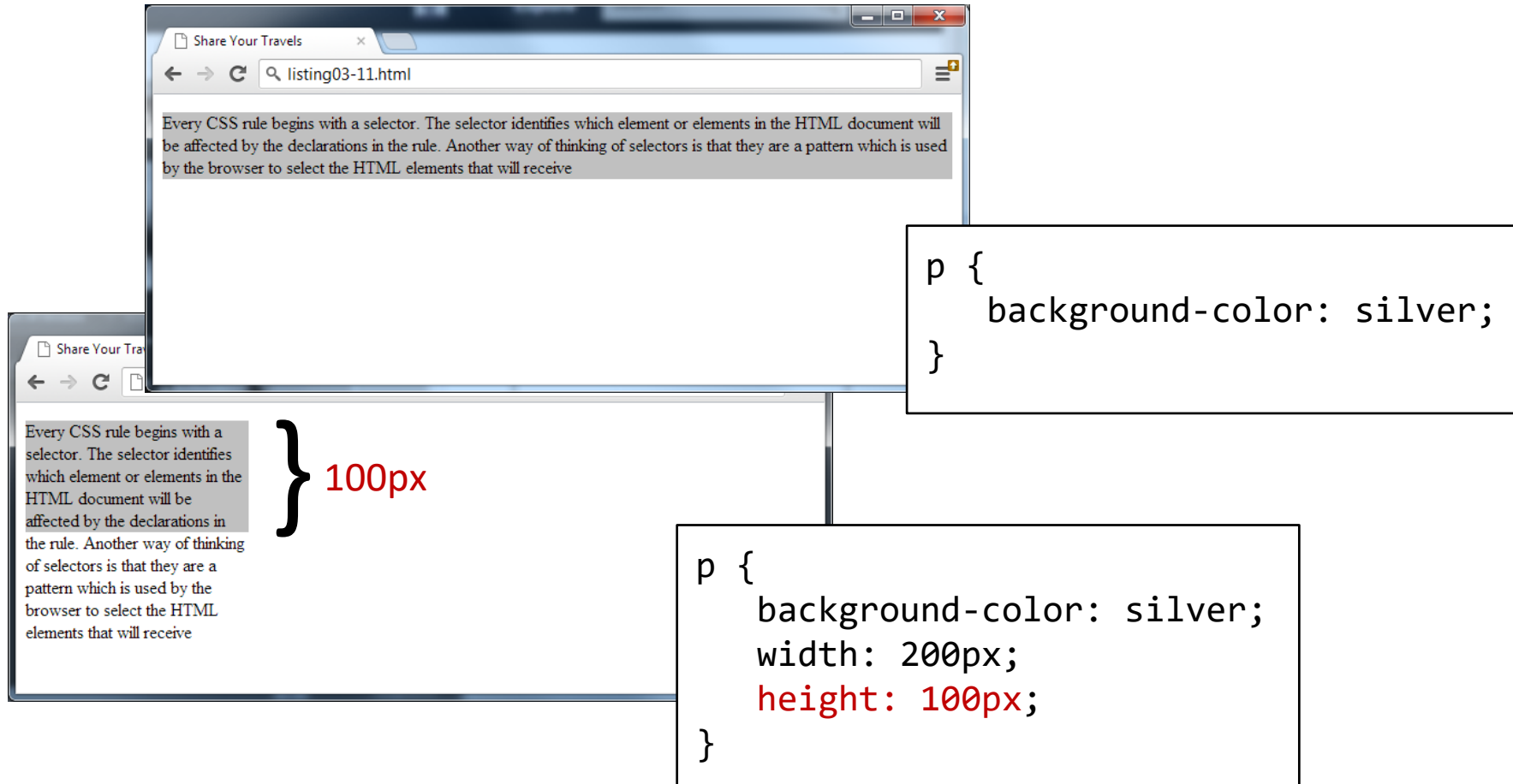
```
div {  
  box-sizing: border-box;  
  width: 200px;  
  height: 100px;  
  padding: 5px;  
  border: 2px solid black;  
  margin: 10px;  
}
```



True element width = 10px + 200px + 10px = 220px

True element height = 10px + 100px + 10px = 120px

# Width and Height – Issue 2



## Width and Height – Issue 2

The **overflow property** specifies whether to clip content or to add scrollbars when an element's content is **too big to fit** in a specified area.

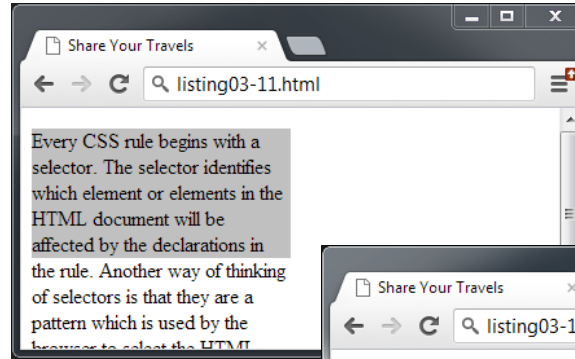
**Visible** - allows the content to be rendered outside the box.

**Hidden** – clips the content and does not appear beyond the box.

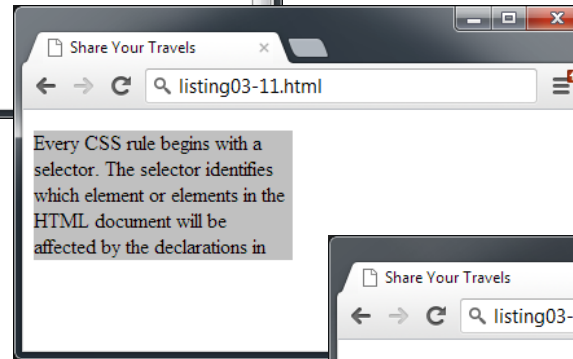
**Scroll** - the scrollbars will always be there, even if the content fits in the box.

**Auto** - The auto value allows the browser to decide how to handle overflow. In most cases, scrollbars are added only when the content doesn't fit into the box.

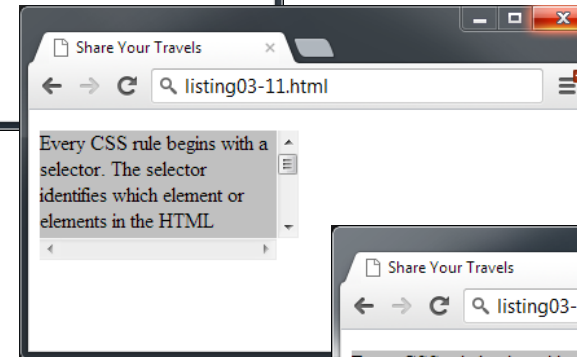
overflow: **visible**; (default)



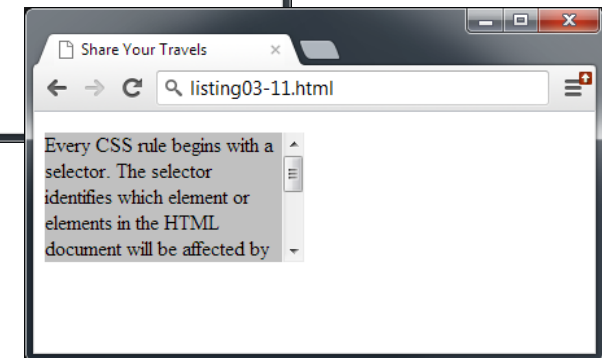
overflow: **hidden**;



overflow: **scroll**;



overflow: **auto**;



# More Controls for the Developers



# CSS At-rules

- At-rules are CSS statements that instructs CSS how to behave.
- They begin with an at sign, '@' character, followed by an identifier
  - Identifiers are the defined keywords by the CSS specification.
- Examples:
  - @charset
  - @import
  - @namespace
  - @media
  - @supports
  - @viewport
  - :
  - :

# CSS At-rules

## @charset

- Example: `@charset "UTF-8";`
- The @charset at-rule specifies the character encoding used in the style sheet.
- It must be the first element in the style sheet; however, a character set that is placed on the HTTP header will override any @charset rule.
- This at-rule is useful when using non-ASCII characters in some CSS properties, like the content property.

# CSS At-rules

## @import

- Examples: `@import 'global.css';`  
`@import url("global.css");`  
`@import url("global.css") screen;`
- This rule instructs the stylesheet to **request and include** an external CSS file as if the contents of that file were right where that line is.
- These rules must precede all other types of rules, except @charset rules.
- Can include list of comma-separated media queries after the URL.
  - If the browser does not support any these queries, it does not load the linked resource.

# CSS At-rules

## @supports

- Example: 

```
@supports (display: grid)
{
    div {
        display: grid;
    }
}
```
- A **conditional group rule** that will apply its content if the browser meets the criteria of the given condition.
  - This is called a feature query.
- Can use the 'not' operator, 'and' operator, and 'or' operator to form the condition.

```
@supports (display: table-cell) and (display: list-item) {...}
```

# CSS Functions

- CSS functions are used as a value for various CSS properties.
  - e.g., `background-color: rgb(255,0,0);`
- Here are some example functions:

Function	Description
<a href="#">attr()</a>	Returns the value of an attribute of the selected element
<a href="#">calc()</a>	Allows you to perform calculations to determine CSS property values
<a href="#">hsl()</a>	Defines colors using the Hue-Saturation-Lightness model (HSL)
<a href="#">hsla()</a>	Defines colors using the Hue-Saturation-Lightness-Alpha model (HSLA)
<a href="#">linear-gradient()</a>	Sets a linear gradient as the background image. Define at least two colors (top to bottom)
<a href="#">radial-gradient()</a>	Sets a radial gradient as the background image. Define at least two colors (center to edges)
<a href="#">rgb()</a>	Defines colors using the Red-Green-Blue model (RGB)
<a href="#">var()</a>	Inserts the value of a custom property

# CSS Variables

- Custom properties (sometimes referred to as CSS variables or cascading variables) are entities defined by developers that contain specific values **to be reused** throughout a document.
- We can declare a custom property using this notation
  - `--fancy-color: pink;`
  - The variable name must begin with two dashes (--) and is case sensitive
- To use the custom property, we use the var() CSS function
  - `color: var(--fancy-color);`

# CSS Variables

- Variables in CSS should be declared within a CSS selector that defines its scope.
  - For a global scope you can use either the `:root` or the `body` selector.

```
:root {  
  --main-bg-color: coral;  
}  
  
#div1 {  
  background-color: var(--main-bg-color);  
}  
  
#div2 {  
  background-color: var(--main-bg-color);  
}
```

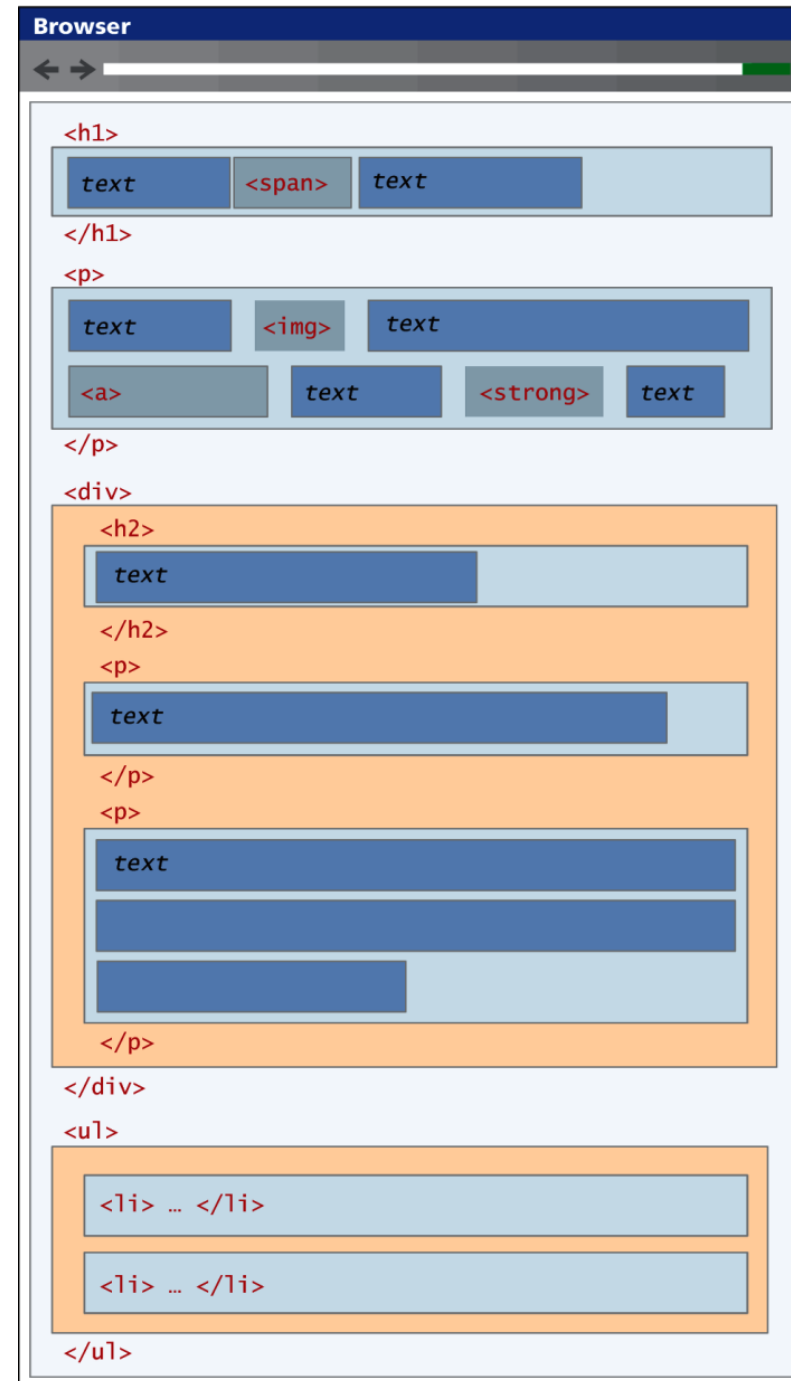
# CSS Layout

Display, Visibility, Floats, Positioning



# Normal Flow

- Normal flow refers to how the browser will **normally display** **block-level** elements and **inline** elements *from left to right and from top to bottom*.
- The HTML is displayed in the exact order in which it appears in the source code.



A document consists of block-level elements stacked from top to bottom.

Within a block, inline content is horizontally placed left to right.

Some block-level elements can contain other block-level elements (in this example, a <div> can contain other blocks).

In such a case, the block-level content inside the parent is stacked from top to bottom within the container (<div>).

# Display Property

- The main methods of achieving page layout in CSS are all values of the **display property**.
- **Everything** in normal flow has a value of display, which has a default setting.
  - E.g., for <p>, its display property is set to block; for <a>, is set to inline.
- We can change this default display behavior by changing the display property of the element.

```
<head>
<style>
  li {
    display: inline;
    margin-right: 10px;
  }
  li.coming-soon {
    display: none;
  }
</style>
</head>
<body>
  <ul>
    <li>Home</li>
    <li>Products</li>
    <li class="coming-soon">Services</li>
    <li>About</li>
    <li>Contact</li>
  </ul>
</body>
```

Home Products About Contact

# Display Property

- The values this property can take are:
  - Inline - This causes a block-level element to act like an inline element.
  - Block - This causes an inline element to act like a block-level element.
  - Inline-block
    - This causes a **block-level element** to flow **like an inline** element, but it can be sized using width and height and maintains its block integrity like a block box. Therefore, it won't be broken across paragraph lines like an inline box.
  - None - This **hides** an element from the page. No space is allocated to this element and its descendants.
  - Flex - This makes an element to use the **Flexbox model** and becomes a flex container.
  - Grid - This makes an element to use the **Grid model** and becomes a grid container.
  - . . . .

# Visibility Property

- The visibility property specifies whether or not an element is visible.
- This property can take two values:
  - **hidden**
    - This hides the element.
  - **visible**
    - This shows the element.
- The hidden element *still takes up space* on the page.

```
<head>
<style>
  li {
    display: inline;
    margin-right: 10px;
  }
  li.coming-soon {
    visibility: hidden;
  }
</style>
</head>
<body>
  <ul>
    <li>Home</li>
    <li>Products</li>
    <li class="coming-soon">Services</li>
    <li>About</li>
    <li>Contact</li>
  </ul>
</body>
```

Home Products

About Contact

# Float Property

- The float property allows you to **place the element** as far to the left or right of **the containing element** and change it from normal flow.
- The **surrounding** content floats around the floated element.
- The float property has four possible values:
  - left — Floats the element to the left.
  - right — Floats the element to the right.
  - none — Specifies no floating at all. [default]
  - inherit — Inherited from the element's parent.

# Float Property

```
<head>
<style>
img {
  float: right;
}
</style>
</head>
<body>
<h1>The float Property - Right</h1>
```

```
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus
imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae
scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices
nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue
ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in
odio. Praesent convallis urna a lacus interdum ut hendrerit risus
congue.</p>|
</body>
```

## The float Property - Right

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue.



## The float Property - None



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue.

## The float Property - Left



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue.

# Placing Elements Side-by-side

- A lot of layouts place boxes next to each other. The float property is commonly used to achieve this.
- When elements are floated, the height of the boxes can affect where the following elements sit.

```
<style>
  body {
    width: 750px;
    font-family: Arial, Verdana, sans-serif;
    color: #665544;
  }
  p {
    width: 230px;
    float: left;
    margin: 5px;
    padding: 5px;
    background-color: #efefef;
  }
</style>
```

## The float Property

Paragraph 1 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 2 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio.

Paragraph 3 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 4 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 5 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 6 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

# Placing Elements Side-by-side

- Setting the height of the paragraphs to be the same height as the tallest paragraph would solve this issue.
  - E.g., height: 250px;

## The float Property

Paragraph 1 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 2 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio.

Paragraph 3 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 4 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 5 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 6 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

- We can apply the **clear property** to specify on which sides of an element other floating elements are **not allowed to touch**.
  - Possible values of clear property:
    - none, left, right, both
  - E.g., set `<p> paragraph 4 ... </p>` with clear: left;

## The float Property

Paragraph 1 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 2 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio.

Paragraph 3 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 4 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 5 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Paragraph 6 - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.



# Position Property

- CSS has the following positioning schemes that allow you to control the layout of a page:
  - Normal flow (**static positioning**)
  - Relative positioning
    - This moves an element **relative to** the location it would occupy in the **normal flow**. This does not affect the position of **surrounding elements**; they **stay in the position** they would be in in normal flow.
  - Absolute positioning
    - This positions the element in relation to its **containing element**. **The space it would have occupied is released**. Other elements will flow into the released space.
  - Fixed positioning
    - This is a form of absolute positioning that positions the element in relation to the **browser window**, as opposed to the containing element.
  - Sticky positioning
    - This makes an element act **like static positioning until** it hits a **defined offset** from the viewport, at which point it acts like fixed positioning.
- You specify the positioning scheme using the **position property** in CSS.

# Relative positioning

```
<!DOCTYPE html>
<html>
<head>
<style>
div.relative {
  position: relative;
  top: 25px;
  left: 30px;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>
```

```
<h2>position: relative;</h2>
```

```
<p>An element with position: relative; is positioned relative to its
normal position:</p>
```

```
<div class="relative">
This div element has position: relative;
</div>
```

```
<p>The element is positioned according to the normal flow of the
document, and then offset relative to itself based on the values of top,
right, bottom, and left. The offset does not affect the position of any
other elements; thus, the space given for the element in the page layout
is the same as if position were static.</p>
```

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

## position: relative;

An element with position: relative; is positioned relative to its normal position:

This div element has position: relative;  
The element is positioned according to the normal flow of the document, and then offset relative to itself based on the values of top, right, bottom, and left. The offset does not affect the position of any other elements; thus, the space given for the element in the page layout is the same as if position were static.

# Absolute positioning

```
<!DOCTYPE html>
<html>
<head>
<style>
div.relative {
  position: relative;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}
```

```
div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #AAAEDE;
}
```

```
</style>
</head>
<body>
```

```
<h2>position: absolute;</h2>
```

<p>An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):</p>

```
<div class="relative">This div element has position: relative;
  <div class="absolute">This div element has position: absolute;</div>
  <p>This element follows the previous absolutely positioned element.</p>
</div>
```

## position: absolute;

An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):

This div element has position: relative;

This element follows the previous absolutely positioned element.

This div element has position: absolute;

# Fixed Positioning

```
<!DOCTYPE html>

<html>
<head>
<style>
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>

<h2>position: fixed;</h2>

<p>An element with position: fixed; is positioned relative to the
viewport, which means it always stays in the same place even if the page
is scrolled:</p>

<div class="fixed">
This div element has position: fixed;
</div>

</body>
</html>
```

## position: fixed;

An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled:

This div element has position: fixed;

# Sticky Positioning

```
<!DOCTYPE html>
<html>
<head>
<style>
div.sticky {
  position: sticky;
  top: 20px;
  width: 100px;
  padding: 5px;
  background-color: #cae8ca;
  border: 2px solid #4CAF50;
}
</style>
</head>
<body>
```

<p>Try to <b>scroll</b> inside this frame to understand how sticky positioning works.</p>

<p>Note: IE/Edge 15 and earlier versions do not support sticky position.</p>

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

```
<div style="padding-bottom:2000px">
```

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

<p>Scroll back up to remove the stickyness.</p>

<p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p>

<p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus

Try to **scroll** inside this frame to understand how sticky positioning works.

Note: IE/Edge 15 and earlier versions do not support sticky position.

I am sticky!

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

Scroll back up to remove the stickyness.

Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.

Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.

I am sticky!

Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.

Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.

# Flexbox

- Provide a more efficient way to **lay out, align and distribute space** among items in a container, even when their size is **unknown** and/or **dynamic**.
- Give the **container** (parent element) the ability to alter its **items'** (children) width/height (and order) to best fill the available space

Container



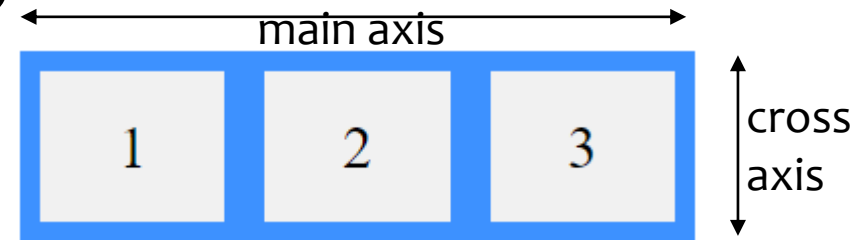
# Flex Container

- A Flexible Layout must have a **parent element** with the display property set to flex.
- Direct child elements(s) of the flexible container automatically becomes flexible items.

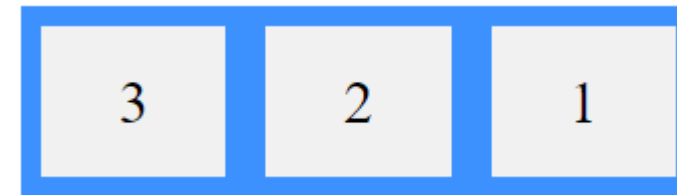
```
section {  
    display: flex;  
}
```

# Flex-Direction Property

- Flexbox provides a property called flex-direction that specifies what **direction the flexbox children** (items) are laid out in.



- `flex-direction: row`
  - [default] Stacks the flex items horizontally (from left to right).

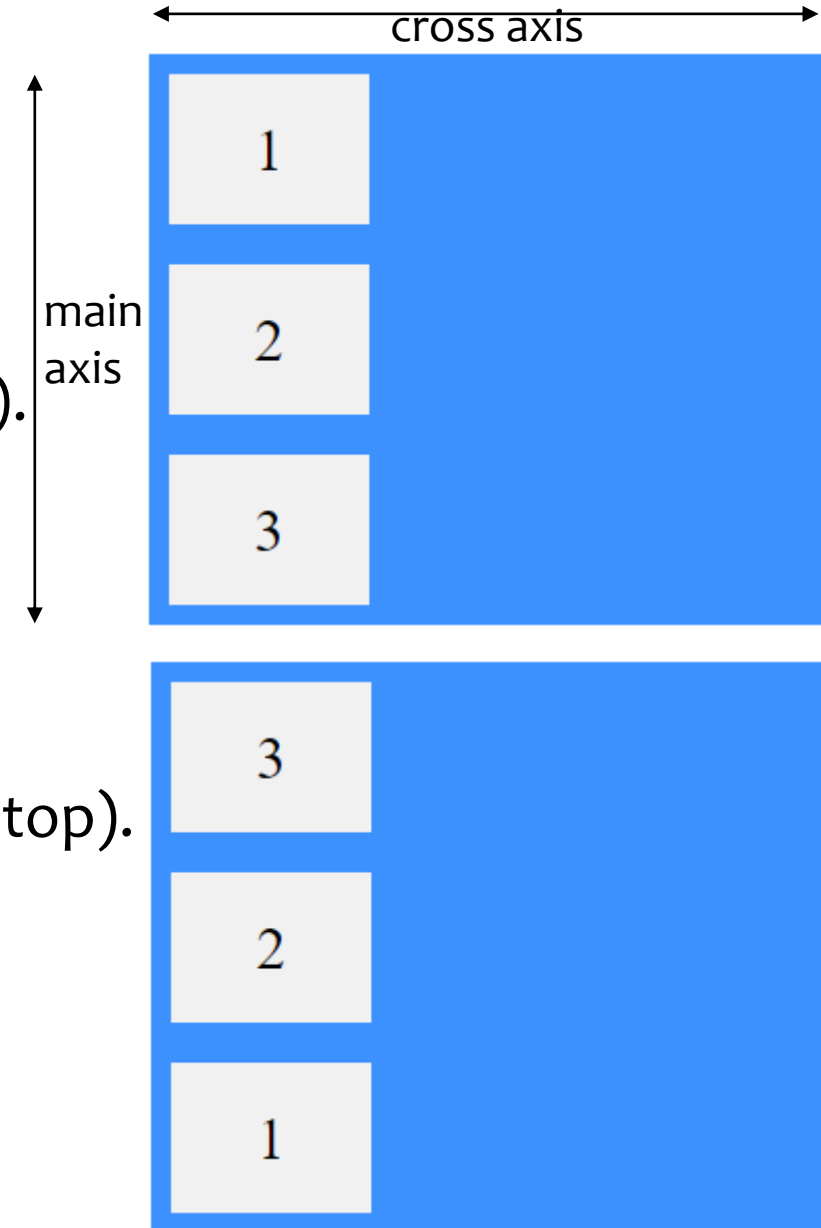


- `flex-direction: row-reverse`
  - Stacks the flex items horizontally (but from right to left).



# Flex-Direction Property

- flex-direction: column
  - Stacks the flex items vertically (from top to bottom).
- flex-direction: column-reverse
  - Stacks the flex items vertically (but from bottom to top).

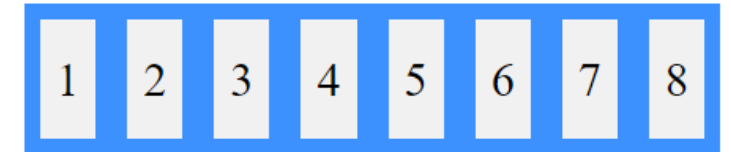


## Flexbox Initial Setting

- Items display in a row (the flex-direction property's default is row).
- The items start from the left (start) edge of the main axis.
- The items do not stretch on the main dimension, but can shrink.
- The items will stretch to fill the size of the cross axis.
- The flex-basis property is set to auto.
- The flex-wrap property is set to nowrap.

## Flex-Wrap Property

- By default, flex items will all try to fit onto one line. You can change that and **allow the items to wrap** as needed.

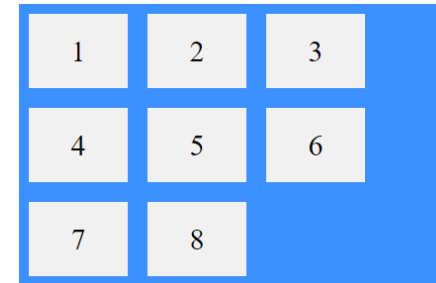


- flex-wrap: nowrap

- [default] Specifies that the flex items will not wrap.

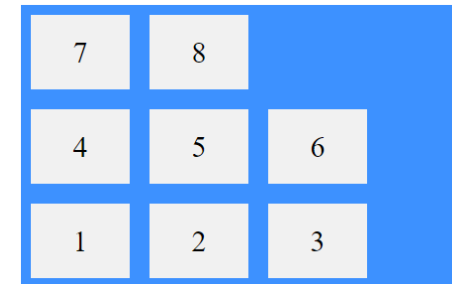
- flex-wrap: wrap

- Specifies that the flex items will wrap.



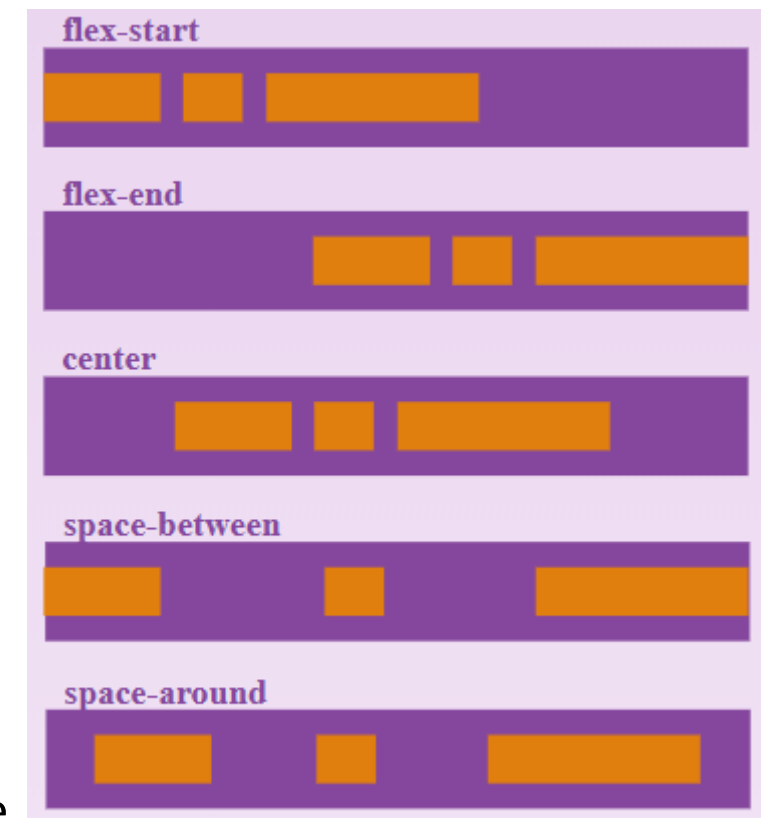
- flex-wrap: wrap-reverse

- Specifies that the flex items will wrap if necessary, in reverse order.



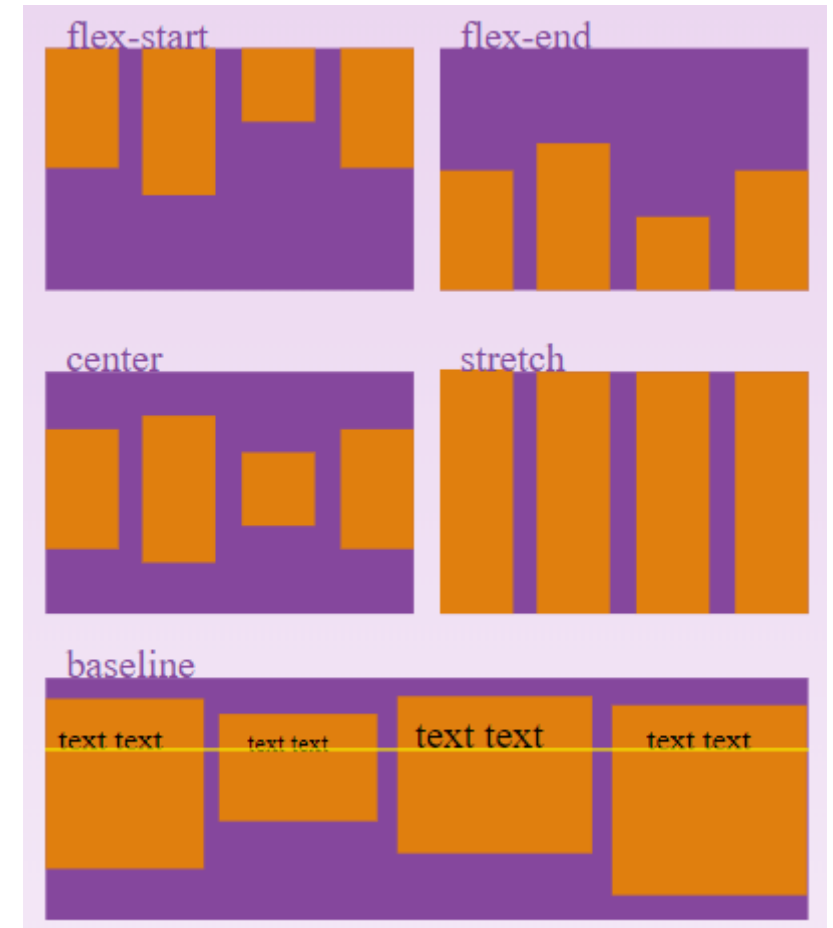
# Justify-Content Property

- This defines the **alignment along the main** axis.
- `justify-content: flex-start`
  - [**default**] Aligns the flex items at the beginning of the container.
- `justify-content: flex-end`
  - Aligns the flex items at the end of the container.
- `justify-content: center`
  - Aligns the flex items at the center of the container.
- `justify-content: space-between`
  - Displays the flex items with space between the lines.
- `justify-content: space-around`
  - Displays the flex items with space before, between, and after the lines.



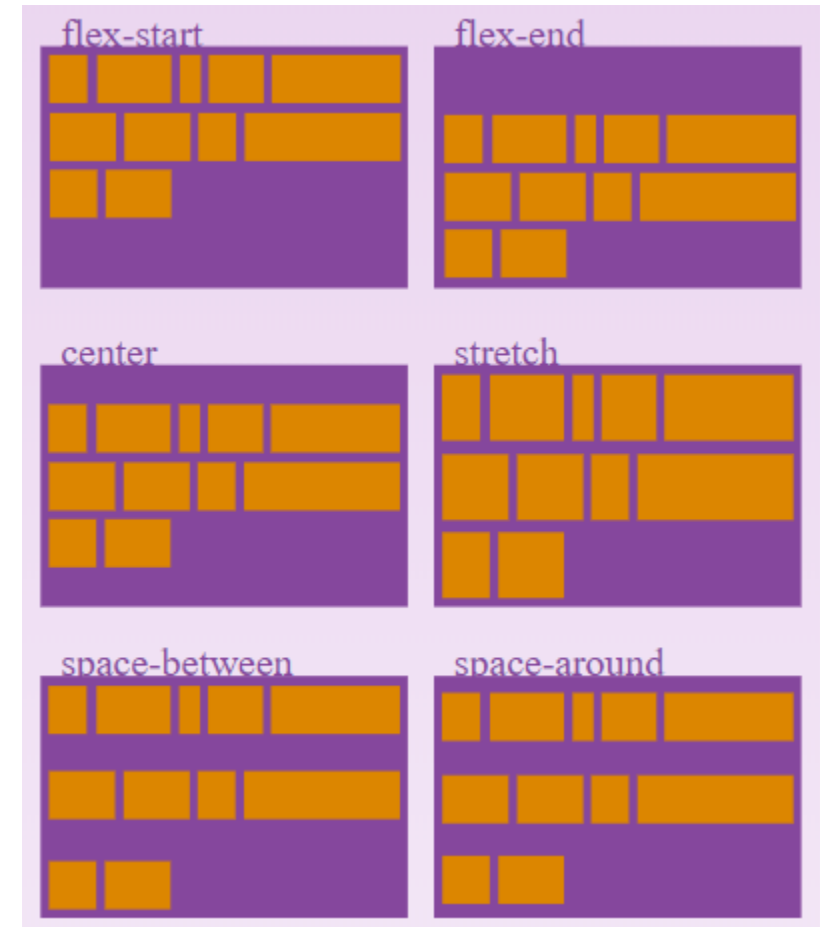
# Align-Items Property

- This defines how flex items are laid out **along the cross axis** on the current line.
- `align-items: stretch`
  - `[default]` Stretches the flex items to fill the container.
- `align-items: center`
  - Aligns the flex items in the middle of the container.
- `align-items: flex-start`
  - Aligns the flex items at the top of the container.
- `align-items: flex-end`
  - Aligns the flex items at the bottom of the container.
- `align-items: baseline`
  - Aligns the flex items such as their baselines aligns.



# Align-Content Property

- This aligns a flex container's lines within when there is extra space in the **cross axis**.
- `align-content: stretch`
  - [**default**] Stretches the flex lines to take up the remaining space.
- `align-content: space-between`
  - Displays the flex lines with equal space between them.
- `align-content: space-around`
  - Displays the flex lines with space before, between, and after them
- `align-content: center`
  - Displays the flex lines in the middle of the container.
- `align-content: flex-start`
  - Displays the flex lines at the start of the container.
- `align-content: flex-end`
  - Displays the flex lines at the end of the container.



# Control on Flex Items

- The flex item properties are:

- Order

- By default, flex items are laid out in the source order. However, the order property controls the order in which they appear in the flex container.

<https://i.cs.hku.hk/~atctam/c3322/CSS/flex-order.html>

- align-self

- Specifies the alignment for the selected item inside the flexible container.
    - Overrides the default alignment set by the container's align-items property.

<https://i.cs.hku.hk/~atctam/c3322/CSS/flex-align-self.html>

<https://i.cs.hku.hk/~atctam/c3322/CSS/flex-flex.html>

- flex-grow

- Specifies how much a flex item will grow to occupy the remaining space relative to the rest of the flex items
    - e.g., flex-grow: 1;

- flex-shrink

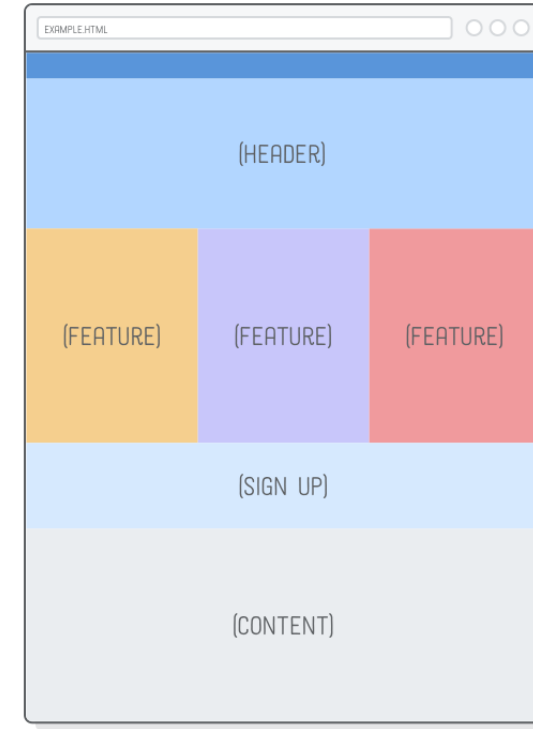
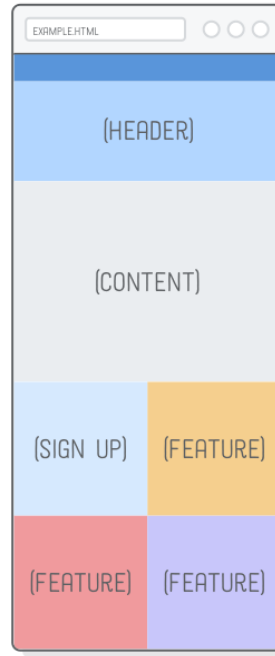
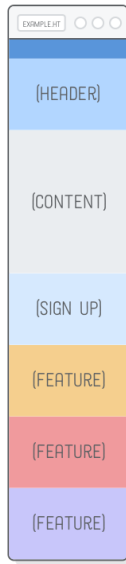
- Specifies how much a flex item will shrink relatively

- flex-basis

- Specifies the initial length of a flex item

- flex

- Shorthand property for flex-grow, flex-shrink, and flex-basis



# Responsive Web Design



# Responsive Web Design

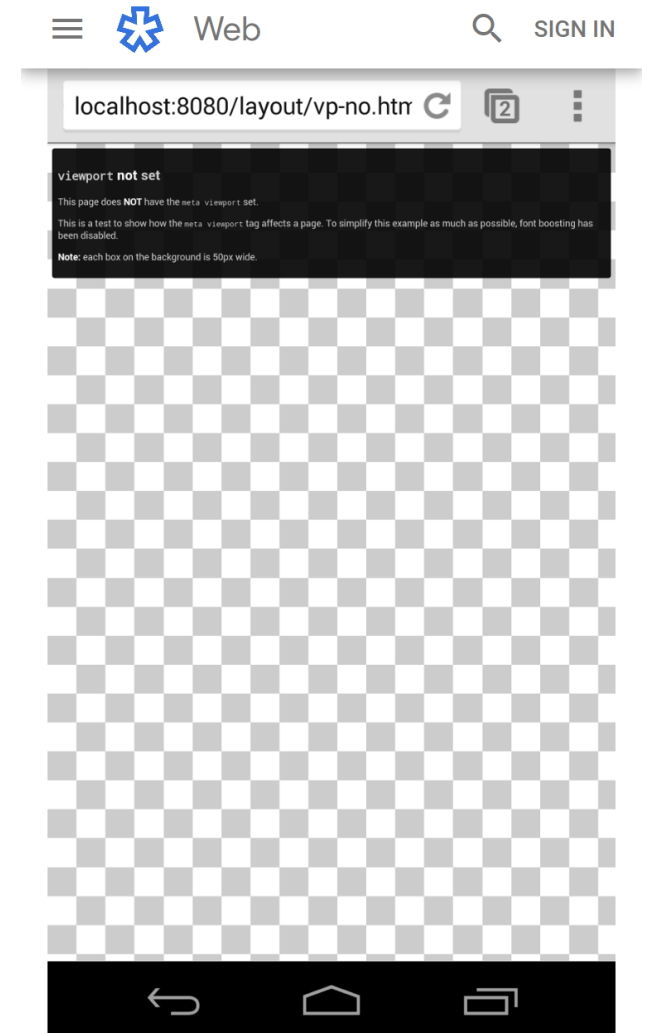
- “Responsive web design” refers to the idea that your website **should display equally well** in everything from widescreen monitors to mobile phones.
- The ultimate goal of a responsive site is that it has **one codebase** of HTML, CSS, and JavaScript that can then be displayed on any device that wants to view it.

# Responsive Design

- There are four key components that make responsive design work.
  - Setting viewports via the <meta> tag
  - Customizing the CSS for **different viewports** using media queries
  - Scaling images to the viewport size
  - Flexible layouts

# Set the Viewport

- To fit standard websites onto small screens, mobile browsers **render the page on a** canvas called the **viewport** and then **shrink that** viewport down **to fit the width of the screen** (device width).
  - For example, on iPhones, Mobile Safari sets the viewport width to 980 pixels, so a web page is rendered as though it were on a desktop browser window set to 980 pixels wide. But that rendering gets shrunk down to 320 pixels wide when the iPhone is in portrait orientation, cramming a lot of information into a tiny space.

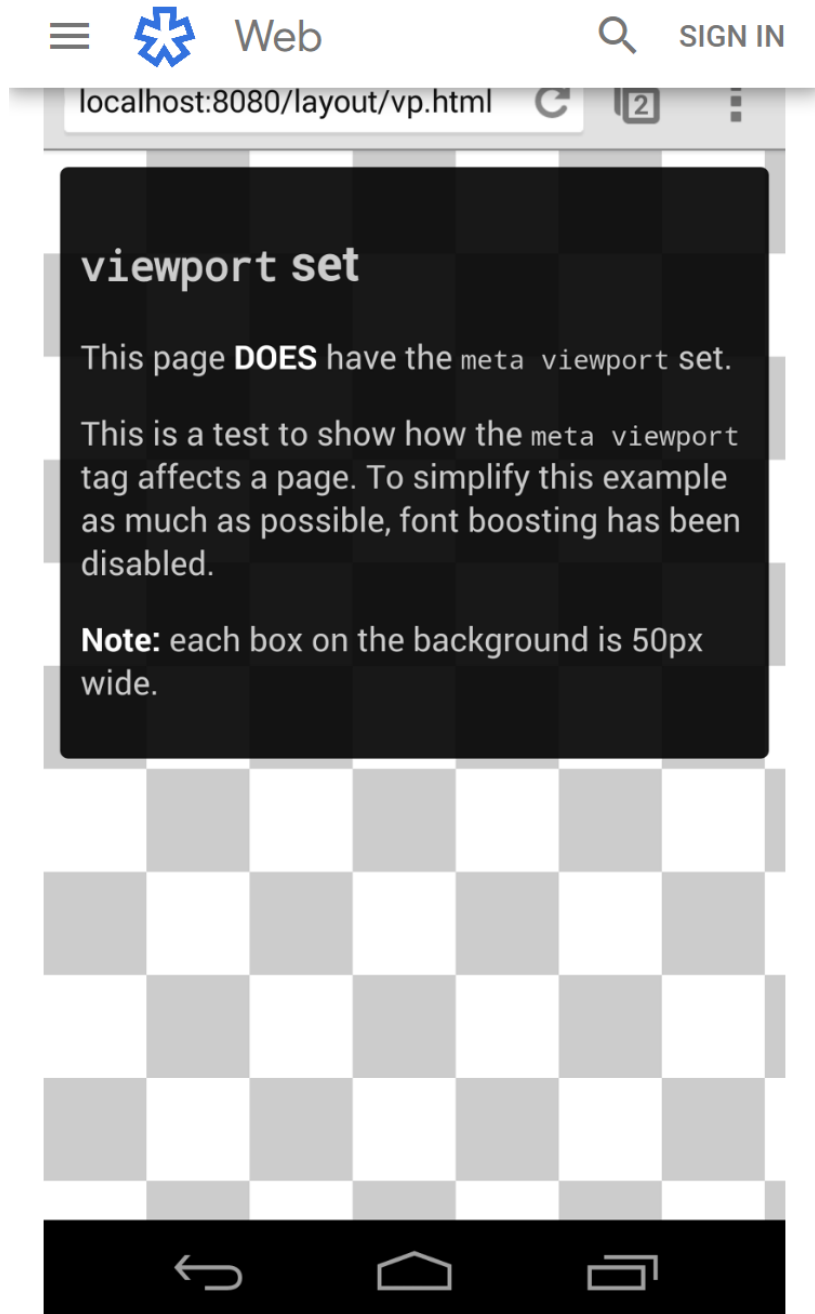


# Set the Viewport

- The web page can **tell the mobile browser the viewport size** to use via the viewport <meta> element.
- Add the following meta element to the head of the HTML document:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

- This line tells the browser to set
  - The width of the viewport equal to the width of the device screen (width=device-width).
  - The initial-scale to the zoom level to 1 (100%), means that the page will be rendered at the size determined by the width attribute, and will not be zoomed in or out .

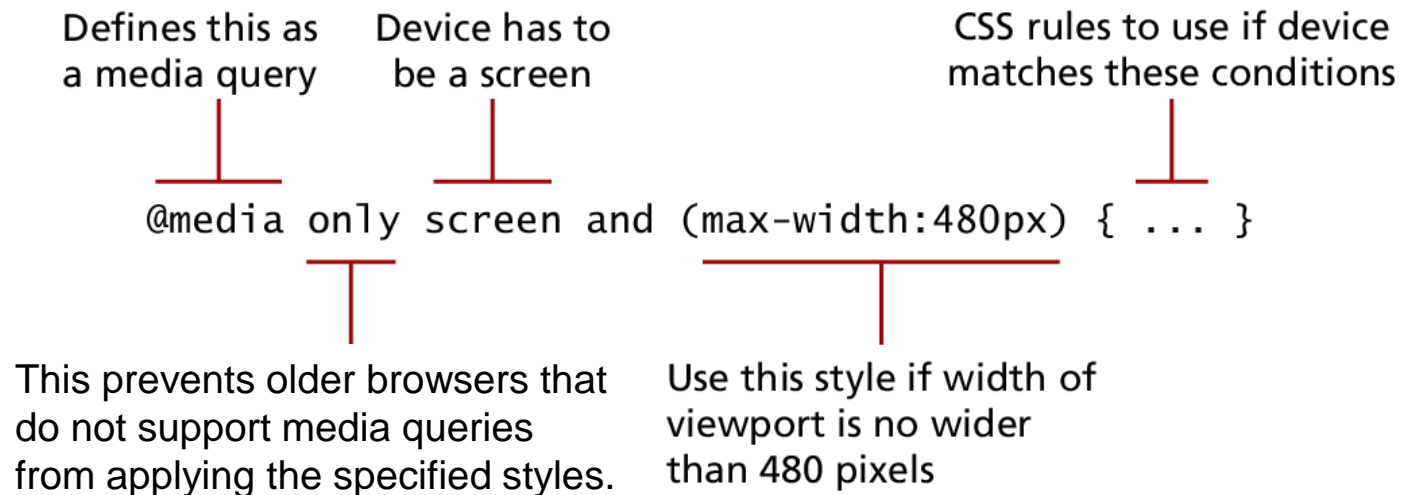


## Set the Viewport

Page with the viewport set

# Media Queries

- The other key component of RWD is **CSS media queries**.
- A media query is a way to apply style rules based on the medium that is displaying the file.
  - You can use these queries **to look at the capabilities of the device**, and **then define CSS rules** to target that device.

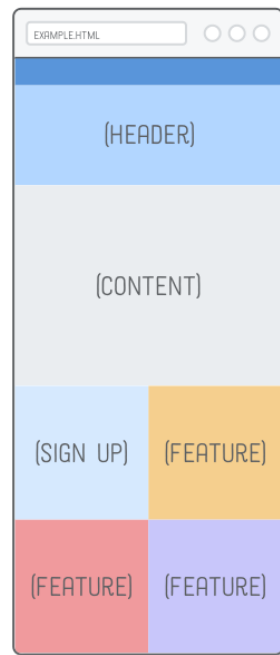


# Media Queries

```
/* Mobile Styles */
@media only screen and (max-width: 400px)
{ . . . }

/* Tablet Styles */
@media only screen and (min-width: 401px) and (max-width: 960px)
{ . . . }

/* Desktop Styles */
@media only screen and (min-width: 961px)
{ . . . }
```

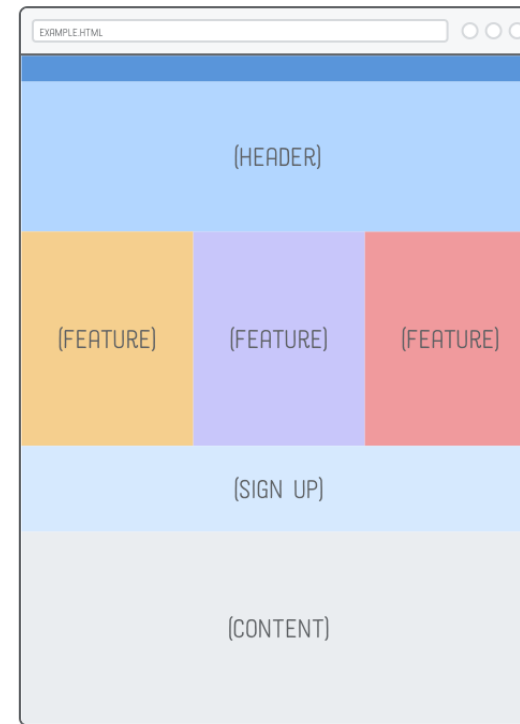


You can also have different stylesheets for different media, like this:

```
<link rel="stylesheet" media="screen and (max-width: 400px)" href="mobile.css">
```

```
<link rel="stylesheet" media="screen and (min-width: 401px) and (max-width: 960px)" href="tablet.css">
```

```
<link rel="stylesheet" media="screen and (min-width: 961px)" href="desktop.css">
```



# Media Queries

- Browser features you can examine with Media Queries

Feature	Description
width	Width of the viewport
height	Height of the viewport
aspect-ratio	The ratio between the width and the height of the viewport
max-width	The maximum width of the display area, such as a browser window
min-width	The minimum width of the display area, such as a browser window
orientation	Whether the device is portrait or landscape
color	The number of bits per color









We can find more information on Media Queries at [https://www.w3schools.com/cssref/css3\\_pr\\_mediaquery.asp](https://www.w3schools.com/cssref/css3_pr_mediaquery.asp)



## RWD – How to choose breakpoints

- Contemporary responsive sites will typically provide CSS rules for phone displays first, then tablets, then desktop monitors, an approach called **progressive enhancement**, in which a design is adapted to progressively more advanced devices.
- Create breakpoints **based on content**, never on specific devices, products, or brands.
  - Design the content to fit on a small screen size first, then expand the screen until a breakpoint becomes necessary.

# RWD – How to choose breakpoints

New York, NY			
Tuesday, April 15th Overcast			
 58°F Precipitation: 100% Humidity: 97% Wind: 4 mph SW Pollen Count: 36			
Today		68° 36°	Pollen 36
Wednesday		50° 39°	Pollen 36
Thursday		55° 39°	Pollen 36
Friday		54° 43°	Pollen 36
Saturday		64° 46°	Pollen 36
Sunday		64° 50°	Pollen 36
Monday		61° 50°	Pollen 36

New York, NY			
Tuesday, April 15th Overcast			
 58°F Precipitation: 100% Humidity: 97% Wind: 4 mph SW Pollen Count: 36			
Today		68° 36°	Pollen 36
Wednesday		50° 39°	Pollen 36
Thursday		55° 39°	Pollen 36
Friday		54° 43°	Pollen 36
Saturday		64° 46°	Pollen 36
Sunday		64° 50°	Pollen 36
Monday		61° 50°	Pollen 36

# Making Images Responsive

- Images can be difficult to make responsive.
  - How do we provide images with appropriate dimensions to large screens, taking advantage of the screen space, but not waste bandwidth by sending those huge images to devices with small screens?
- Two common ways to deal with:
  - Set the image width to something flexible, and save the image in a large file size so that it can flex to fit large and small screens.
  - **Change the images** that display depending upon the device that is viewing it.

# Making Images Responsive

- Set the image width to something flexible.
  - Set the max-width property of an image to 100%, the image will be responsive and scale up and down when resizing the screen, but never scale up to be larger than its original size.
- Different Images for Different Devices
  - Use media queries to determine which version of the images to load.

```
img {  
  max-width: 100%;  
  height: auto;  
}
```

```
/* For devices smaller than 400px: */  
body {  
  background-image: url('img_smallflower.jpg');  
}  
  
/* For devices 400px and larger: */  
@media only screen and (min-width: 400px) {  
  body {  
    background-image: url('img_flowers.jpg');  
  }  
}
```

# Flexible Layouts

- One of the main problems faced by web designers is that the size of the screen used to view the page can vary quite a bit. Satisfying different users of different devices can be difficult.
- Most designers use the following approaches to dealing with the problems of screen size.
  - Liquid (fluid) layout
    - A liquid layout **uses relative units** such as percentages and ems, rather than using fixed units such as pixels.
  - Adaptive layout
    - An adaptive layout uses fixed units, but **using media queries to adaptively change the sizes** when the viewport is at certain widths.
  - Responsive layout
    - A responsive layout uses **using both relative units and media queries**. It takes the best from both liquid and adaptive layouts.

# Reading

- MDN Web Docs
  - Introduction to CSS
    - [https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction\\_to\\_CSS](https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS)
  - Introduction to CSS Layout
    - [https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\\_layout/Introduction](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Introduction)

# References

- Some slides are borrowed from the book:
  - Fundamentals of Web Development by Randy Connolly and Ricardo Hoar, published by Pearson.
- A Complete Guide to Flexbox
  - <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
- Responsive Design, No 10. of HTML & CSS Is Hard
  - <https://internetingishard.com/html-and-css/responsive-design/>