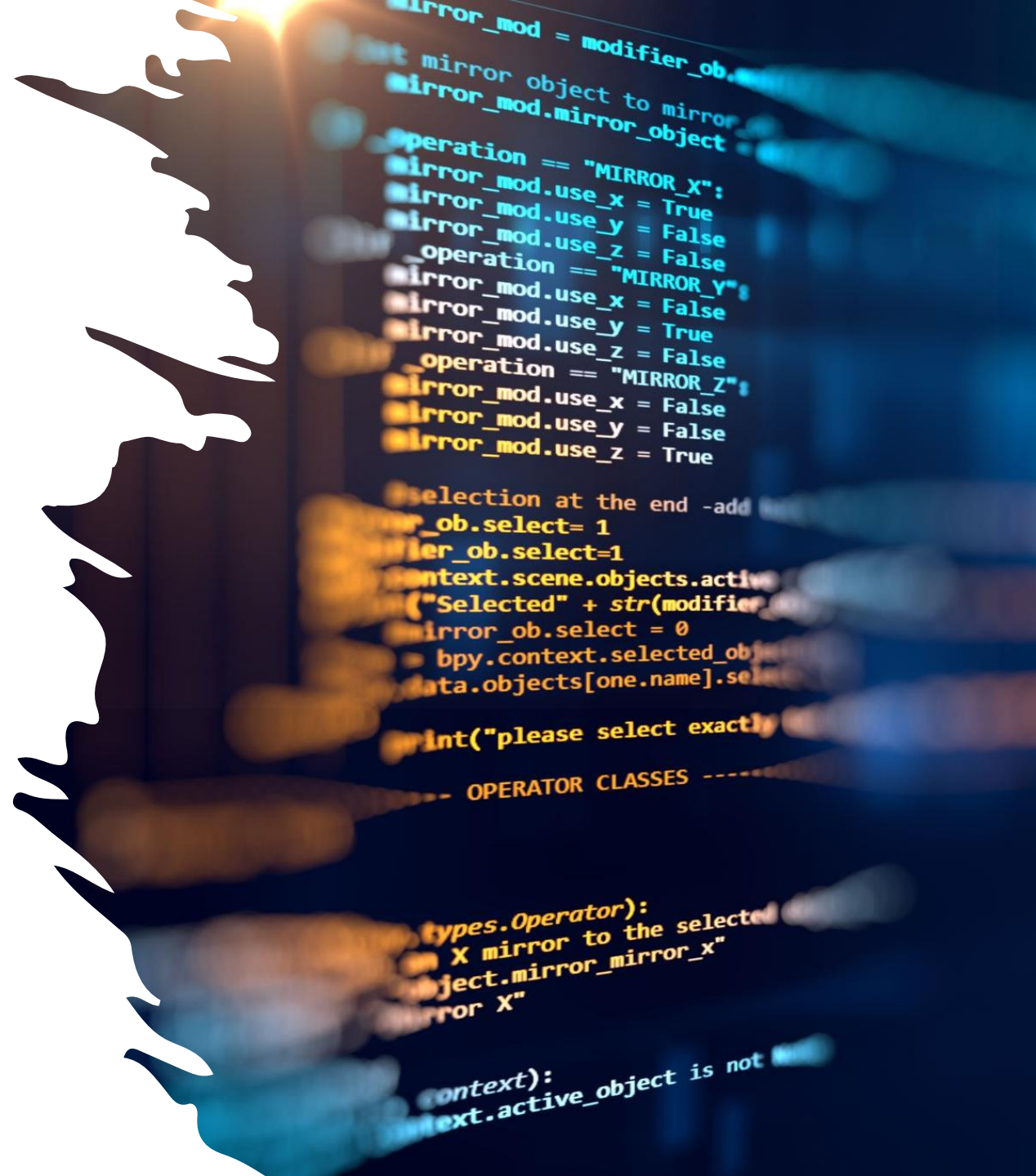


INTRODUCTION TO CSS

Frontend web development with
React

TOPICS

- What is CSS?
- Why CSS is important?
- How to include CSS in your web pages
- CSS Syntax
- Cascading and Specificity
- Selectors and Properties
- CSS Box Model
- CSS Layout
- Best Practices



WHAT IS CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in HTML.

- 1.CSS styles web content, controlling fonts, colors, layouts, and spacing to enhance visual appeal and user experience.
- 2.It separates content (usually in HTML) from its presentation, promoting maintainability by allowing design changes without affecting content.
- 3.CSS ensures a consistent look and feel across web pages by applying styles uniformly.
- 4.It enables responsive design, making websites adapt to different screen sizes and devices for broader accessibility.
- 5.CSS enhances efficiency by reusing styles, simplifying code, and reducing redundancy.
- 6.It contributes to accessibility by improving readability and usability for people with disabilities through proper styling.



WHY CSS IS IMPORTANT

CSS is crucial in web development because it **separates content** (usually written in HTML) from its **presentation**, allowing for easy styling and design changes without affecting the underlying content. This separation enhances website maintainability and fosters a consistent and visually appealing user experience across different web pages.

```
mirror_mod = modifier_ob.  
# Add mirror object to mirror  
mirror_mod.mirror_object  
operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True  
  
# Selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.name))  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
  
print("please select exactly  
-- OPERATOR CLASSES -----  
  
types.Operator):  
    on X mirror to the selected  
    object.mirror_mirror_x"  
    mirror X"  
  
context):  
context.active_object is not
```


INCLUDING CSS TO A WEB PAGE: INLINE CSS

This method involves adding CSS directly within the HTML elements using the "style" attribute. It's useful for applying styles to specific elements. For example:

```
<p style="color: blue; font-size: 16px;">This is a blue, 16px text.</p>
```

INLINE CSS

Internal CSS is placed within a `<style>` element in the HTML's `<head>` section. It's useful for applying styles to a single HTML document. For example:

```
<head>
```

```
<style> p { color: green; font-size: 18px;  
} </style>
```

```
</head>
```

```
<body>
```

```
<p>This is a green, 18px text.</p>
```

```
</body>
```

EXTERNAL CSS

External CSS is stored in a separate .css file and linked to the HTML document using the <link> element. This method is ideal for applying styles consistently across multiple pages. For example:

```
<head>
```

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

```
</head>
```

```
<body>
```

```
<p class="blue-text">This is a blue text.</p>
```

```
</body>
```

```
//In the styles.css:
```

```
.blue-text {
```

```
color: blue;
```

```
}
```

CASCADING IN CSS

Cascading in CSS refers to the process of determining which styles take precedence when multiple conflicting style rules target the same HTML element. The CSS cascade follows a specific order of importance:

Inline Styles: Styles applied directly to an HTML element using the "style" attribute take the highest precedence.

Internal (or Embedded) Styles: These are defined within the HTML document in the <style> element in the <head> section and apply to all elements in that document.

Styles defined in an external .css file linked to the HTML document using the <link> element. External styles apply to multiple pages and are the most common way to manage styles.



ALSO ON CSS

In CSS, the term "cascading" also implies that if a selector is styled multiple times, the styles defined in the last rule will take precedence. For example, if you have two rules targeting the same element, such as `p { color: yellow; }` and `p { color: blue; }`, the blue color will be applied to the element. Additionally, in cases like `p { color: green; color: violet; }`, the violet color will be the one applied, as it's the most recent style declaration for the same property.

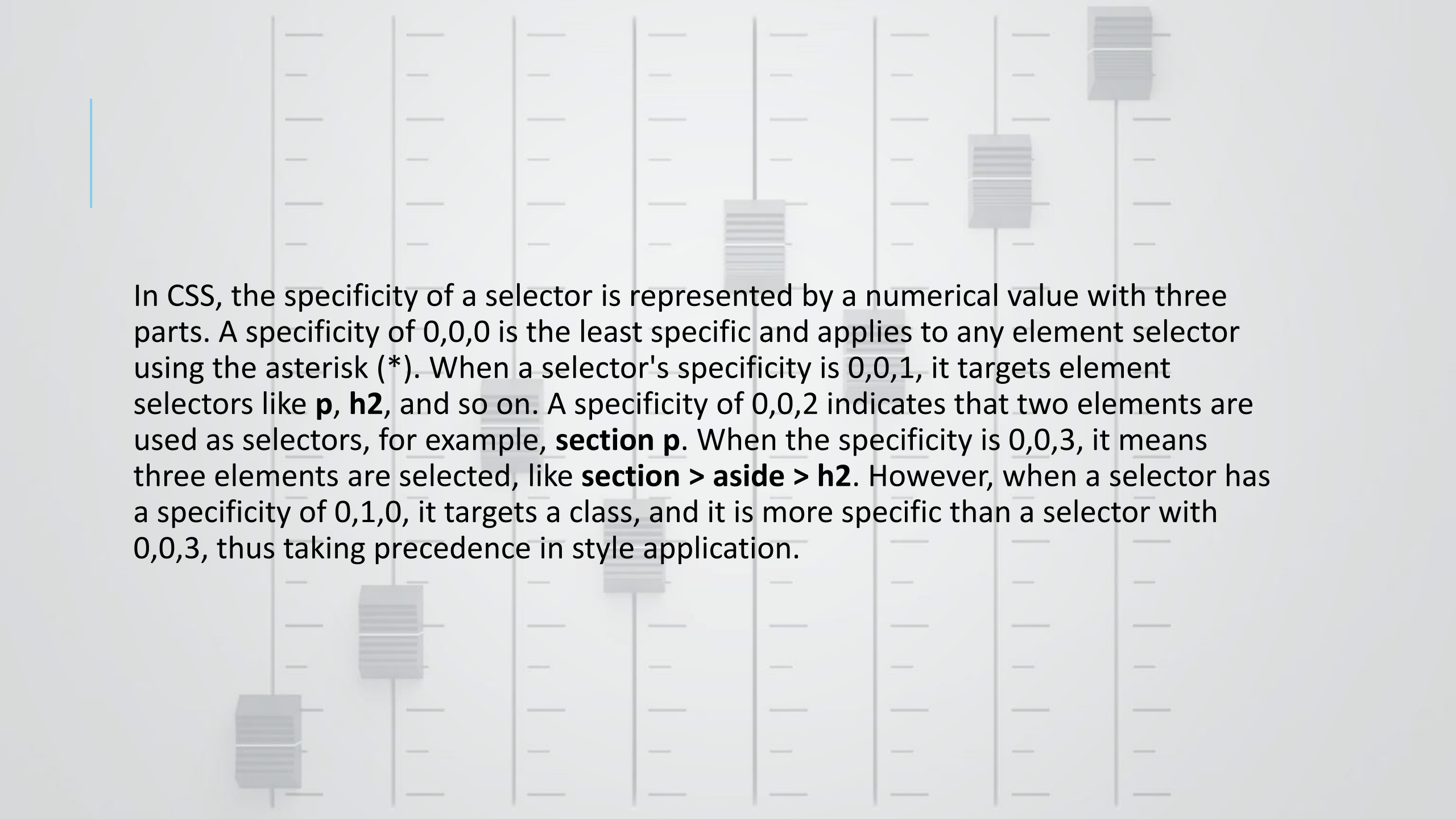
SPECIFICITY

If there are conflicting styles within the same type (e.g., two internal styles), specificity comes into play. Specificity is determined by the number and weight of selectors used in a style rule.

Specificity is represented using a four-part value, like 0, 0, 0, 0, where the higher values take precedence.

The four parts are for inline styles, IDs, classes/attributes/pseudo-classes, and elements.

The more specific a selector is, the higher its precedence.



In CSS, the specificity of a selector is represented by a numerical value with three parts. A specificity of 0,0,0 is the least specific and applies to any element selector using the asterisk (*). When a selector's specificity is 0,0,1, it targets element selectors like **p**, **h2**, and so on. A specificity of 0,0,2 indicates that two elements are used as selectors, for example, **section p**. When the specificity is 0,0,3, it means three elements are selected, like **section > aside > h2**. However, when a selector has a specificity of 0,1,0, it targets a class, and it is more specific than a selector with 0,0,3, thus taking precedence in style application.

SELECTORS



Selectors are used to target HTML elements to apply styles. There are various types of selectors in CSS:

Element Selector: It selects all HTML elements of a particular type. E.g. `p { color: blue; }`

ID Selector: It selects a specific HTML element with a unique ID attribute. E.g. `#unique-paragraph { font-size: 18px; }`

Class Selector: It selects HTML elements with a specific class attribute. E.g. `.highlighted { background-color: yellow; }`

Attribute Selector: It selects HTML elements based on their attributes. eg;. `a[href^="https"] { color: green; }`

Pseudo-class Selector: It selects elements based on their state or position. E.g. `a:hover { text-decoration: underline; }`

Pseudo-element Selector: It selects parts of an element, such as the first letter or line. `p::first-letter { font-size: 24px; }`

The background of the slide is a close-up photograph of numerous colorful 3D letter blocks. The blocks are in various colors including red, yellow, blue, and green, and are scattered across the frame. Some blocks are in sharp focus, while others are blurred in the background, creating a sense of depth. The lighting is soft, highlighting the smooth texture of the plastic blocks.

PROPERTIES IN CSS

color Property: This property is used to set the text color of an element. You can specify colors using a variety of methods, including named colors, hexadecimal color codes, RGB values, and HSL values.

Example:

```
.container { background-color: #00aaff; }
```

```
p {color: blue;}
```

```
section {color: rgb(122, 45, 87);}
```

color: Defines the text color.

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

```
p { color: red; }
```

```
.container { background-color: #00aaff; }
```

COLOR PROPERTIES:

FONT PROPERTIES:

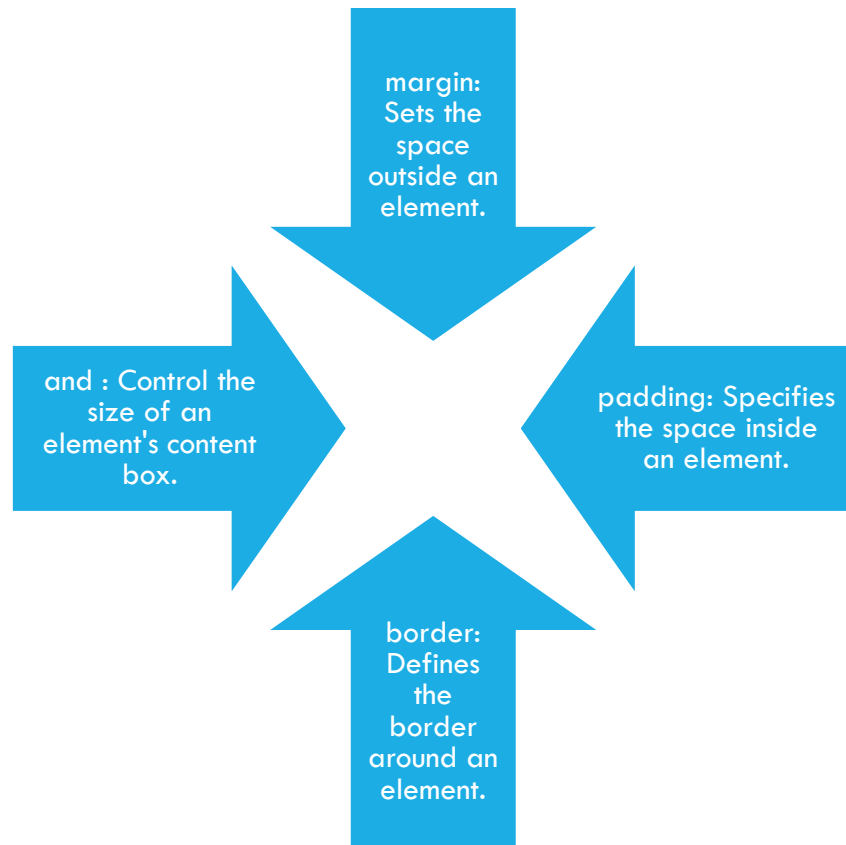
font-size:
Specifies
the size
of the
font.

Font-weight:
Defines the
weight of a font

font-family:
Defines the font
for text.

```
h1 { font-size: 24px; }  
body { font-family: Arial, sans-serif; }  
:first-child{font-weight: 700;}
```

BOX MODEL: WIDTH AND HEIGHT, BORDER, MARGIN AND PADDING:



```
img { border: 2px solid #333; }  
  
li { padding: 5px; }  
  
div { margin: 10px; }  
  
.container { width: 300px; height: 200px; }
```



Text Properties - text-align: Aligns text within an element.



text-decoration: Controls text decoration.



Display Properties - display: Determines how an element is displayed.



Positioning Properties - position: Defines the positioning method of an element.

```
p { text-align: center; }  
a { text-decoration: underline; }  
ul { display: inline; }  
.header { position: fixed; top: 0; }
```

TEXT DISPLAY, POSITION, AND DISPLAY PROPERTIES

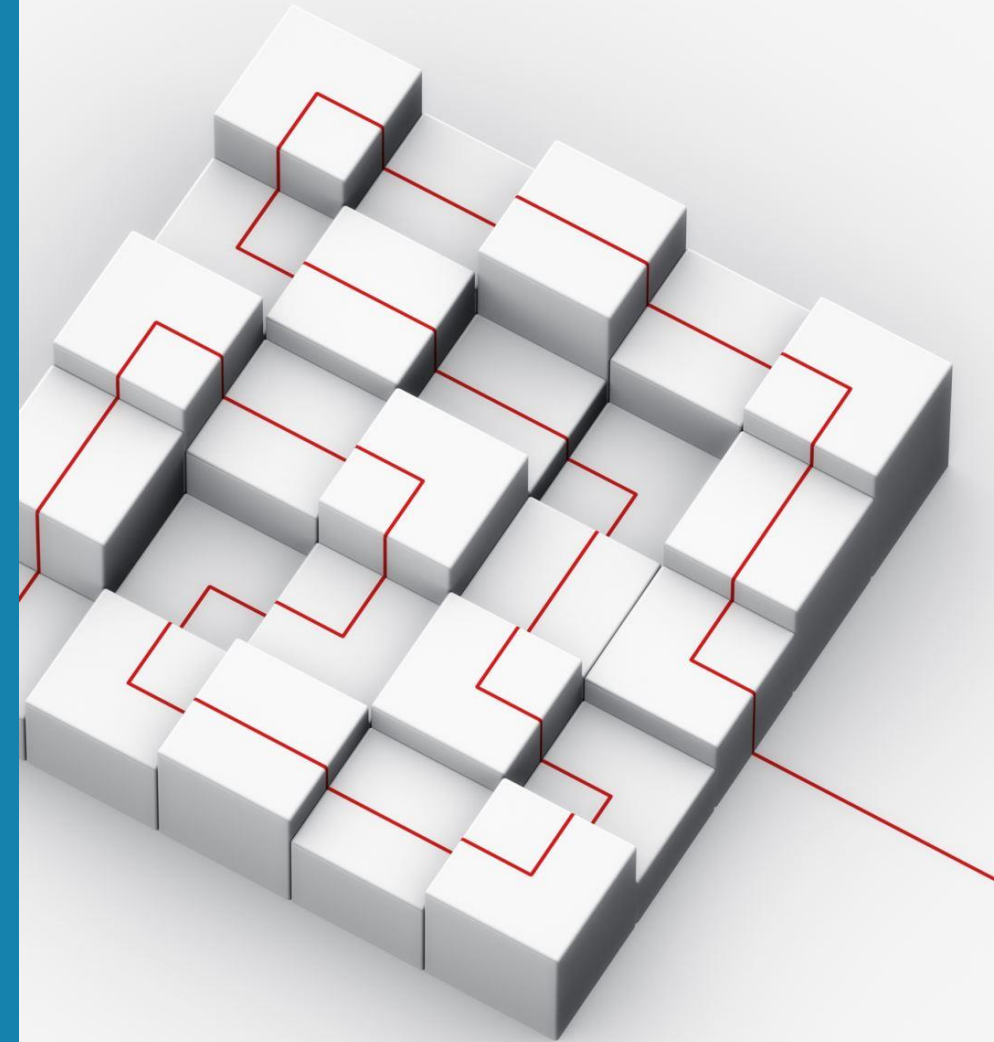
CSS LAYOUT

CSS layout refers to the arrangement and positioning of elements on a web page. It's a crucial aspect of web development as it defines how content is structured and displayed to users. As a frontend teacher, it's important to understand CSS layout principles to effectively teach web design and development. Here's an overview of CSS layout concepts.



BOX MODEL

- The box model is fundamental to CSS layout. Each HTML element is considered a rectangular box, comprising content, padding, border, and margin.
- Content: The actual content (text, images) within the element.
- Padding: The space between the content and the border.
- Border: The border around the element.
- Margin: The space between the element and other elements.

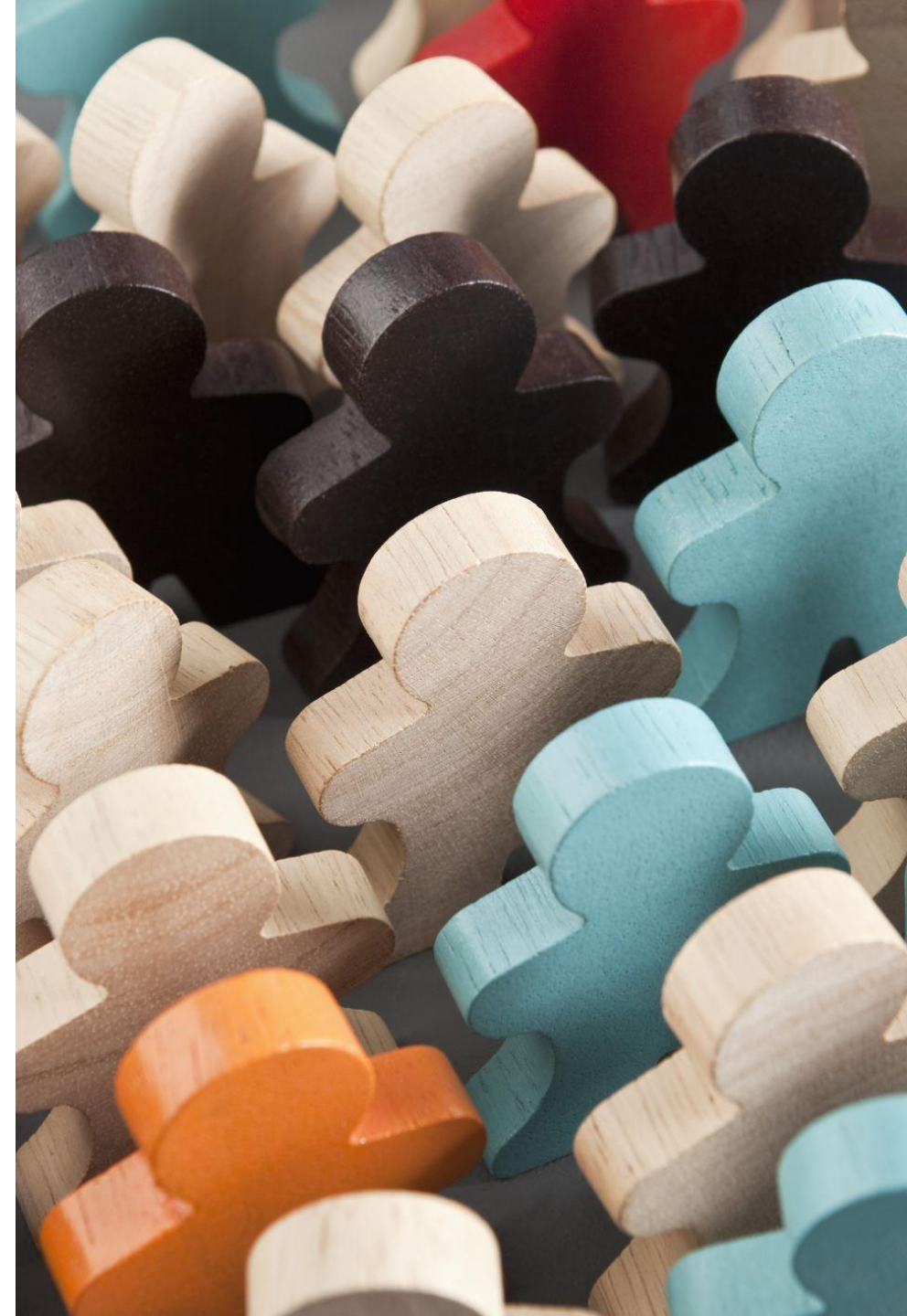


DISPLAY PROPERTIES

- The **display** property determines how an element is rendered in the layout. Common values include **block**, **inline**, **inline-block**, **flex**, and **grid**.
- **block** elements stack vertically, taking up the full width available.
- **inline** elements flow horizontally, taking up only as much width as necessary.

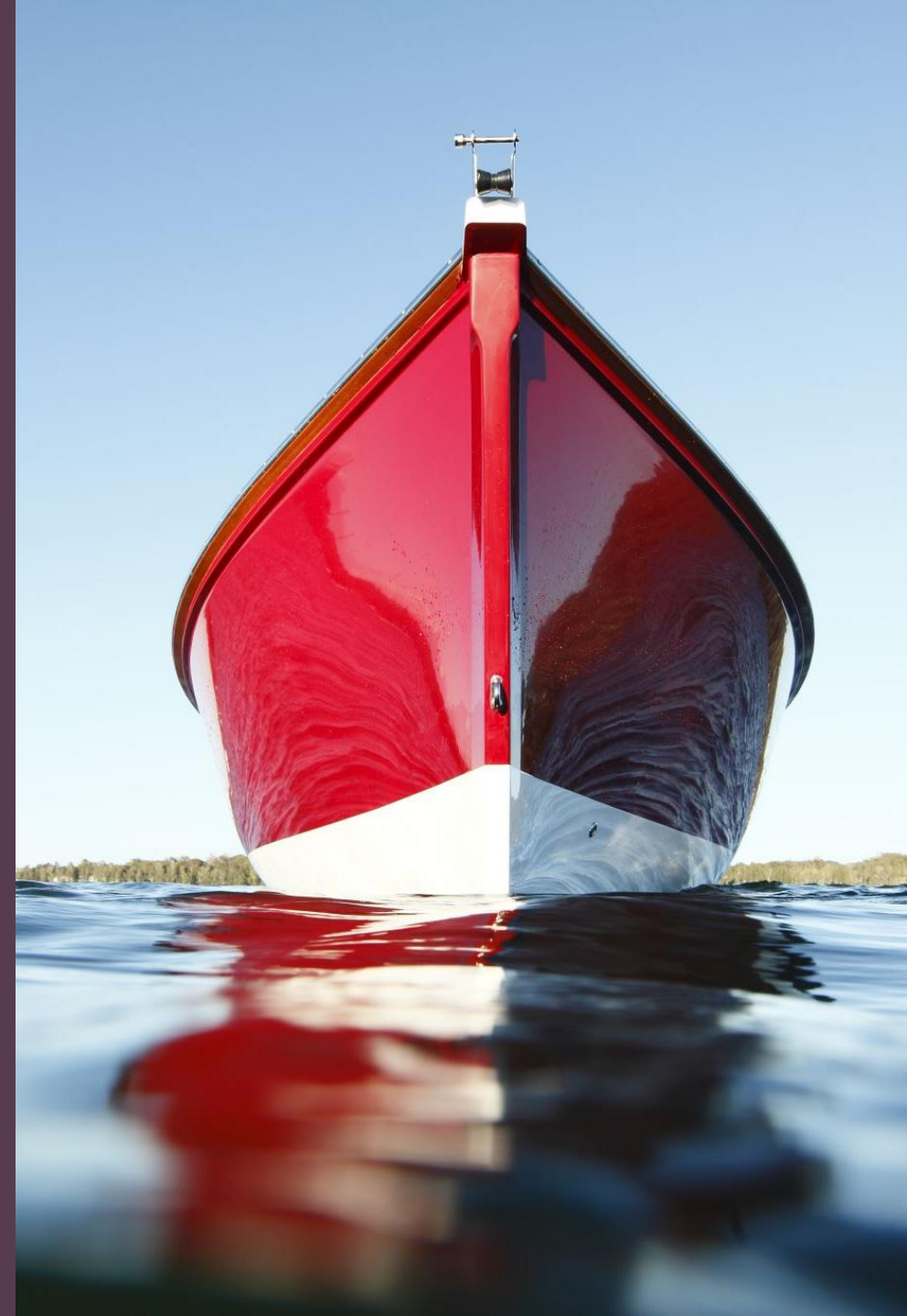
POSITIONING PROPERTIES

- The **position** property controls how elements are positioned within the document flow.
- Values include **static** (default), **relative**, **absolute**, and **fixed**.
- **relative** allows for adjustments relative to the element's normal position.
- **absolute** positions an element relative to its nearest positioned ancestor.
- **fixed** positions an element relative to the viewport.



FLOAT AND CLEAR

- The **float** property is used to move an element to the left or right within its container, allowing text or other elements to flow around it.
- The **clear** property ensures that elements do not wrap around a floated element.



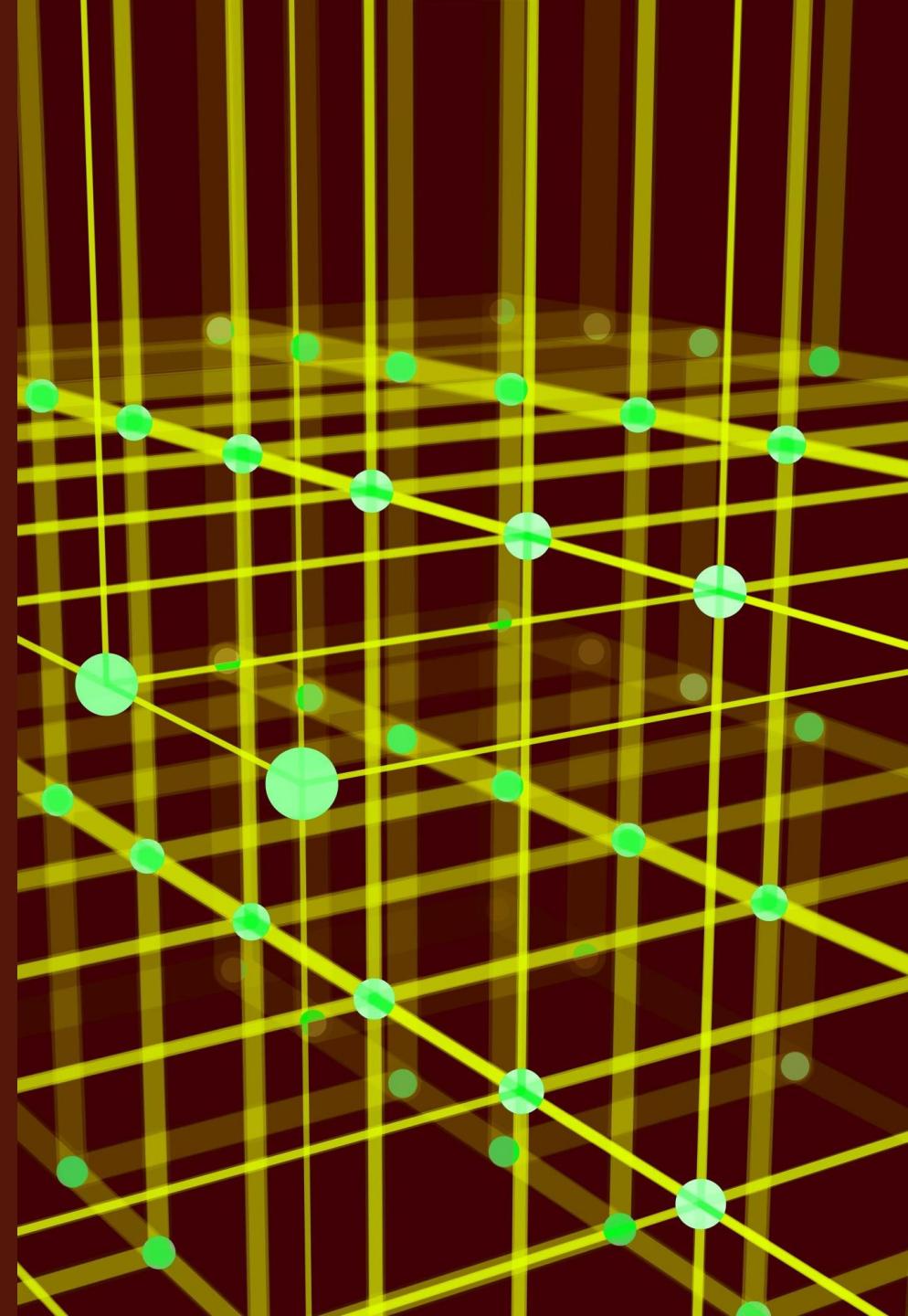
FLEXBOX

- Flexbox is a one-dimensional layout model that allows for the distribution of space within a container.
- It simplifies the alignment and distribution of items in a row or column, making it ideal for responsive design.



GRID LAYOUT

- CSS Grid Layout is a two-dimensional layout system that divides the layout into rows and columns.
- It's highly versatile and excels at creating complex, grid-based designs.



CENTERING AND POSITIONING

- CSS can be used to center elements both horizontally and vertically.
- Common methods include using **margin: auto**, **text-align: center**, **position**, and **transform**.



GRID PROPERTIES

display: grid: To create a grid layout, you set an element's display property to grid.

Example: `.container { display: grid; }`

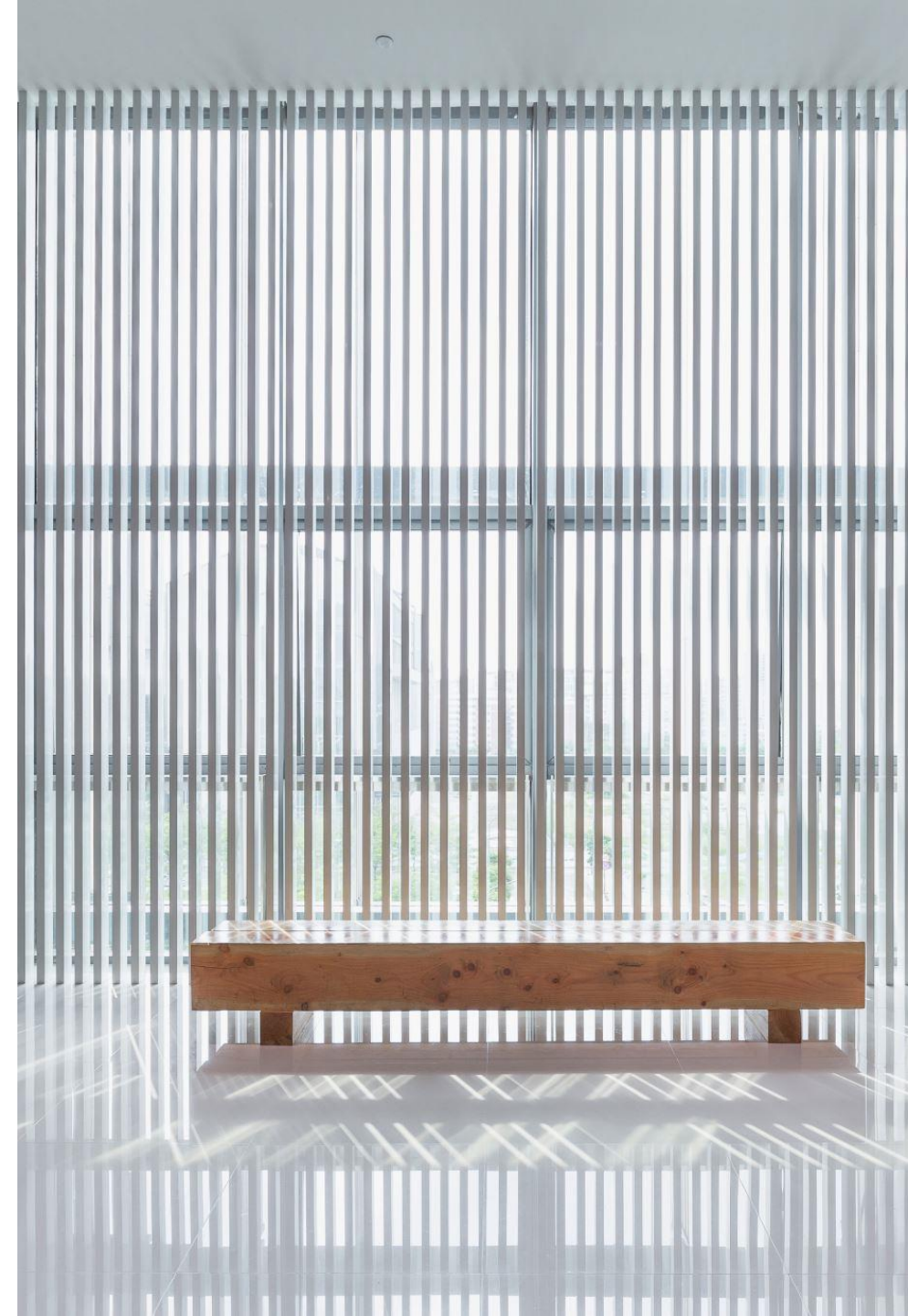
grid-template-columns and **grid-template-rows:** These properties define the size and structure of the grid's columns and rows, respectively.

You can specify sizes in various units (e.g., pixels, percentages, or "fr" units for flexible sizing).

Example: `.container { grid-template-columns: 1fr 2fr 1fr; /* Three columns with the middle column being twice as wide */ grid-template-rows: 100px 200px; /* Two rows with fixed heights */ }`

grid-gap: The grid-gap property sets the gap (space) between grid items in both rows and columns. It's a shorthand for grid-row-gap and grid-column-gap.

Example: `.container { grid-gap: 10px 20px; /* 10px gap between rows and 20px gap between columns */ }`



MORE GRID PROPERTIES

grid-template-areas: This property allows you to create named grid areas, making it easy to place items within the grid. You define the layout using a grid template, specifying named areas for each cell. Example:

```
.container { grid-template-areas: "header header" "nav main" "footer footer"; }
```

grid-column and **grid-row:** These properties specify the start and end positions of an item within the grid, based on the grid lines. You can use line numbers or named grid areas. Example:

```
.item { grid-column: 2 / 4; /* Start at column line 2 and end at column line 4 */ grid-row: 2 / 3; /* Start at row line 2 and end at row line 3 */ }
```

grid-area: This is a much better shorthand for grid-column and grid-row. Example:

```
.mix { grid-area: 2/1/3/4 /* row start at 2, column start at 1, row end at 3, column end at 4 */ }
```

MORE PROPERTIES

`justify-items` and `align-items`: These properties control the alignment of grid items within their grid cells. `justify-items` aligns items along the inline (horizontal) axis, and `align-items` aligns items along the block (vertical) axis.

Example: `.item { justify-items: center; /* Center horizontally within the cell */ align-items: start; /* Align to the top of the cell */ }`

`justify-content` and `align-content`: These properties control the alignment of the entire grid within its container. They work similarly to `justify-items` and `align-items`, but for the grid as a whole.

Example: `.container { justify-content: center; /* Center the grid horizontally within its container */ align-content: space-between; /* Distribute items evenly along the vertical axis */ }`

FLEX PROPERTIES

display: flex: To create a Flexbox container, you set the display property of an element to flex.

Example: `.container { display: flex; }`

flex-direction: This property specifies the direction in which the flex items are arranged within the flex container. It can be set to row, row-reverse, column, or column-reverse.

Example: `.container { flex-direction: row-reverse; /* Flex items are arranged in a row in reverse order */ }`

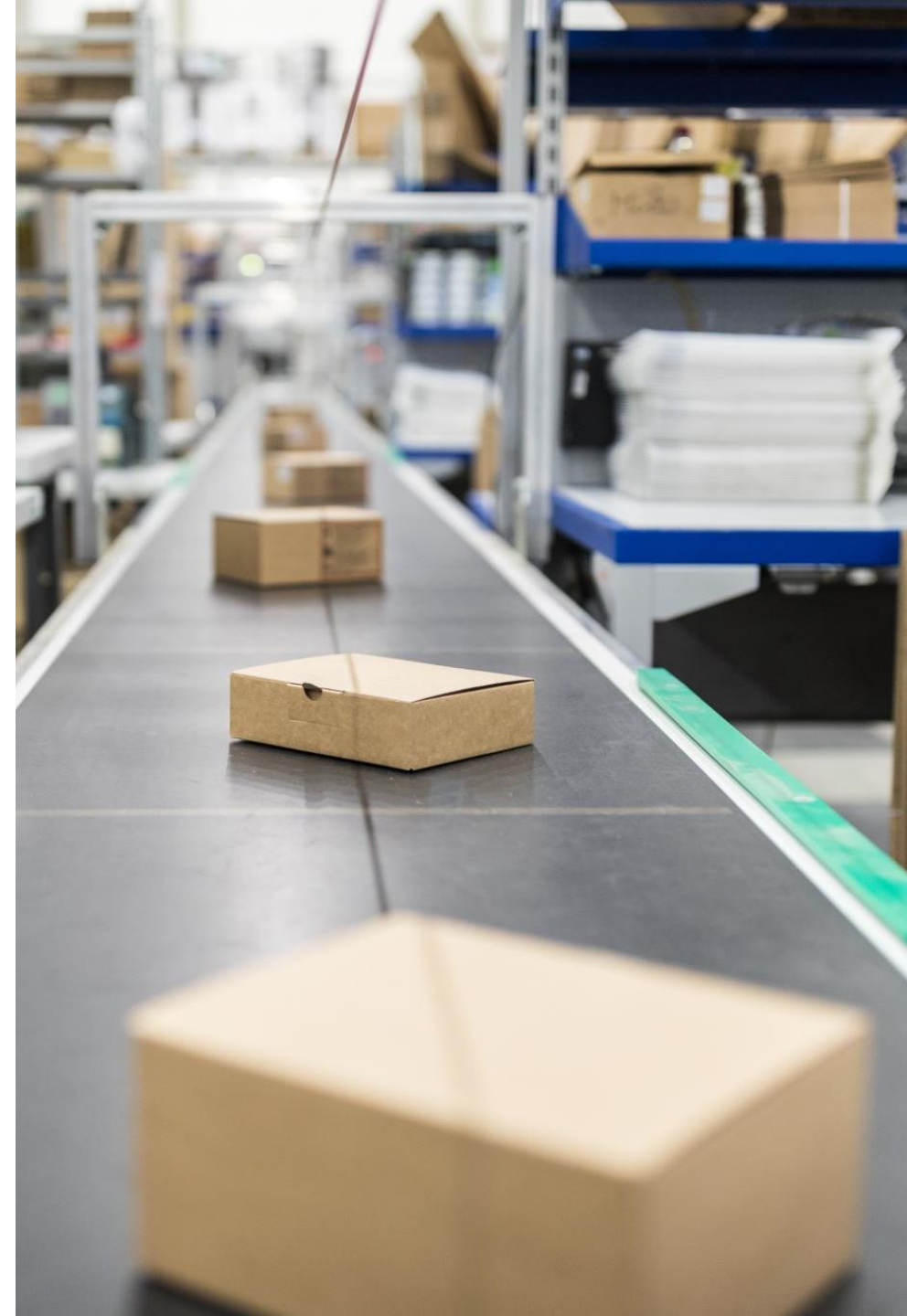
flex-wrap: The flex-wrap property controls whether flex items should wrap to the next line if they don't fit in the container. Values include nowrap, wrap, and wrap-reverse.

Example: `.container { flex-wrap: wrap; /* Flex items wrap to the next line if necessary */ }`

justify-content: This property determines how extra space in the flex container is distributed along the main axis. It's used for aligning flex items horizontally.

Values include flex-start, flex-end, center, space-between, and space-around.

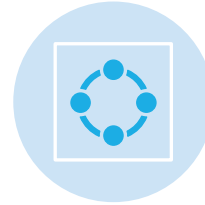
Example: `.container { justify-content: space-between; /* Flex items are evenly spaced along the main axis */ }`



MORE FLEX PROPERTIES



align-items: align-items controls how flex items are aligned within the flex container along the cross axis. Values include flex-start, flex-end, center, baseline, and stretch.



Example: `.container { align-items: center; /* Flex items are centered along the cross axis */ }`



align-content: Similar to justify-content, align-content is used to control how extra space in the flex container is distributed along the cross axis.



Example: `.container { align-content: space-between; /* Flex items are evenly spaced along the cross axis */ }`

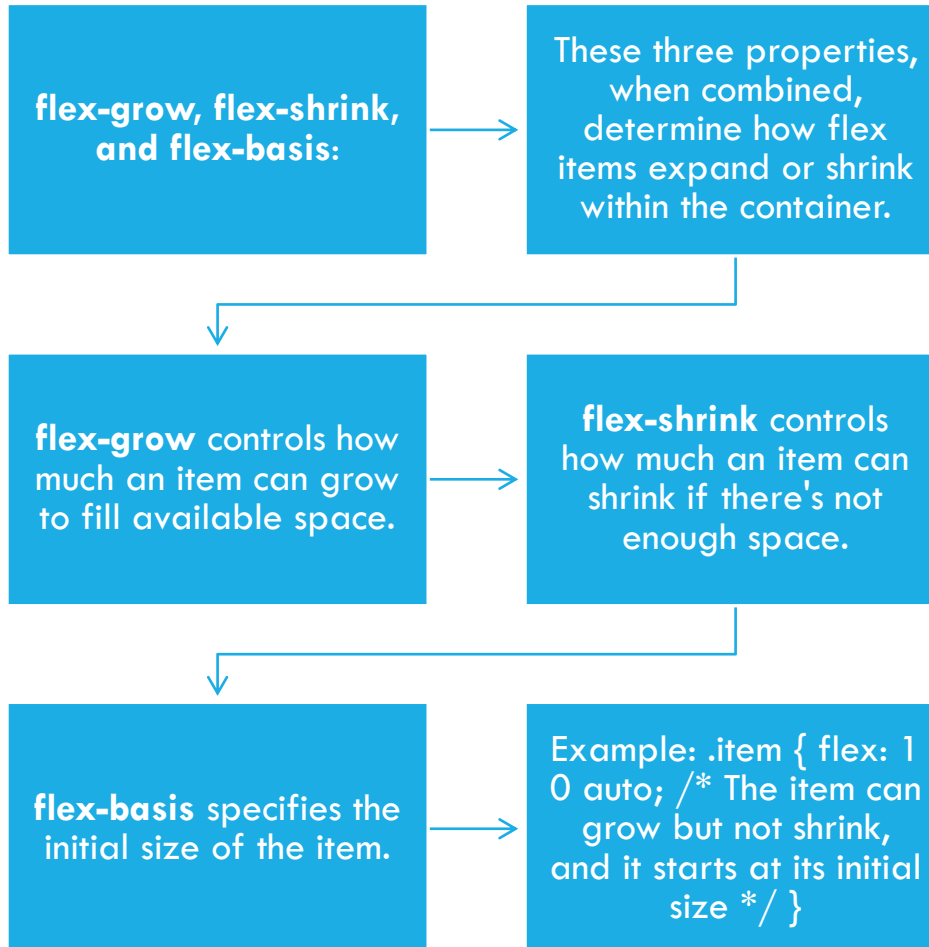


order: The order property allows you to specify the order in which flex items are displayed, overriding their default order in the HTML structure. Elements with a lower order value appear first.



Example: `.item { order: 2; /* This item will be displayed after items with lower order values */ }`

MORE FLEX PROPERTIES



OTHER NOTABLE LAYOUT PROPERTIES

Float and Clear: The float property is used to move an element to the left or right within its container, allowing text or other elements to flow around it.

The clear property ensures that elements do not wrap around a floated element.

Box Sizing: The box-sizing property controls how the total width and height of an element is calculated.

border-box includes padding and border in the element's total width/height, which can simplify layout calculations.

Overflow Property: The overflow property determines how content that overflows its container is handled, whether it's clipped, hidden, or displayed with scrollbars.



COMING NEXT...

CSS MEDIA QUERY