THRIVE
Women for Women

Introduction to CSS II

Lecture Flow

- *Introduction to CSS Layout*
- *Positioning elements*
- *Layout with “display” and “position”*
- *Introduction to Flexbox and CSS Grid*
- *Responsive tips & Techniques*
- *Q&A*

Introduction to CSS Layout

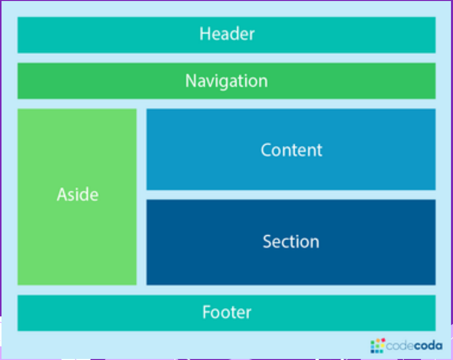
Why Layout matters

- *Controls how content is arranged on a webpage*
- *Good layout improves readability, usability, and visual flow*
- *A well-structured layout helps users find information faster*

Bad
WEBSITE
VS
Good
WEBSITE

✓ *HTML structure and CSS layout*

- ***HTML provides the structure and meaning***
- ***CSS provides the visual positioning, spacing, and styling***
- ***Clean HTML = easier and more predictable CSS layout***
- ***CSS relies on the box model to understand spacing and alignment***


Header
Navigation
Aside
Content
Section
Footer
codecoda

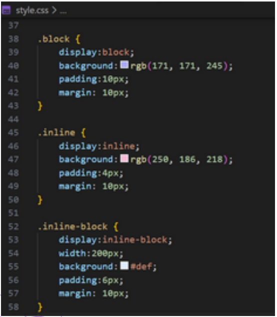
📦 What Does "Positioning" Mean in CSS?

- *It controls where an element appears in the document and how it interacts with other elements.*

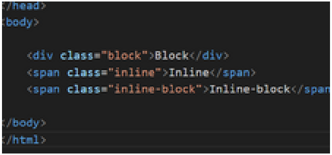
- *While normal flow automatically stacks elements from top-to-bottom, positioning gives developers the power to adjust spacing, overlap elements, and create more complex layouts.*

✨ *Display: Inline, Block, Inline-Block*

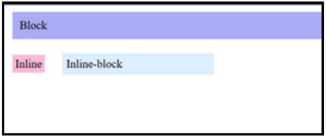
- ***display: block - fills width, starts new line.***
- ***display: inline - flows with text, no width/height.***
- ***display: inline-block - like inline but can set width/height.***



```
style.css > ...

.block {
    display:block;
    background: rgb(171, 171, 245);
    padding:10px;
    margin: 10px;
}

.inline {
    display:inline;
    background: rgb(250, 186, 218);
    padding:4px;
    margin: 10px;
}

.inline-block {
    display:inline-block;
    width:200px;
    background: #def;
    padding:6px;
    margin: 10px;
}
```



```
</head>
<body>

    <div class="block">Block</div>
    <span class="inline">Inline</span>
    <span class="inline-block">Inline-block</span

</body>
</html>
```



Block

Inline Inline-block

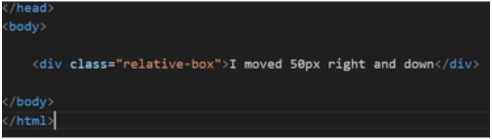
⭐ Position Overview

1. **position: static (default positioning)**

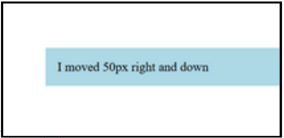
   - It follows the normal flow of the page. You cannot move static elements using top, left, right, or bottom properties.

2. position: relative 💫

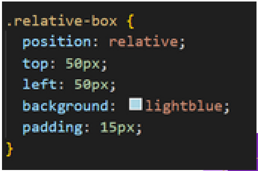
It keeps the element in normal flow but allows you to shift it using top, left, right, and bottom. The space it originally occupied remains, even after movement.



```
</head>
<body>

    <div class="relative-box">I moved 50px right and down</div>

</body>
</html>
```



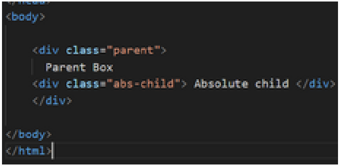
I moved 50px right and down



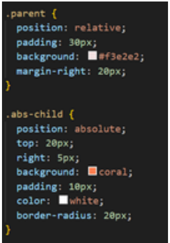
```
.relative-box {
  position: relative;
  top: 50px;
  left: 50px;
  background: lightblue;
  padding: 15px;
}
```

3. position: absolute 💫

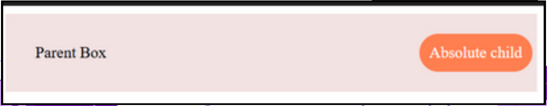
Removes an element from the normal flow so it no longer affects other elements. It positions itself relative to the nearest ancestor with position: relative, absolute, or fixed.
If none is found, it positions relative to the viewport.



```
<body>

    <div class="parent">
     Parent Box
    <div class="abs-child"> Absolute child </div>
    </div>

</body>
</html>
```

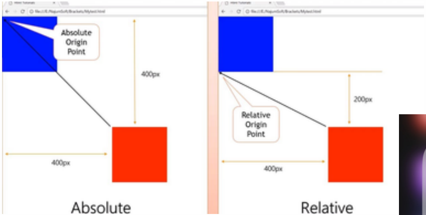


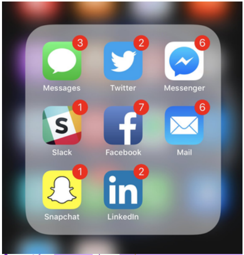
```
.parent {
  position: relative;
  padding: 30px;
  background: #f3e2e2;
  margin-right: 20px;
}

.abs-child {
  position: absolute;
  top: 20px;
  right: 5px;
  background: coral;
  padding: 10px;
  color: white;
  border-radius: 20px;
}
```



Parent Box

Absolute child

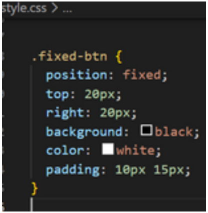




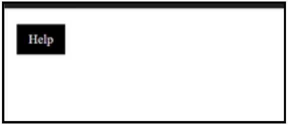
THRIVE
Women for Women

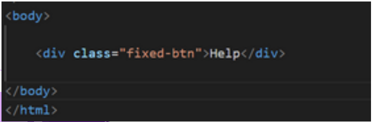
4. position: fixed 💫

Fixed positioning attaches an element to the viewport, meaning it always stays visible even when the user scrolls. This makes it ideal for sticky headers, floating action buttons, or “Back to Top” controls.



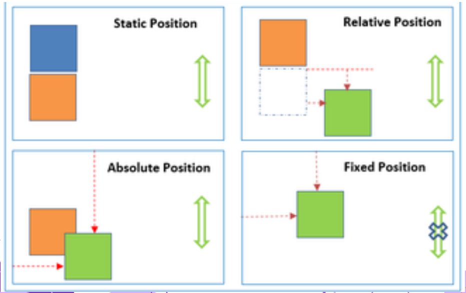
```
style.css > ...

.fixed-btn {
  position: fixed;
  top: 20px;
  right: 20px;
  background: ■black;
  color: □white;
  padding: 10px 15px;
}
```





```
<body>

    <div class="fixed-btn">Help</div>

</body>
</html>
```

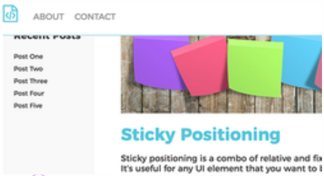
Compare and Contrast



5. position: sticky 💫

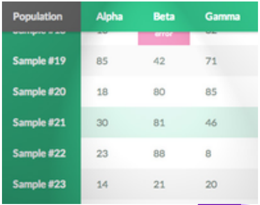
Sticky positioning behaves like relative until the user scrolls past a threshold, after which it sticks to a specified position. This is especially useful for table headers, section titles, or navigation inside long pages.

🎁 Unlike fixed positioning, sticky only sticks within its parent container.



Sticky Positioning

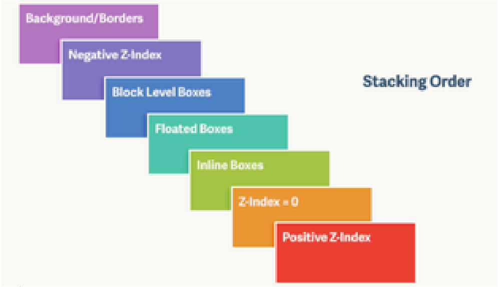
Sticky positioning is a combo of relative and fix
It's useful for any UI element that you want to b



Population	Alpha	Beta	Gamma
[illegible]	[illegible]	error	[illegible]
Sample #19	85	42	71
Sample #20	18	80	85
Sample #21	30	81	46
Sample #22	23	88	8
Sample #23	14	21	20

z-index & stacking context

- **z-index controls stacking order for positioned elements (non-auto only works when element is positioned).**
- **New stacking contexts created by position + z-index, opacity < 1, transform, filter, isolation, etc.**
- **A child can never rise above its parent's stacking level**


Background/Borders
Negative Z-Index
Block Level Boxes
Floated Boxes
Inline Boxes
Z-Index = 0
Positive Z-Index
Stacking Order

Exercise 1: Move an element around using CSS positioning

1. Choose one element to move (example: Box, image etc.)
2. Apply inline CSS to change its position using one of these properties:
   - position
   - top, left, right, or bottom
3. The element must visibly move from its original place.

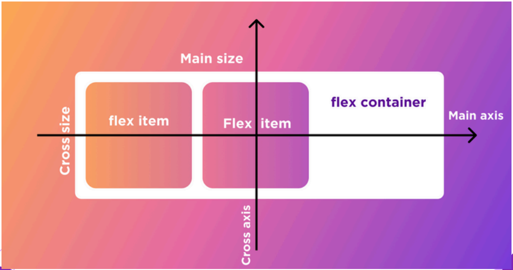
Bonus Challenge ⭐

Use z-index to move your chosen element on top of or behind another element.

📏Flexbox Basics (1D Layout)

- It provides a one-dimensional layout system, it aligns items along one direction, either a row or a column. It simplifies aligning items, spacing them evenly, and building responsive components.
- Flexbox automatically adjusts how items behave when screen sizes change, giving developers a simpler and more predictable way to layout elements.

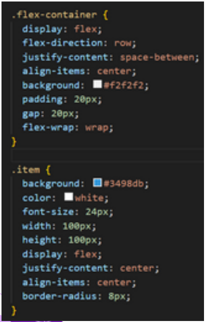
- Flexbox works by turning a container into a **flex container,** and the items inside it become **flex items**.
- Flexbox focuses on two axes: **the main axis** (the direction items flow) and **the cross axis** (perpendicular direction). By mastering these principles, one can produce layouts such as navigation bars, card grids, product lists, or centered content with minimal code.


Main size
flex container
flex item
Flex item
Main axis
Cross size
Cross axis

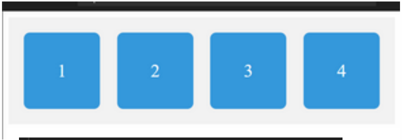
Flexbox Properties

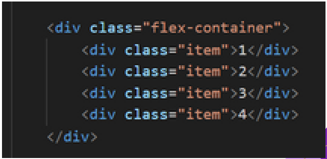
- **display: flex** - applied to container and enables Flexbox;
  - children become flex items.
- **flex-direction** - applied to container and sets main axis (row, row-reverse, column, column-reverse).
- **justify-content** - applied to container and aligns items along main axis (start, center, space-between, etc.).
- **align-items** - applied to container and aligns items along cross axis (start, center, stretch).
- **gap** - applied to container and sets spacing between items without margins.

- **flex-wrap** - applied to container and allows items to wrap onto new lines if needed.
- **align-content** - applied to container and controls spacing between rows when wrapping.
- **flex-grow** - applied to item and specifies how much a flex item expands to fill space.
- **flex-shrink** - applied to item and specifies how much an item shrinks when space is limited.
- **flex-basis** - applied to item and defines initial size before flexing happens.
- **order** - applied to item and reorders items without changing HTML.



```css
.flex-container {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  align-items: center;
  background: #f2f2f2;
  padding: 20px;
  gap: 20px;
  flex-wrap: wrap;
}

.item {
  background: #3498db;
  color: white;
  font-size: 24px;
  width: 100px;
  height: 100px;
  display: flex;
  justify-content: center;
  align-items: center;
  border-radius: 8px;
}
```





```html
<div class="flex-container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
</div>
```

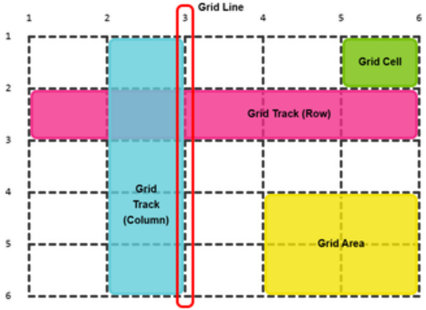
CSS Grid

- It is a **2D** layout system that allows you to design web pages using rows and columns simultaneously.
- It gives developers full control over both horizontal and vertical alignment, making it ideal for complex, grid-like layouts.
- Unlike Flexbox, Grid allows you to define track sizes, gaps, and areas that adapt naturally to different screen sizes.

🔍 Key CSS Grid Concepts

- **Grid Container** - Element with “display: grid”.
- **Grid Items** - Direct children of the grid container.
- **Grid Lines** - Invisible row/column dividing lines.
- **Grid Tracks** - The space between two lines (a row or a column).
- **Grid Cells** - A single rectangle in the grid.

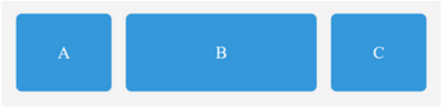
- **Grid Areas -** Named sections spanning multiple cells.
- **fr Unit -** Fractional space unit for flexible columns/rows.
- **gap -** Space between rows/columns.
- **auto-placement -** Browser automatically places items.

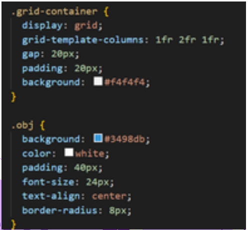

Grid Line
1
2
3
4
5
6
1
2
3
4
5
6
Grid Cell
Grid Track (Row)
Grid Track (Column)
Grid Area

Flexbox vs Grid
(The 1D vs 2D Concept)

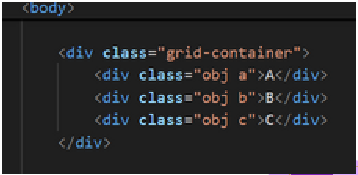
- **Flexbox**: a **one-dimensional** layout system; it controls layout in **one direction at a time**, either a row or a column. It excels at distributing space along a single axis and aligning items flexibly. Flexbox is best for navbars, buttons, cards in a row, or vertically aligning items inside a container.

- **CSS Grid:** a **two-dimensional** system that handles both rows AND columns **simultaneously**. You can place items anywhere in a grid, spanning multiple rows and columns.
- This makes Grid ideal for full-page layouts, forms, galleries, and dashboard-style designs where structure matters in more than one direction.





```css
.grid-container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  gap: 20px;
  padding: 20px;
  background: #f4f4f4;
}

.obj {
  background: #3498db;
  color: white;
  padding: 40px;
  font-size: 24px;
  text-align: center;
  border-radius: 8px;
}
```



```html
<body>

    <div class="grid-container">
        <div class="obj a">A</div>
        <div class="obj b">B</div>
        <div class="obj c">C</div>
    </div>
```

Exercise 2 - Flexbox: Align and Arrange Boxes

1. Add **display: flex** to the parent container.
2. Use at least two Flexbox alignment properties:

- justify-content
- align-items
- gap (optional)

Bonus Challenge ⭐

Use **flex-direction: column** to stack the items vertically on small screens.

Exercise 3 - CSS Grid: Create a 2×3 Grid Layout

1. Use **display: grid** on the container.
2. Define the grid using:
   - grid-template-columns
   - grid-template-rows
3. Use gap: to add spacing between cells.

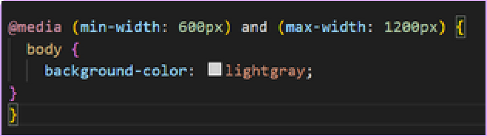
🌐 Responsive Tips & Techniques

Responsiveness in web design is an approach that makes web pages render well on a variety of devices and screen sizes by automatically adapting layouts, images, and functionality.

1. **Use Media Queries** (@media)

Media queries let you change your CSS at specific screen sizes.
They are the foundation of responsive design.

!Tip Alert



```
@media (min-width: 600px) and (max-width: 1200px) {
  body {
    background-color: lightgray;
}
}
```

They are the foundation of responsive design.

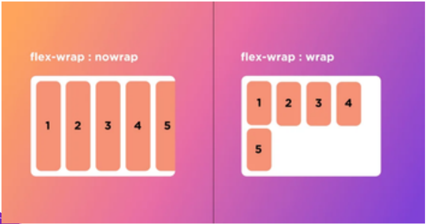
2. **Use Relative Units (%, em, rem, vh, vw)**

Avoid fixed pixels for layout and spacing.

- **%** → adjusts based on container width
- **em** → scales based on parent font size
- **rem** → scales based on root font size (recommended!)
- **vh/vw** → adjusts based on screen size

3. Flexbox Layout for Automatic Wrapping

Let items wrap instead of overflowing.



THRIVE
Women for Women

Q/A

Resources

THRIVE
Women for Women

- W3 School Css
- CSS Grid Tutorial - Video
- Flexbox Froggy - Playground

THRIVE
Women for Women

Thank You!!!