# Lecture 1:- Introduction to CSS – Notes

**🔹 What is CSS?**

* **CSS** stands for **Cascading Style Sheets**.
* It is used to **style** HTML content and make websites **visually appealing**.
* The term "**Cascading**" refers to the **hierarchical order** of style rules, from general to specific — similar to how water flows down in a **waterfall**. inline to external

The **Cascade** in **Cascading Style Sheets (CSS)** is the set of rules that the browser follows to decide **which style to apply when there is a conflict**. For example, if an element is styled in both an external stylesheet and with inline CSS, which one should take precedence?

### Cascading CSS applies styles based on:

* **Importance**: Styles marked with !important override others.
* **Specificity**: Inline styles > ID selectors > class selectors > element selectors.
* **Source order**: Later styles override earlier ones if specificity is the same

**🔹 Why CSS is Important**

* HTML was initially only for **content structure**, with **no easy styling options**.
* Developers struggled to control how web pages looked.
* Early attempts to style HTML were **inefficient and messy**.

**🔹 The Evolution of Web Styling**

**📜 The Old HTML Days:**

* Initially, websites were styled using HTML tags like:
  + <font> — controlled **size**, **color**, **face** of text.
  + <center> — controlled **layout/positioning**.
* These methods led to:
  + **Cluttered HTML** code.
  + **Mixing of content and design**.
  + Difficult maintenance as websites grew.

**⚠️ Deprecated Elements:**

* <font>, <center>, and their attributes (size, color, face) are now **deprecated**.
* Deprecated means: **no longer recommended** for use.
* Reason: HTML should focus on **structure**, not **style**.

**🔹 Birth of CSS**

* In **1996**, **W3C** introduced CSS as a **separate styling language**.
* Led by **Håkon Wium Lie** (Father of CSS).
* This created a **clean separation**:
  + HTML → for **content**
  + CSS → for **styling/design**

**🔹 Benefits of CSS**

* Clean and **modular code**.
* Easier **maintenance** and **scalability**.
* Can transform websites **without altering HTML**.

**🔹 Real Example (CSS in Action)**

* Website demo: <https://appbrewery.github.io/just-add-css>
* Two versions shown:
  1. **HTML only** – Basic, bland.
  2. **HTML + CSS** – Stylish, colorful, structured.
* **Key observation**:
  1. The **HTML remains the same**.
  2. Only the **CSS file** changes the appearance.

**🔹 Other Stylesheet Languages**

* CSS is the **standard**, but others exist:
  + **Sass** – *Syntactically Awesome Stylesheets*
  + **Less** – *Leaner CSS*
* These build on top of CSS but require additional tools.

# Lecture 2:- How to add CSS

**🔹 1. Inline CSS**

**Definition:** CSS is written directly within an HTML tag using the style attribute.

**Syntax Example:**

<h1 style="color: blue;">Style me in blue</h1>

**When to Use:**

* When you want to style **only one element quickly**.
* Good for **testing or debugging**.
* **Not recommended** for full pages—hard to maintain.

**🔹 2. Internal CSS**

**Definition:** CSS is written inside a <style> tag within the <head> of the same HTML document.

**Syntax Example:**

<head>

<style>

h1 {

color: red;

}

</style>

</head>

**When to Use:**

* When styling **only one HTML page**.
* Better than inline for multiple elements.

**🔹 3. External CSS**

**Definition:** CSS is placed in a separate .css file and linked in the HTML using the <link> tag.

**Example Structure:**

**styles.css**

h1 {

color: green;

}

**index.html**

<head>

<link rel="stylesheet" href="styles.css" />

</head>

**When to Use:**

* Best for **multi-page websites**.
* Most maintainable and scalable approach.

**🧪 Exercise Recap**

You were asked to:

1. Create **three links** in index.html to:
   * inline.html
   * internal.html
   * external.html
2. Apply:
   * **Inline style** on inline.html (h1 in blue)
   * **Internal style** on internal.html (h1 in red)
   * **External style** on external.html (h1 in green using styles.css)

# Lecture 3:- What Are CSS Selectors?

CSS selectors **do not** refer to the word "cascade" directly, but they are part of how the **Cascade** in **Cascading Style Sheets (CSS)** works. The cascade is about *which styles apply when multiple rules could apply*, based on:

* **Specificity** (IDs > classes > elements),
* **Order of appearance**, and
* **Source** (inline > internal > external stylesheets).

Now let's break down the types of selectors from the lecture:

**🔹 1. Element Selector**

* Targets **all elements** of a given type.

h1 {

color: blue;

}

This targets all <h1> elements.

**🔹 2. Class Selector**

* Starts with a **dot (.)** and targets **elements with that class**.

.red-text {

color: red;

}

**In HTML:**

<h2 class="red-text">Red</h2>

You can apply the same class to **multiple elements**.

**🔹 3. ID Selector**

* Starts with a **hash (#)** and targets the **unique element** with that ID.

#main {

color: green;

}

In HTML:

<h2 id="main">Green</h2>

Each ID should be **used only once** per page.

**🔹 4. Attribute Selector**

* Targets elements by attribute or attribute value.

p[draggable] {

color: orange;

}

p[draggable="false"] {

color: red;

}

**🔹 5. Universal Selector**

* Uses \* to select **everything**.

\* {

margin: 0;

padding: 0;

}

**🧠 Where Does "Cascade" Come In?**

If you apply:

h2 {

color: blue;

}

.red-text {

color: red;

}

And your HTML is:

<h2 class="red-text">Example</h2>

Which color wins? → **Red**, because .red-text is more **specific** than just h2.  
This is **how the cascade works**.

# Lecture 4 css propetires

## Part 1: CSS Colors

**1. Basic Syntax Recap**

selector {

property: value;

}

**2. Color Properties**

* background-color: Sets the background color of an element.
* color: Sets the **text** color.

**Example:**

body {

background-color: red;

}

h1 {

color: blue;

}

**3. Types of Color Values**

**a. Named Colors**

* e.g. red, blue, olivedrab, cornflowerblue
* Full list: [MDN Named Colors](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value#named_colors)

**b. Hex Codes**

* Format: #RRGGBB (e.g., #5D3891)
* Each pair (RR, GG, BB) is a value from 00 to FF (0 to 255 in hex)

**c. RGB Format**

* Example: rgb(93, 56, 145) (same color as #5D3891)

**4. Color Palette Tool**

* Use [ColorHunt.co](https://colorhunt.co/) to pick color schemes.
* Each palette gives hex values—just copy and use in CSS.

**📝 Practice Task:**

* Use provided files with an h1 and h2.
* Change:
  + body background color (named color)
  + h1 text and background (named colors)
  + h2 text and background (hex colors)

## part 2:- font properties

CSS provides several properties to control the appearance of fonts on a webpage. These properties help define the **size**, **weight**, **family**, and more for your text content.

**🎯 1. Common Font Properties**

* color: Sets the color of the text.
* font-size: Controls the size of the text.
* font-weight: Determines how bold or light the text appears.
* font-family: Specifies the typeface (e.g., Arial, Times New Roman).

**🔤 2. font-size**

You can define font size using different **units**:

**➤ a) Absolute Units**

* px (pixels)
  + 1px ≈ 1/96 inch ≈ 0.26mm (very small square)
* pt (points)
  + 1pt ≈ 1/72 inch ≈ 0.35mm
  + Common in print and word processing (e.g., 12pt font in Word)

**➤ b) Relative Units**

* em: Relative to **parent element's** font-size
  + 1em = 100% of parent's font size
  + 2em = 200% of parent's font size
* rem: Relative to **root element’s** (<html>) font-size
  + 1rem = 100% of root font size
  + Consistent and recommended for maintainability

**➤ c) Named Sizes**

* small, medium, large, x-large, xx-large etc.

🔁 **em vs rem Example**:  
If html { font-size: 20px }, then:

* 2rem = 40px (always based on root)
* If footer { font-size: 80px }, then 2em inside footer = 160px (based on parent)

**💪 3. font-weight**

Controls how thick or light text appears:

* Keyword values:
  + normal, bold, lighter, bolder
* Numeric values (more granular):
  + Range from 100 (lightest) to 900 (boldest)

Example:

p {

font-weight: 600;

}

* lighter or bolder adjust weight relative to the parent.

**🧬 4. font-family**

Specifies the font to use. You can provide a **fallback list**:

p {

font-family: "Helvetica", sans-serif;

}

* "Helvetica": Specific font
* sans-serif: Fallback generic font category
  + **Sans-serif**: Clean fonts (no decorative "feet")
  + **Serif**: Fonts with small lines/strokes at ends (e.g., Times New Roman)

Tip: Always include a generic family (serif, sans-serif, monospace) as a fallback.

**🧪 Practice Tip**

Try editing font-size, em, rem, and root styles in a demo file like font-size.html to observe changes live in the browser.

# lecture 5:- Inspecting CSS

**🎯 Lecture 5: Inspecting CSS using Chrome Developer Tools**

**✅ Objective**

Understand how to inspect and manipulate CSS in real time using **Chrome Developer Tools** (DevTools) to debug styles, test changes, and learn from existing websites.

**🧰 What are Chrome Developer Tools (DevTools)?**

* A set of web development tools built directly into Google Chrome.
* Used for inspecting HTML/CSS, debugging JavaScript, and analyzing performance.
* Comes **pre-installed** with Chrome browser.

💡 For this course, Chrome is required to ensure consistency.

**🖥️ How to Open DevTools**

**1. Using Menu**

* Click the **three-dot menu** in Chrome → More Tools → Developer Tools

**2. Using Shortcuts**

* **Mac:** Command + Option + I
* **Windows:** Ctrl + Shift + I
* **Or:** F12 (on many systems)

**3. Using Right-Click**

* Right-click any element → Click **Inspect**

**🔍 Understanding the Elements Panel**

* When you inspect an element, the **HTML** is shown on the left, and **CSS styles** on the right.
* You can:
  + Select elements visually using the top-left **selector icon** in DevTools.
  + See which styles apply to which elements.
  + View the **source CSS file** the rule is coming from.

**🎨 Live Editing CSS**

* Add new rules: Click the + button in the Styles panel.
* Toggle rules on/off using the checkboxes or strikethrough.
* Changes are **temporary** and only visible to **you** in your browser.
* **Does NOT** affect:
  + Other users
  + The server
  + The actual source code

**🧠 Key Concepts**

* HTML elements have **default (user-agent)** styles.
* Custom CSS **overrides** default styles.
* If multiple rules apply, CSS uses **specificity** to determine which rule wins.
* **Strikethrough styles** in DevTools = overridden styles.

**🧮 Computed Tab**

* Shows the **final, calculated styles** for the selected element.
* Helpful for seeing:
  + Actual font sizes
  + RGB color values
  + Box model dimensions

**🎯 Bonus: CSS Overview Panel**

A handy tool inside DevTools for a summary of styles on the page.

**How to Access:**

* Click **three-dot menu inside DevTools** → More Tools → **CSS Overview**
* Click **Capture Overview**

**Shows:**

* Number of background/text colors
* Font families used
* Contrast issues
* Useful for analyzing or mimicking design from another website

**🧪 Hands-on Exercise**

Use this website to practice:  
**🔗** [**appbrewery.github.io/css-inspection**](https://appbrewery.github.io/css-inspection/)

**Tasks:**

1. Inspect elements like <h1>, <p>, <a>, and <body>.
2. Use DevTools to find:
   * Named background color of body → (e.g., aliceblue)
   * Font size of <h1>
   * Actual applied styles using Computed tab
   * Font family used on the page

**📝 Conclusion**

Inspecting CSS helps you:

* Debug layout issues
* Learn from existing websites
* Test new styles instantly
* Understand how default vs custom styles behave

# Lecture 6:- Box model

**CSS Box Model — Summary and Key Concepts**

**What is the Box Model?**

* Every HTML element is treated as a **box**.
* This box consists of several layers around the content, which control spacing and layout.
* The box is made up of:  
  **Content → Padding → Border → Margin**

**Components of the Box Model**

1. **Content**
   * This is the actual content of the element (text, images, etc.).
   * Width and height properties apply directly to the content area.
2. **Padding**
   * Space **inside** the box, between the content and the border.
   * Adds spacing *around* the content but *inside* the border.
   * Does **not** change the overall width and height of the element box, but visually pushes the border outward.
3. **Border**
   * A line surrounding the padding and content.
   * Can be styled with thickness, style (solid, dashed, dotted), and color.
   * Borders add to the visible box but **do not change the element’s width and height** — they extend outward.
4. **Margin**
   * Space **outside** the border, separating the element from other elements.
   * Controls the distance between neighboring boxes.
   * Margins on adjacent elements add up (e.g., two 10px margins become 20px total space).

**Width and Height**

* Controls the size of the content box.
* Can be set in **pixels** or **percentages** for responsiveness.

**Border Syntax**

* The border property has three parts:  
  border: <thickness> <style> <color>;  
  Example: border: 10px solid black;

**Border Width Shorthand**

* You can specify 1 to 4 values:
  + 1 value: applies to all sides
  + 2 values: first for top & bottom, second for left & right
  + 3 values: top, left & right, bottom
  + 4 values: top, right, bottom, left (clockwise order)

**Padding and Margin Shorthand**

* Works similarly to border-width with 1 to 4 values, using the same clockwise order.

**The Invisible Container — <div>**

* The div element creates a box that groups multiple elements together.
* It is invisible by default but can be styled using CSS.
* Helps to organize and style multiple elements as one unit.

**Debugging Tip — "Pesticide" Chrome Extension**

* Shows boxes around elements to visualize the box model during development.
* Helps debug spacing, padding, borders, and margins visually.

**Practical Exercise Summary**

* Create boxes with specific padding and borders.
* Use margin to align boxes corner-to-corner without gaps.
* Apply different border widths on different sides.
* Group elements inside divs for better structure and styling.

**Visual Box Model Diagram (for your mental image):**

[ Margin ]

[ Border ]

[ Padding ]

[ Content ]

Intermidiate css

# Lecture 1:- The Cascade - Specificity and Inheritance

**What is the CSS Cascade?**

* **Cascade** means when multiple CSS rules target the same element/property, the browser must decide which rule to apply.
* Think of it like water flowing down a series of pools — styles "cascade" down and the last, most important style “wins.”
* This system resolves conflicts between CSS rules by following a hierarchy.

**Four Key Factors in the CSS Cascade:**

1. **Position (Order of Rules)**
   * Later rules override earlier ones if they have the same specificity.
   * In a stylesheet, a rule written at the bottom overrides those above it.
2. **Specificity (Selector Strength)**
   * Different selectors have different strengths, calculated based on how specific they are.
   * The order from least to most specific:
     + Element selectors (li)
     + Class selectors (.class)
     + Attribute selectors ([draggable]) — often same or slightly higher than classes
     + ID selectors (#id) — most specific
   * The more specific selector overrides less specific ones.
3. **Type of Style Application**
   * Styles can come from:
     + External stylesheet (<link>)
     + Internal stylesheet (<style>)
     + Inline styles (style attribute)
   * Inline styles have higher priority than internal, which have higher priority than external.
4. **Importance (!important)**
   * Adding !important to a CSS rule makes it top priority, overriding even inline styles or IDs.
   * Use sparingly, as it can make debugging harder.

**How These Work Together**

* The browser first looks at **importance** (!important rules).
* If no !important, it looks at **type** (inline > internal > external).
* If same type, it looks at **specificity** (ID > attribute > class > element).
* If specificity is the same, it looks at **position** in the stylesheet (later overrides earlier).

**Example Recap**

If you have:

li { color: green; } /\* element selector \*/

.first-class { color: orange; } /\* class selector \*/

[draggable] { color: purple; } /\* attribute selector \*/

#first-id { color: yellow; } /\* ID selector \*/

* The list item with ID first-id will appear **yellow** because ID is most specific.
* An inline style (e.g., <li style="color: red;">) beats all of the above unless overridden by !important.

**Why Use This Knowledge?**

* Helps you debug why your styles are not applied.
* Helps you write efficient and maintainable CSS.
* Avoids confusion and conflicts in larger projects.

**Want to practice?**

* Try writing conflicting rules and predict the output.
* Experiment with specificity and !important.
* Use developer tools in your browser to inspect which rules apply.

# Lecture 2 :- CSS Selector Combinations

**1. Grouping Selector (comma ,)**

* Selects multiple selectors and applies the same style.
* Example:

h1, h2 {

color: blueviolet;

}

* Targets **all** h1 and h2 elements.

**2. Child Selector (>)**

* Selects elements that are **direct children** of a parent.
* Example:

.box > p {

color: firebrick;

}

* Targets all p elements that are **direct children** of any element with class .box.
* Does **not** select grandchildren or deeper descendants.

**3. Descendant Selector (space)**

* Selects elements that are descendants (children, grandchildren, etc.) of a parent.
* Example:

.box li {

color: blue;

}

* Targets all li elements that are anywhere inside an element with class .box, regardless of how deeply nested.

**4. Chained Selectors (no spaces)**

* Selects elements that match **all** specified selectors on the **same element**.
* Example:

h1#title.big.heading {

/\* style here \*/

}

* Targets an h1 element with id="title" and classes big and heading.

**Now, for your last challenge:**

**Select only these two bullet points and change their color to seagreen.**

You want to target **two specific li items** inside your HTML without adding classes or changing the HTML structure.

**How to approach this:**

* Inspect the HTML structure of those two li elements.
* See if they share a **common ancestor** with a class or ID.
* Use a **descendant selector** or a **combination of selectors** that selects exactly those two lis.

**Example based on typical HTML structure:**

Suppose your HTML looks like this simplified version:

<div class="box">

<ul>

<li>Item 1</li> <!-- select this -->

<li>Item 2</li> <!-- select this -->

<li>Item 3</li>

</ul>

</div>

If you want to target only the first two lis inside .box ul, but not the third, you can use:

* **CSS nth-child** selector combined with descendant selector:

css

.box ul li:nth-child(-n+2) {

color: seagreen;

}

* li:nth-child(-n+2) means select the first and second child elements.

**If the structure is different, maybe something like:**

<div class="box">

<ul>

<li>Item A</li>

<li>Item B</li>

<li>Item C</li>

</ul>

</div>

Use the same approach:

.box ul li:nth-child(1),

.box ul li:nth-child(2) {

color: seagreen;

}

**Key Points:**

* Use **descendant selectors** to narrow your scope to the .box container.
* Use **pseudo-classes** like :nth-child() to select specific list items.
* Don’t forget to separate multiple selectors with commas if you use separate lines.

**Final CSS for the challenge:**

.box ul li:nth-child(1),

.box ul li:nth-child(2) {

color: seagreen;

}

This will apply the seagreen color **only to the first two li elements inside .box ul** and leave others untouched.

# Lecture 3:- CSS positioning

**1. Static Positioning (default)**

* Elements flow naturally in the HTML document, one after the other.
* top, left, right, bottom properties **do not apply**.
* This is the default behavior, so you rarely set it explicitly.

**2. Relative Positioning**

* Moves the element **relative to its original static position**.
* The space where the element *would have been* remains reserved.
* You can use top, left, right, bottom to nudge the element around.
* Good for small tweaks in layout without breaking document flow.

**3. Absolute Positioning**

* Positions the element relative to the **nearest positioned ancestor** (ancestor with position other than static).
* If no such ancestor exists, it positions relative to the entire document (top-left corner of the page).
* Removes the element from the normal document flow, so it doesn't affect other elements.
* Often used for tooltips, modals, dropdown menus, or UI components needing precise placement.
* Remember to set the ancestor's position: relative (or other positioning) if you want the absolute child positioned relative to that ancestor.

**4. Fixed Positioning**

* Positions relative to the **viewport** (browser window), **not the page**.
* Element stays fixed in place even when the page scrolls.
* Commonly used for sticky headers, footers, or floating buttons.

**Bonus: z-index**

* Controls stacking order on the **z-axis** (depth).
* Only works on positioned elements (position not static).
* Higher z-index values sit **on top** of lower values.
* Default z-index is 0.
* Negative values push elements behind others.

**Extra Notes on Positioning & Box Model**

* position moves the element relative to something (original spot, ancestor, viewport).
* margin, padding, width, and height remain distinct and interact with positioned elements as expected.
* Positioning can override or ignore normal layout behavior.

**Your next step with the exercise:**

* Use your knowledge of positioning to build the rectangle with the red circle inside.
* Use position: relative for the container and position: absolute for the circle if you want to place it precisely inside.
* For the circle: use border-radius: 50% (not just pixels) for a perfect circle.

# Project :- css flag

Advanced css

# Lecture 1:-**CSS Display Property — Key Points**

* **block**
  + Takes full width of container (100% width)
  + Starts on a new line (stacked vertically)
  + You can set width and height freely
* **inline**
  + Takes only as much width as content needs
  + Flows with text (stays on the same line as other inline elements)
  + Cannot set width or height (ignored)
  + Examples: <span>, <a>, <strong>
* **inline-block**
  + Hybrid of block and inline
  + Can set width and height like block elements
  + Flows inline like inline elements (side by side)
  + Useful when you want to size an element but keep it on the same line with others
* **none**
  + Completely hides the element from the layout
  + Element takes no space, not rendered visually

**Real-life usage tips:**

* Use **block** for large sections, headers, paragraphs — things that should stack vertically
* Use **inline** for small pieces inside text, like styling part of a sentence
* Use **inline-block** when you want multiple boxes next to each other but still want to control their size
* Use **display: none** to toggle visibility (like hiding/showing menus or modals)

# Lecture 2:- **CSS Float Property?**

* **Purpose:** It allows an element (commonly images or blocks) to be taken out of the normal flow and positioned **left** or **right**, so that text or inline content wraps around it.
* Originally inspired by **print layouts** like newspapers where images often float with text wrapped around.

**Basic Usage**

img {

float: left; /\* makes the image float to the left \*/

}

* The surrounding text will wrap around the right side of this floated image.
* You can also do float: right; to make the image float on the right and text wrap on the left.

**Important Notes**

* When you float an element, it’s **removed from the normal document flow**.
* Other block elements will try to flow around it.
* By default, block elements take up full width, but floated elements shrink to their content width and let text wrap around them.

**Clearing Floats**

Sometimes you don’t want text or other elements to wrap around a floated element.

For example, if you have a footer below floated elements:

css

CopyEdit

footer {

clear: both; /\* prevents footer from wrapping around floated elements \*/

}

* clear: left; clears only left floats
* clear: right; clears only right floats
* clear: both; clears floats on both sides

This forces the footer (or any block) to start below any floated elements above it.

**Example**

Imagine the HTML:

<div class="cat-block">

<img src="cat.jpg" alt="cat" />

<p>Lorem ipsum ...</p>

</div>

<div class="dog-block">

<img src="dog.jpg" alt="dog" />

<p>Lorem ipsum ...</p>

</div>

<footer>

&copy; 2025 My Company

</footer>

CSS:

.cat-block {

float: left;

background-color: aquamarine;

width: 45%;

}

.dog-block {

float: right;

background-color: lightcoral;

width: 45%;

}

.cat-block img,

.dog-block img {

float: left;

margin-right: 10px; /\* space between image and text \*/

}

footer {

clear: both;

text-align: center;

margin-top: 20px;

}

* The .cat-block floats left, .dog-block floats right.
* Images inside each block also float left so that their text wraps nicely.
* The footer is cleared so it sits below both blocks.

**When to Use Float Today?**

* **Mainly for wrapping text around images or small elements.**
* Avoid using floats for complex layouts nowadays.
* Modern CSS layout tools like **Flexbox** and **CSS Grid** offer much better, predictable ways to build layouts without the quirks of floats.

**Summary**

| **Property** | **Values** | **What it does** |
| --- | --- | --- |
| float | left, right | Floats element left or right, text wraps around |
| clear | left, right, both | Stops element from wrapping around floated elements |

# Lecture 3:-**Why Responsive Design?**

* Screens come in many sizes: desktops, laptops, tablets, phones.
* We want websites to **look good and function well on all of them**.
* Responsive design means the layout **adapts automatically** depending on the screen width.

**How to Test Responsiveness?**

* Resize your browser window by dragging its edges or corners.
* Notice how website elements **change their layout or visibility** based on width.
* For example, navigation menus collapse into a hamburger menu on smaller screens.

**Four Main Methods to Create Responsive Layouts**

**1. Media Queries**

* CSS rule blocks that apply only when the screen size matches certain conditions.
* Syntax:

css

CopyEdit

@media (max-width: 600px) {

/\* CSS here applies only when viewport width ≤ 600px \*/

}

* Allows you to customize styles for different screen widths.
* Example: Hide navigation links and show a menu icon on small screens.

**2. CSS Grid**

* A 2D layout system that lets you define rows and columns.
* You can create complex grid layouts with precise control over rows, columns, gaps, and spanning.
* Example:

css

CopyEdit

.container {

display: grid;

grid-template-columns: 1fr 1fr; /\* two equal columns \*/

grid-template-rows: 100px 200px 200px;

gap: 30px;

}

.first {

grid-column: span 2; /\* spans across both columns \*/

}

* Great for complex layouts where both rows and columns need controlling.

**3. Flexbox**

* A 1D layout system for arranging items in a row or column.
* Flex items can grow or shrink based on their flex properties.
* Example:

css

CopyEdit

.container {

display: flex;

}

.item {

flex: 1; /\* equal width \*/

height: 100px;

}

.item-large {

flex: 2; /\* twice as wide \*/

}

.item-small {

flex: 0.5; /\* half width \*/

}

* Perfect for simpler horizontal or vertical layouts.
* Automatically adapts to screen size as widths/heights are ratios of available space.

**4. Bootstrap Framework**

* An external CSS (and JS) library that provides prebuilt responsive components and a grid system.
* Based on Flexbox.
* Uses a 12-column grid layout:
  + .col-6 means the element takes up 6/12 = 50% of the container width.
  + Columns automatically adjust as the screen resizes.
* Comes with styled components like buttons, cards, navigation bars.
* Good for quick and consistent responsive designs without writing much CSS.

**Summary: Which to Use?**

* **Media Queries**: Great for fine-tuning responsiveness by conditionally applying CSS.
* **CSS Grid**: Best for complex two-dimensional layouts.
* **Flexbox**: Ideal for simpler, linear layouts.
* **Bootstrap**: Perfect if you want prebuilt responsive components and grid system.

If you want, I can help you with hands-on examples for any of these methods next — just say which one interests you! Or I can walk you through building a small responsive layout step-by-step.

**What do you want to explore next?**

* Media Queries deeper dive?
* CSS Grid layout example?
* Flexbox example?
* Using Bootstrap for responsiveness?

4.1-mini

# Lecture 4:-**CSS Media Queries — Notes**

**What are Media Queries?**

* A CSS technique to apply styles conditionally based on device characteristics (most commonly screen width).
* Allows creation of **responsive websites** that adapt layout and design to different screen sizes.
* Syntax uses the @media keyword followed by a condition called a **breakpoint**.

**Breakpoints**

* Define screen size ranges for applying specific styles.
* Two common types:
  + **max-width:** Applies styles when viewport width is **less than or equal** to a specified value.
    - Example: @media (max-width: 600px) targets screens **600px or smaller** (e.g., mobile phones).
  + **min-width:** Applies styles when viewport width is **greater than or equal** to a specified value.
    - Example: @media (min-width: 600px) targets screens **600px or larger** (e.g., tablets, desktops).

**Combining Breakpoints**

* You can combine conditions with the and keyword.
* Example:

css

CopyEdit

@media (min-width: 600px) and (max-width: 900px) {

/\* Styles for screens between 600px and 900px \*/

}

* This targets a **specific range** of screen widths.

**Media Types**

* Default: screen — targets display screens (monitors, phones, tablets).
* Other types:
  + print — applies styles only when printing the webpage.
* Usually, screen can be omitted because it’s the default.

**Typical Use Case Example**

css

CopyEdit

div {

width: 200px;

height: 200px;

background-color: blue;

}

/\* When screen width is 600px or less (mobile), change size \*/

@media (max-width: 600px) {

div {

width: 100px;

height: 100px;

}

}

**Common Device Breakpoints (Example)**

| **Device** | **Width Range (px)** | **Background Color** |
| --- | --- | --- |
| Mobile | 319px - 480px | Salmon |
| Tablet / iPad | 481px - 1200px | Powderblue |
| Laptop | 1201px - 1600px | Limegreen |
| Desktop | 1601px and above | Seagreen |

**How to Test Responsiveness**

* Use **Chrome Developer Tools**:
  1. Open Developer Tools (F12 or Right-click → Inspect).
  2. Click the **Toggle device toolbar** button (phone/tablet icon or Ctrl + Shift + M).
  3. Select preset devices like iPhone SE, iPad, or use the responsive mode to adjust width manually.
* Observe how your page styles change with different viewport widths.

**Important Notes**

* Media queries override default CSS rules when conditions are met.
* Use max-width to target **small screens and below**.
* Use min-width to target **larger screens and above**.
* Combine min and max-width to target **specific ranges**.
* Avoid overly complex queries unless necessary; most responsive designs use simple breakpoints.
* The screen keyword is optional but helps specify media type explicitly if needed.
* The print media query can provide a different style for printed pages.

**Additional Resources**

* [MDN Web Docs: Using Media Queries](https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries)
* W3Schools: CSS Media Queries

# **Project Summary: Responsive Website for a Creative Design Agency**

**Project Goal**

* Build a **simple, clean, responsive website** for a Creative Design Agency (dev.com).
* Show two blocks of text describing the agency’s work.
* Include a footer with the company motto: **"Create, Develop Design."**
* Make sure the website **scales responsively**, especially for smaller screens like iPhones.
  + On wide screens: The two text blocks appear **side by side**.
  + On small screens (around 600px width or less): The two blocks **stack vertically** and center.

**Key Concepts to Use**

* **Display property** (block, inline-block)
* **Floats** (float: left; and float: right;)
* **Media queries** for responsiveness

**Starting Point**

* Use the provided index.html file.
* It contains basic HTML structure, fonts, images, and text but lacks layout styling.
* Your task is to add CSS to style and layout the page responsively.
* Preview your progress in a browser or using a live preview.

**Layout Details**

* On desktop/wider screens:
  + Two blocks of text side by side (one floated left, one floated right).
  + An image aligned next to text in each block.
* On mobile/smaller screens:
  + Blocks stack vertically (one on top of the other).
  + Blocks centered and width set to 100%.
  + Text is justified for cleaner alignment.

**Tips & Important Notes**

* To make the site responsive, use a media query targeting max-width ~680px.
* Inside the media query:
  + Change the .card or text block elements to display: block and width: 100%.
  + Adjust text alignment to justify.
* Use float: left and float: right on text blocks to position them side by side on larger screens.
* To style a part of the text differently, wrap it in a <span> with a unique class (e.g., .creative) and target that in CSS.
* The solution uses the CSS property object-fit for images to maintain aspect ratio nicely—look it up if unfamiliar.
* Don’t peek at the solution code until you finish your version!

**Workflow & Encouragement**

* Try to recreate a **similar layout**, but feel free to be creative.
* Practice what you learned about:
  + Floats
  + Display types
  + Responsive design with media queries
* Share your final work with a screenshot in the Q&A if you want to show it off.
* If stuck, use developer docs (like MDN) to research unfamiliar properties or techniques.