02. CSS

Timeline

- **Schedule**
  - **Week 1:** HTML & **CSS** & Javascript
  - **Week 2:** Javascript
  - **Week 3:** React & Redux
  - **Week 4:** Individual project(JavaScript)
  - **Week 5-6:** Node.js, Database, Typescript, Security, Testing, CICD, GraphQL, Next JS etc…
  - **Week 7-8:** Group project (TypeScript)

- Resume
- Mock interviews
- Marketing & Job

Attention!

This training is **interview-driven**.

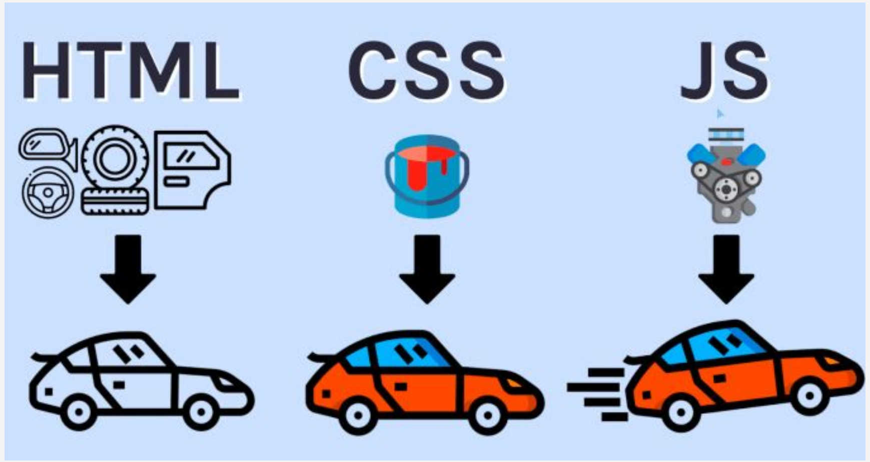
To become a qualified developer, *ChatGPT* and *YouTube* will always be your best friends.

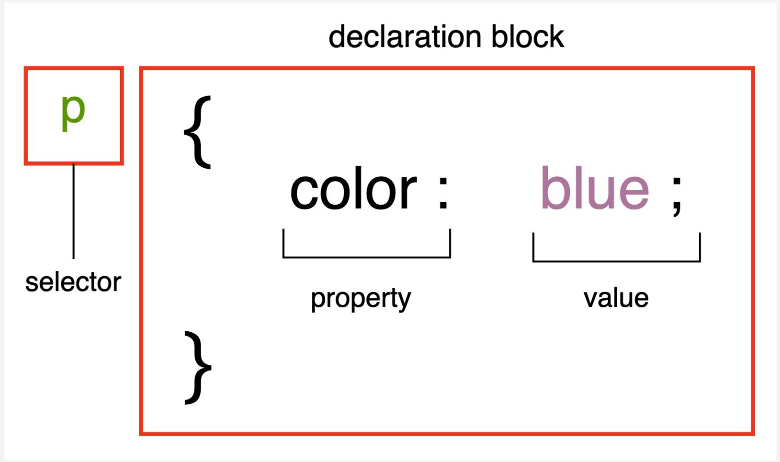
Outline

- CSS & Inserting CSS
- Selectors
- Box Model
- Display & Position
- Z-Index
- Flexbox & Grid
- Responsive Design
- Accessibility, Inclusion

CSS

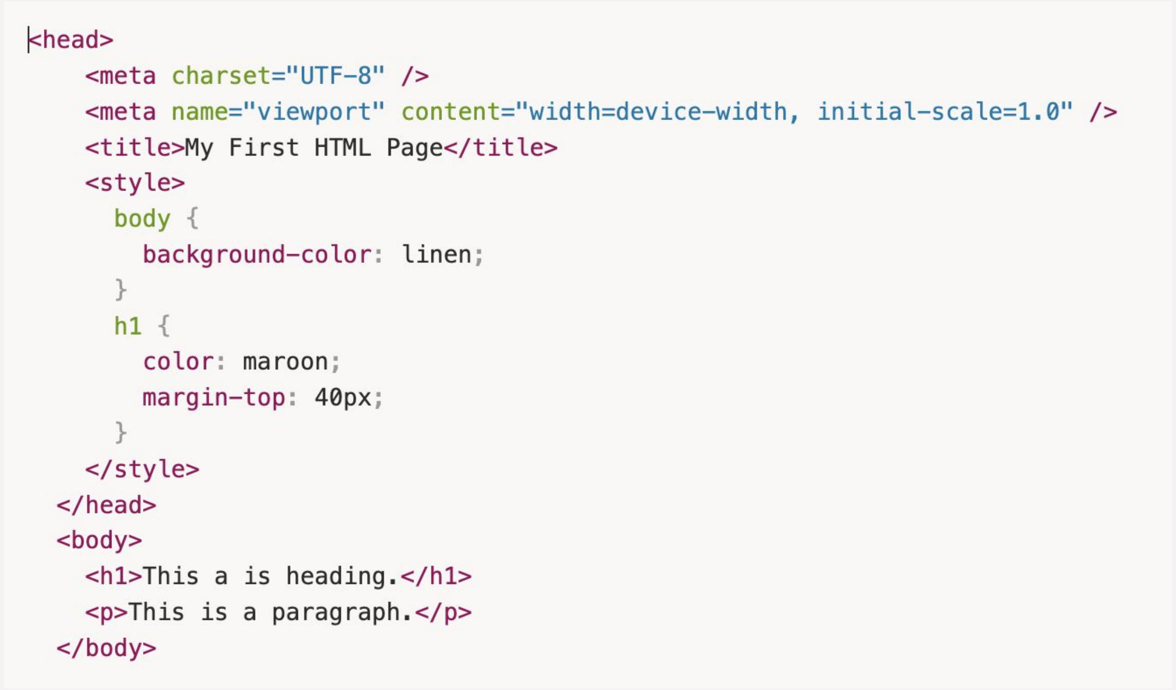
- Cascading Style Sheets (CSS)
  - Style sheet language
  - Layout and customization of HTML elements
  - Syntax: Selector + Declaration block





Inserting CSS: Internal CSS

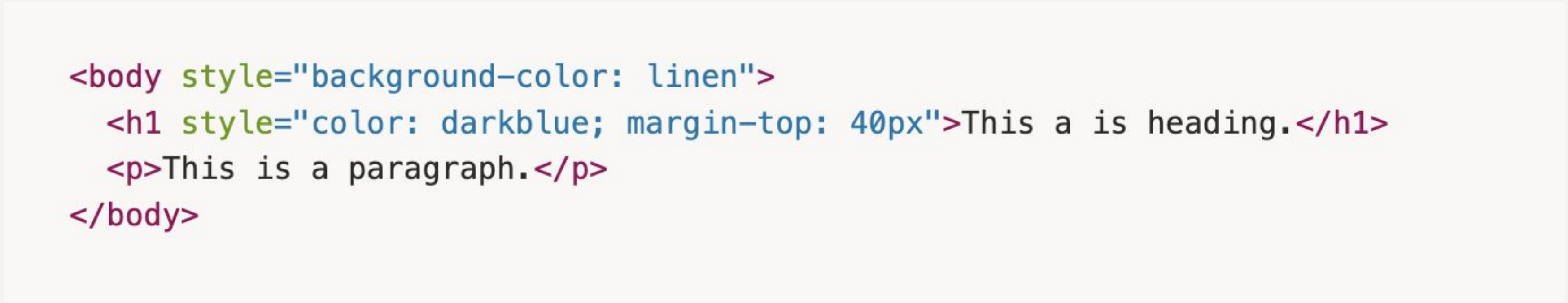
- An internal CSS is defined in the <head> section of an HTML page, within a <style> element



```html
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My First HTML Page</title>
    <style>
      body {
        background-color: linen;
      }
      h1 {
        color: maroon;
        margin-top: 40px;
      }
    </style>
  </head>
  <body>
    <h1>This a is heading.</h1>
    <p>This is a paragraph.</p>
  </body>
```

Inserting CSS: Inline CSS

- Inline CSS applies styles directly to HTML elements using the style attribute



```html
<body style="background-color: linen">
  <h1 style="color: darkblue; margin-top: 40px">This a is heading.</h1>
  <p>This is a paragraph.</p>
</body>
```

Inserting CSS: External CSS

- Separate **.css** file

```html
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>My First HTML Page</title>
  <link rel="stylesheet" href="styles.css" />
</head>
<body>
  <h1>This a is heading.</h1>
  <p>This is a paragraph.</p>
</body>
```

```css
body {
  background-color: linen;
}
/* element selector*/
p {
  font-style: italic;
}
```

Selectors: Simple Selectors

- **element** selector: p - Selects all elements of a given type
- **id (#)** selector: #idName - Targets a specific element by its ID
- **class (.)** selector: .className - Targets elements with a specific class

```html
<body>
  <div class="container">
    <h1 id="heading">This is a heading.</h1>
    <p id="paragraph">This is a paragraph.</p>
  </div>
  <div class="container">
    <span>This is a span</span>
  </div>
</body>
```

```css
/* element selector*/
p {
  font-style: italic;
}

/* id selector*/
#heading {
  color: maroon;
}

.paragraph {
  font-weight: lighter;
}

/* class selector */
.container {
  padding: 20px;
  font-weight: 700;
}
```

Selectors: Combinations

- Descendant combinator: (**Space**)
    - all elements inside another element, at any level
- Child Selector (**>**)
    - Only the direct children, not nested deeper
- Next Sibling Selector (**+**)
    - Selects an element that is immediately after another, at the same level
- subsequent Sibling Selector (**~**)
    - Selects all siblings after a specific element, as long as they are siblings (same parent)

Selectors: Attribute selectors

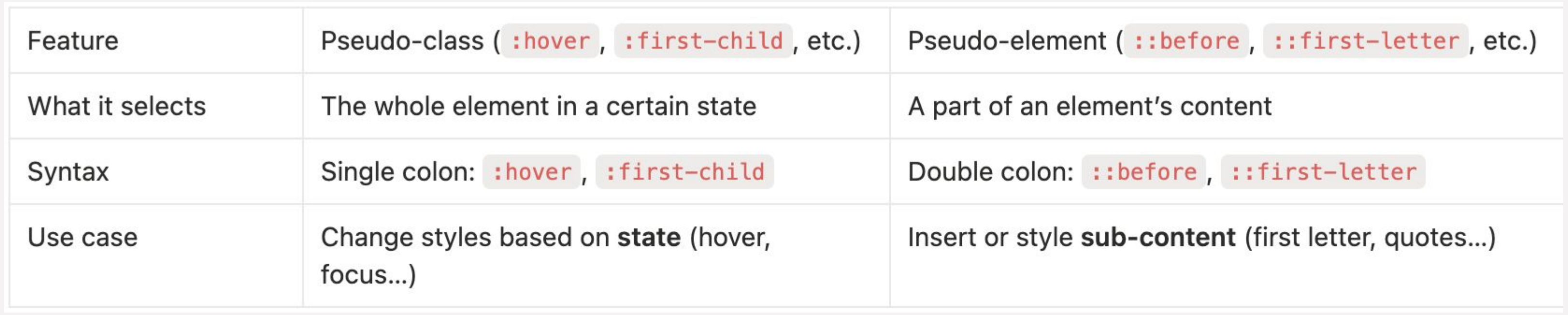
- **Target elements based on the attribute**
  - **[attr]:** Selects any element that has the given attribute, regardless of its value
  - [attr="value"]: Selects elements whose attribute exactly equals that value
  - [attr^="prefix"]: Selects elements whose attribute value starts with “prefix”
  - [attr$=".suffix"]: Selects elements whose attribute value ends with “.suffix.”
  - [attr*="substring"]: Selects elements whose attribute value contains “substring.”
  - [attr~="word"]: Selects elements whose attribute is exactly “word” or contains the word “word”
  - [attr|="value"]: Selects elements whose attribute is exactly “value” or begins with “value-.”

Selectors: pseudo class

- Define a special **state** of an element
- **Interactive states**
  - :hover: when the mouse pointer is over an element
  - :focus: when the element has keyboard or programmatic focus
    - e.g. clicked or tabbed into)
  - :active: while the element is being activated
    - e.g. a button is being pressed
- **Structural position**
  - :first-child: an element that is the first child of its parent
  - :last-child: an element that is the last child of its parent
  - :nth-child(odd) or :nth-child(3n): elements matching a numeric pattern
    - odd positions, every third, etc.

Selectors: pseudo element

- Define a specific **part** of an element
  - ::before: Inserts generated content **before** an element's real content.
  - ::after: Inserts generated content **after** an element's real content.
  - ::first-letter: Targets and styles the first letter of a block-level element.
  - ::first-line: Targets and styles the first line of text in a block-level element.
  - ::selection: Styles text when the user selects it (e.g. with the mouse).
  - ::placeholder: Styles the placeholder text inside an <input> or <textarea>



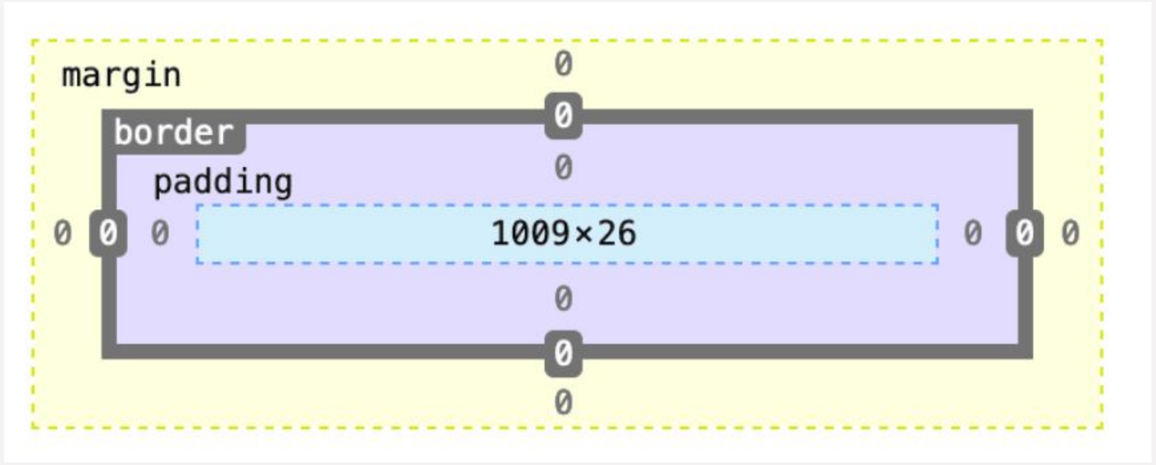
Feature	Pseudo-class (`:hover`, `:first-child`, etc.)	Pseudo-element (`::before`, `::first-letter`, etc.)
What it selects	The whole element in a certain state	A part of an element's content
Syntax	Single colon: `:hover`, `:first-child`	Double colon: `::before`, `::first-letter`
Use case	Change styles based on **state** (hover, focus...)	Insert or style **sub-content** (first letter, quotes...)

Cascade

- Inline CSS > Internal CSS > External CSS
- If two rules have equal specificity, the one declared **last** in the CSS wins
- !important can **override** everything

Box Model

- **Box Model:** Every HTML element is a rectangular box that consists of
  - **Content**: The actual text, image, or content of the element.
  - **Padding**: the transparent space **surrounding** the content
  - **Margin**: the transparent space **outside** the border
  - **Border**: the border surrounding the padding and content

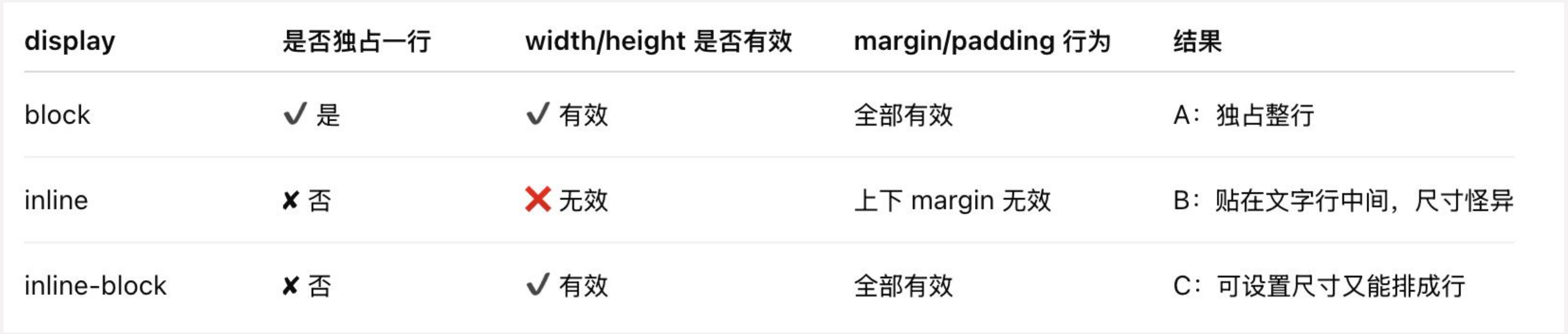


Padding and Margin

- **Padding**
  - Creates space **inside** the box, between the border and the content.
  - Inline elements don’t have padding-top or padding-bottom
- **Margin**
  - creates space **outside** the box, pushing it away from surrounding elements
  - Can have positive or negative lengths
  - **margin: 0 auto** -> center a block element horizontally
- Shorthand Notation
  - top right bottom left
  - top and bottom left and right

Display

- Specifies how an element should be displayed
  - **display: block**: Generates a block-level box
  - **display: inline**: Generates an inline-level box
    - Ignores width/height
  - **display: inline-block:** Combines inline flow with block-box features
    - Like inline, but allows setting width and height
  - **display: none**: The element is removed



display	是否独占一行	width/height 是否有效	margin/padding 行为	结果
block	✓ 是	✓ 有效	全部有效	A：独占整行
inline	✘ 否	❌ 无效	上下 margin 无效	B：贴在文字行中间，尺寸怪异
inline-block	✘ 否	✓ 有效	全部有效	C：可设置尺寸又能排成行

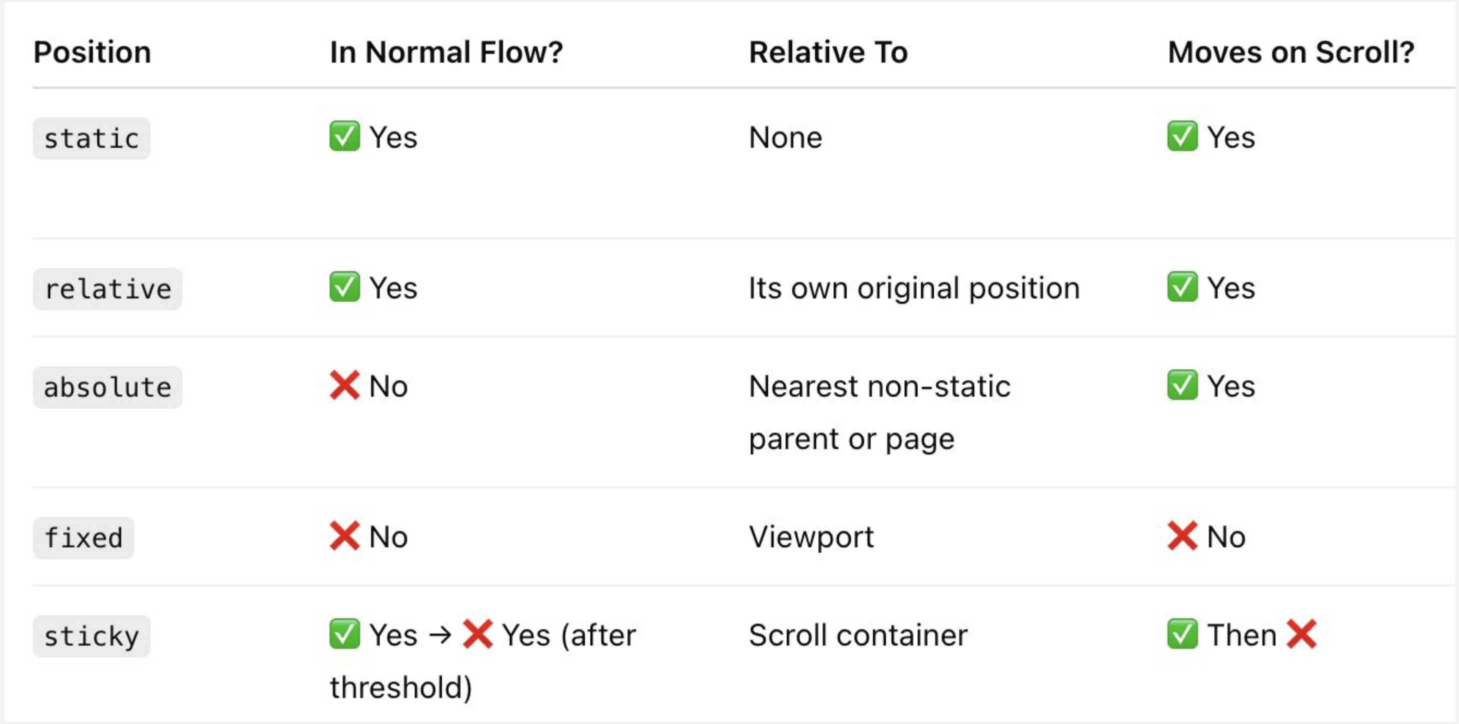
Position

- position: static default
  - Follows normal document flow.
- position: relative
  - Positioned relative to its normal position
  - You can shift it using top/right/bottom/left **relative to where it would’ve been**
- position: absolute
  - Positioned via top/right/bottom/left **relative to its nearest ancestor** that has a non-static position (or the viewport if none)
  - If no ancestor, it uses the document body, and moves along with page scrolling

Position

- position: fixed
  - Always positioned relative to the viewport, it stays put even when you scroll.
- position: sticky
  - Positioned based on the user's scroll position.
  - Toggles between relative and fixed .
  - It is positioned relative until a given offset position is met in the viewport, then it "sticks" in place (like position:fixed)

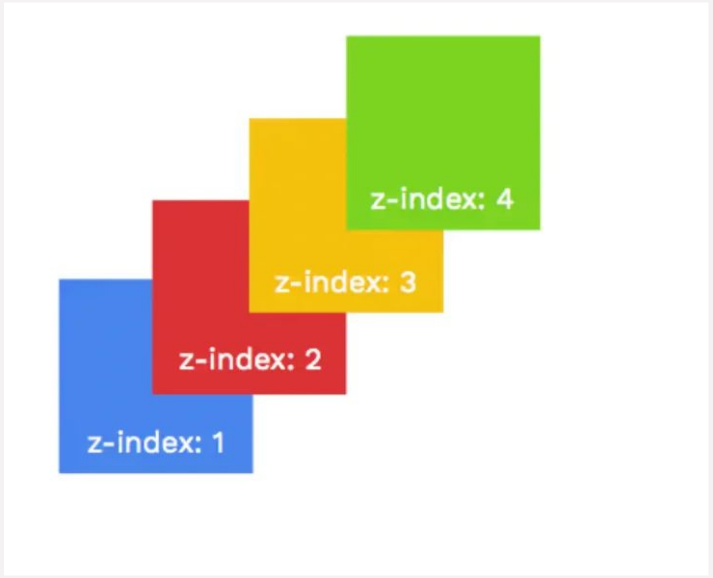
Position



Position	In Normal Flow?	Relative To	Moves on Scroll?
`static`	✅ Yes	None	✅ Yes
`relative`	✅ Yes	Its own original position	✅ Yes
`absolute`	❌ No	Nearest non-static parent or page	✅ Yes
`fixed`	❌ No	Viewport	❌ No
`sticky`	✅ Yes → ❌ Yes (after threshold)	Scroll container	✅ Then ❌

Z-Index

- Determines the vertical stacking order of elements that overlap
- Elements with higher z-index values appear on top
- Works on anything **other than** the default **position:static**



Flexbox (Flexible Box Layout)

- Flexbox is a **one-dimensional** layout method for **arranging** items in a single direction
- https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- **Flex container**
  - display: flex
- **Flex properties**
  - flex-direction: row, row-reverse, column, column-reverse
  - flex-wrap:
    - nowrap (default), If no enough space, a scrollbar appears to display any overlap
  - flex-flow: flex-direction + flex-wrap
  - justify-content: alignment
  - align-items: how items are laid out along the cross axis
  - align-content: property aligns the flex containers' lines

Grid

- CSS Grid is a **two-dimensional** layout system for the web. It lets you design web layouts using rows and columns
- https://css-tricks.com/snippets/css/complete-guide-grid/
- display: grid;
- Defining columns and rows
    - grid-template-columns: define the number and size of columns
    - grid-template-rows: defines the number and size of rows
    - grid-column: start / end; horizontally (across columns).
    - grid-row: start / end;
    - justify-content

Grid

- grid-template-areas
  - define named areas in the grid. Makes the layout more readable
- Auto dimensions:
  - size is based on the content
- repeat() function:
  - repeat(3, 1fr); repeat(2, 100px);
  - fr: fractional unit — divides the space proportionally
- minmax()
  - minimum and maximum size range

Centering - Horizontal

- Inline: text-align: center;
- Block
    - margin: 0 auto;
    - Flexbox
        - display: flex;
        - justify-content: center;

Centering - Vertical

- Flexbox
  - display: flex;
  - align-items: center;
- Grid
  - display: grid;
  - place-items: center;

Centering - vertically and horizontally

- Centering vertically and horizontally
  - Flexbox:
    - display: flex;
    - justify-content: center;
    - align-items: center;
  - Grid
    - display: grid;
    - place-items: center;

Bootstrap and Tailwind

- **Bootstrap** is a free and **open-source** front-end framework that provides **predefined CSS** and JS components
  - Common UI components (modals, navbars, alerts)
  - Don’t need much customization
  - https://getbootstrap.com/docs/5.3/components/buttons/#button-plugin
- **Tailwind** CSS is a **utility-first** CSS framework. Instead of giving you predefined components, it provides lots of small, single-purpose utility classes
  - Utility classes for flex, grid, colors, spacing, etc
  - Highly custom UIs
  - https://tailwindcss.com

Media Queries

- Make layouts adapt to different device sizes
  - Syntax: @media media-type and (condition) {/* CSS rules */}
  - **Breakpoint:** the condition for when a media rule takes effect
    - https://mui.com/material-ui/customization/breakpoints/
    - https://getbootstrap.com/docs/5.3/layout/breakpoints/

Responsive Design

- Web page's layout and content **adapt to different screen sizes and devices**
  - Use **media queries** to set **breakpoints**(change styles based on screen width/height/orientation)
  - Use **fluid layouts**
  - **Flexbox**, **Grid**
  - **Mobile-first approach**: start designing for small screens first, then scale up
  - Use flexible images (max-width) so they resize within containers
  - Tools:
    - **DevTools** in browsers for testing
    - Frameworks like Bootstrap or Tailwind CSS for rapid responsive development

Accessibility and Inclusion

- Accessibility (**a11y**):
  - Focuses on making sure people with **disabilities** can use your site
    - Clear labels and instructions
    - Alt text for images
    - Focus indicators for keyboard users
    - Sufficient color contrast
- **Inclusion**:
  - Everyone feels respected, safe, and represented
    - Localization(**l10n**), Internationalization(**I18n**)
    - Avoiding gender stereotypes in color choices (e.g., pink for girls, blue for boys)
    - Using inclusive language (e.g., “Everyone” instead of “Guys”)
    - Supporting low-bandwidth connections with lightweight images

Additional materials

- CSS Tutorial: https://www.w3schools.com/w3css/defaulT.asp

Group Info

Group 1	Tingwei	Liu
	Haoyu	Li
	Shaolong	Li
Group 2	Sijun	Hua
	Weihang	Guo
	Chieh Jui	Lee
	Weiren	Feng
Group 3	Chunjingwen	Cui
	Huimin	Mu
	Rongwei	Ji
Group 4	Jiahao	Liang
	Lingyu	Xu
	Di	Wang
Group 5	Ziyue	Fan
	Fangqin	Li
	Ruiqi	Wang
	Ruohan	Wang
Group 6	Haozhong	Xue
	Kanghong	Zhao
	Chia Hsiang jia xiang	Wu
	jingwei	Ma

Homework - Requirement

- ❑ 全程recording: 八股闭眼 & 摘耳机，无AI辅助
- ❑ 全程recording: Coding可适当AI辅助，但需写完(边解释边写)，模拟真实面试情景
- ❑ 录音提交地址: https://drive.google.com/drive/folders/1Mrm341uOIS8c2Ty6NwYvvSybHa6hESem?usp=drive_link
- ❑ 提交格式: 小组+日期，例如Group1_12/07/2025
- ❑ 提交截止时间: due第二天5PM
- ❑ 第二天上课会随机抽查前一天的coding作业，现场share屏幕讲解，be prepared

Homework - Class Code

- Write the code as taught in class, take a screenshot of your code, and post it in the Wechat group. **It’s due the same night.**

Homework - W3 schools CSS Quiz

https://www.w3schools.com/css/css_quiz.asp

Please attach the screenshot of your results and post it in the Wechat group.

Homework - Short Answer Questions

1. What is CSS?
2. How do you link a CSS file to an HTML document?
3. What is block element? How is it different from inline, and inline-block elements?
4. What is the difference between pseudo-class and pseudo-element?
5. What is the difference between the child combinator and the descendant combinator?
6. What are two ways that we can make an element invisible? What is the difference?
7. What is the Box Model? Describe each part.
8. What is the usage of !important? What are some use cases?
9. What does z-index do?

Homework - Short Answer Questions

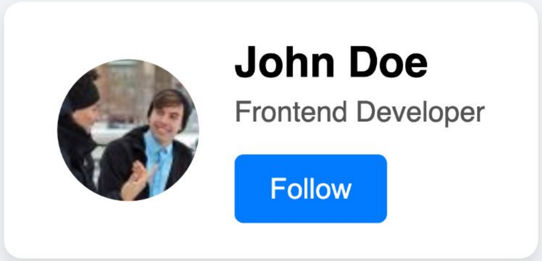
10. Can padding and margin be negative?
11. How do you center a block element with CSS?
12. What are grid items? Can you explain some grid item properties?
13. What is a flex container? Can you explain some flex container properties?
14. What is responsive web design? How do we achieve this?
15. What is Accessibility and Inclusion?

Homework - Coding Question 1

Task: Create a Name card

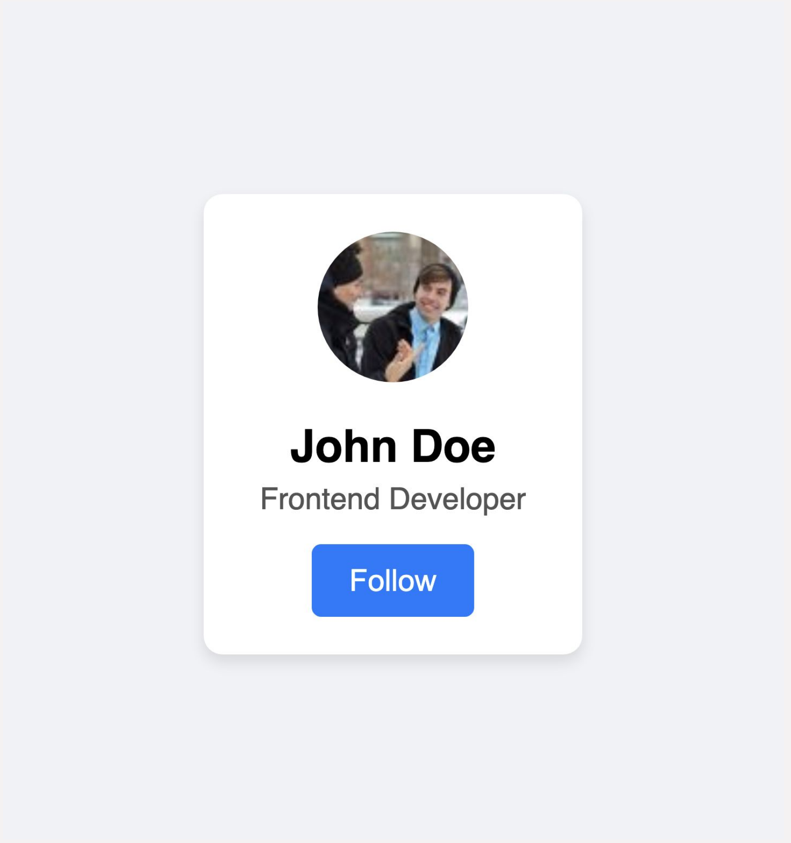
- ❑ Overall background color is #f0f2f5, and font-family is sans-serif
- ❑ Background color of the name card is white
- ❑ Color of the button is #007bff
- ❑ When hovering on Follow button, the color will darken (choose a color you like). The cursor will be a pointer
- ❑ When hovering on the avatar, its size will be a little larger.
- ❑ The avatar image can be found here: https://i.pravatar.cc/80
- ❑ Implement responsive design so that elements switch to a vertical (column) layout.
  - ❑ hint: @media (max-width: 768px)

Homework - Coding Question 1


John Doe

Frontend Developer

Homework - Coding Question 1



Homework - Coding Question 2

Task: Build a simple “Profile Card” page with Tailwind

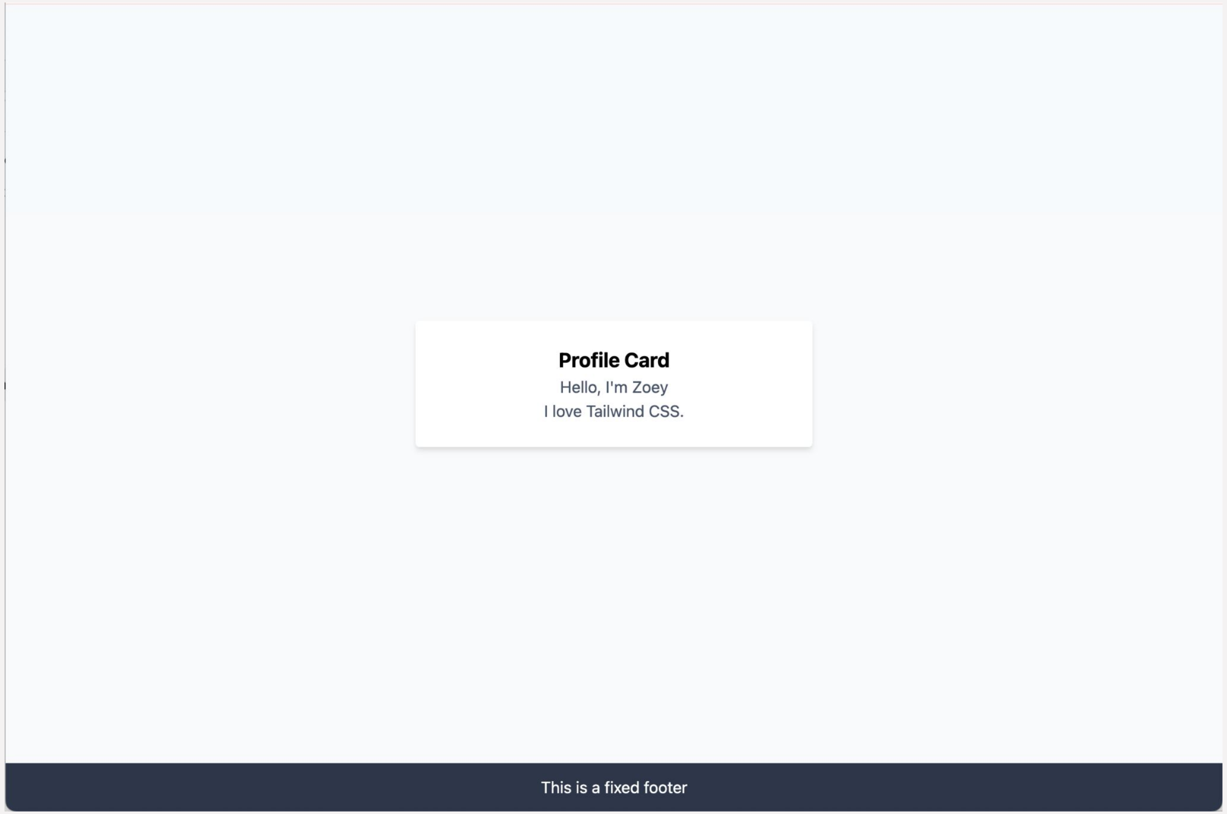
- ❑ A centered card with:
  - ❑ Background color
  - ❑ Padding and margin
  - ❑ A title and short paragraph
  - ❑ Hover effect to change background color
  - ❑ stay centered both vertically and horizontally in the viewport
  - ❑ The card should be centered in the remaining visible area (i.e., it should not be overlapped by the footer).
- ❑ A fixed footer at the bottom:
  - ❑ Keep the footer at the bottom of the screen
  - ❑ Full width with background

Homework - Coding Question 2

Starter Code:

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <meta charset="UTF-8" />
                <title>Tailwind Assignment</title>
                <script src="https://cdn.tailwindcss.com"></script>
        </head>
        <body class="bg-gray-100 flex flex-col min-h-screen">
                <!-- Put your code here -->
        </body>
</html>
```

Homework - Coding Question 2



Profile Card

Hello, I'm Zoey

I love Tailwind CSS.

This is a fixed footer

Homework - Coding Question 3

Task: Using the same HTML, write CSS to achieve three different centering effects:

```
<div class="container">
    <div class="box">Center</div>
</div>
```

1. Horizontal centering: center the box horizontally in the container.
2. Vertical centering: center the box vertically in the container.
3. Horizontal + Vertical centering: center the box both horizontally and vertically in the container