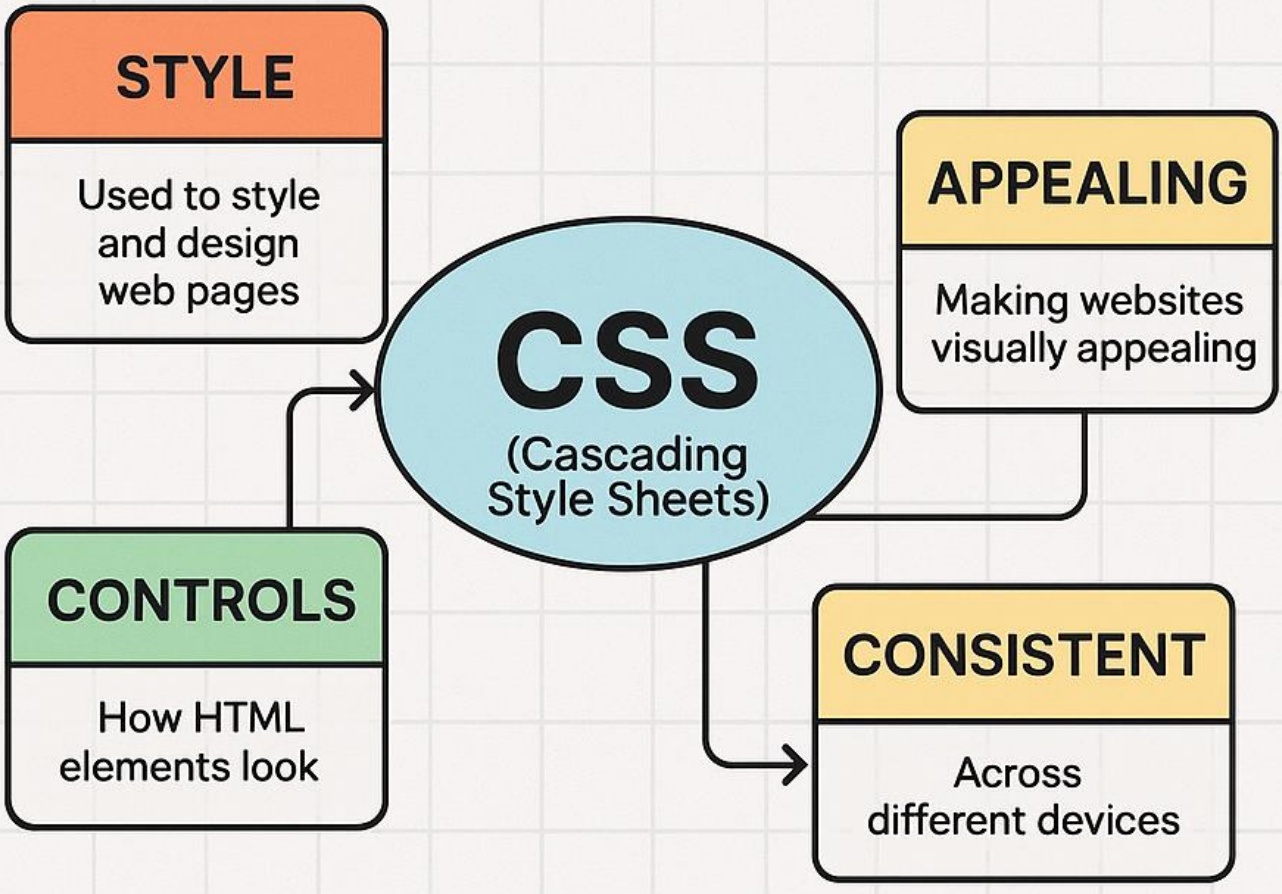
CSS CONCECT


STYLE
Used to style and design web pages
APPEALING
Making websites visually appealing
CSS
(Cascading Style Sheets)
CONTROLS
How HTML elements look
CONSISTENT
Across different devices

HTML + CSS

How They Work Together

HTML

<HL>Heading <h1>

<p Paragraph/p>

CSS

STYLE

BROWSER

p
Paragraph
]
Colx: blue;
font-size: 16px;
}

STRUCTURE

HTML provides the structure and content of a webpage

LINKING

A CSS file is link to an HTML file using the <1ink> élément

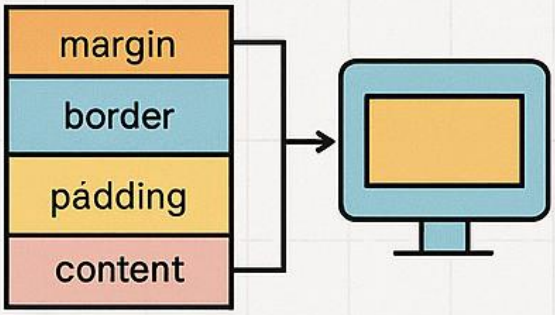
SEMANTIC TAGS

HTML uses sémantic tags like <article> and <footer>

INLLINE / INTERNAL EXTERNAL

CSS can be Included directly in an HTML element (inline) in a <sty1e> block (internal). or in a separate file (éxternal)

BOX MODEL



RESPONSIVEL DESIGN

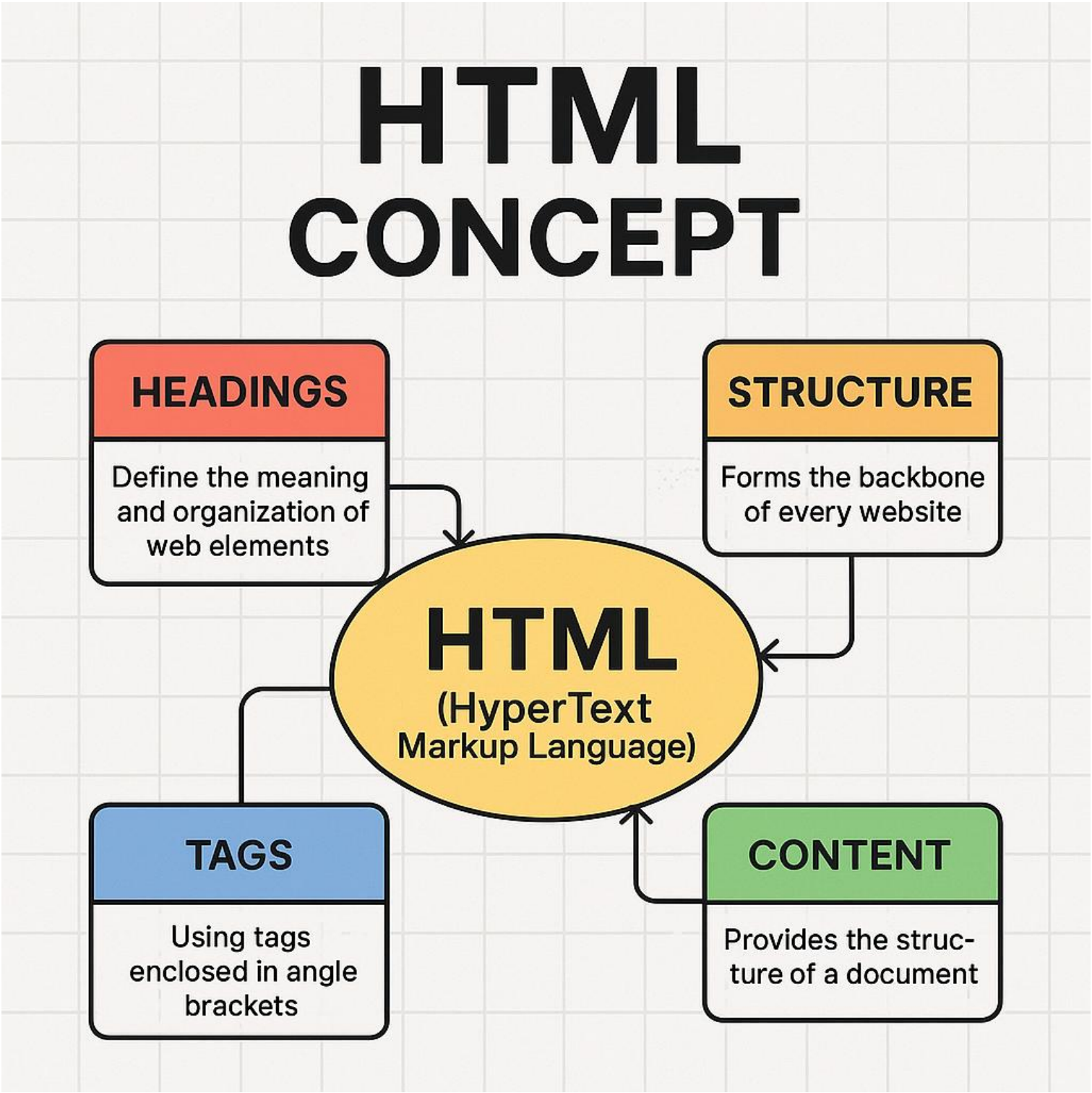
Média query CSS rules adjust the layout based on the device

SEMANTIC TAGS

HTML-uses sémantic tags like <article> and <footer>

REUSAJBILITY & CONSISTENCY

CSS enables reusability any consistency of stylés


HTML
CONCEPT
HEADINGS
Define the meaning and organization of web elements
STRUCTURE
Forms the backbone of every website
HTML
(HyperText Markup Language)
TAGS
Using tags enclosed in angle brackets
CONTENT
Provides the structure of a document

CSS Flexbox Layout

Master One-Dimensional Layouts with Flexibility

↓

1. HTML File: flex.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="flexstyle.css">
    <title>Document</title>
</head>
<body>
    <section class="layout">
        <div>1</div>
        <div>2</div>
        <div>3</div>
        <div>4</div>
    </section>
    <hr>
    <section class="layout">
```

```
        <div>1</div>
        <div>2</div>
        <div>3</div>
        <div>4</div>
    </section>
</body>
</html>
```

Structure Breakdown:

- The HTML page includes a CSS file (`flexstyle.css`) for styling
- Contains **two** `<section>` elements, each with class `.layout`
- Inside each section are four `<div>` boxes labeled 1, 2, 3, and 4
- The `<hr>` tag adds a horizontal line separating the sections

Result: Two flexbox layouts stacked vertically — one above the line and one below.

2. CSS File: flexstyle.css

Parent Container: .layout

```
.layout {
    width: 100%;
    display: flex;
    gap: 16px;
    justify-content: space-around;
}
```

The `.layout` class controls how child boxes are arranged horizontally.

display: flex;

Turns each section into a **flex container**, enabling flexbox layout for its children.

justify-content: space-around;

Spaces out the child boxes evenly with equal space before, between, and after them.

gap: 16px;

Adds a 16-pixel gap between boxes (simpler and cleaner than using margins).

width: 100%;

Makes each layout fill the available width of the page.

Child Elements: .layout div

```
.layout div {
    width: 200px;
    height: 200px;
    background-color: lightblue;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 20px;
    font-weight: bold;
    color: white;
    border-radius: 8px;
    border: 2px solid navy;
}
```

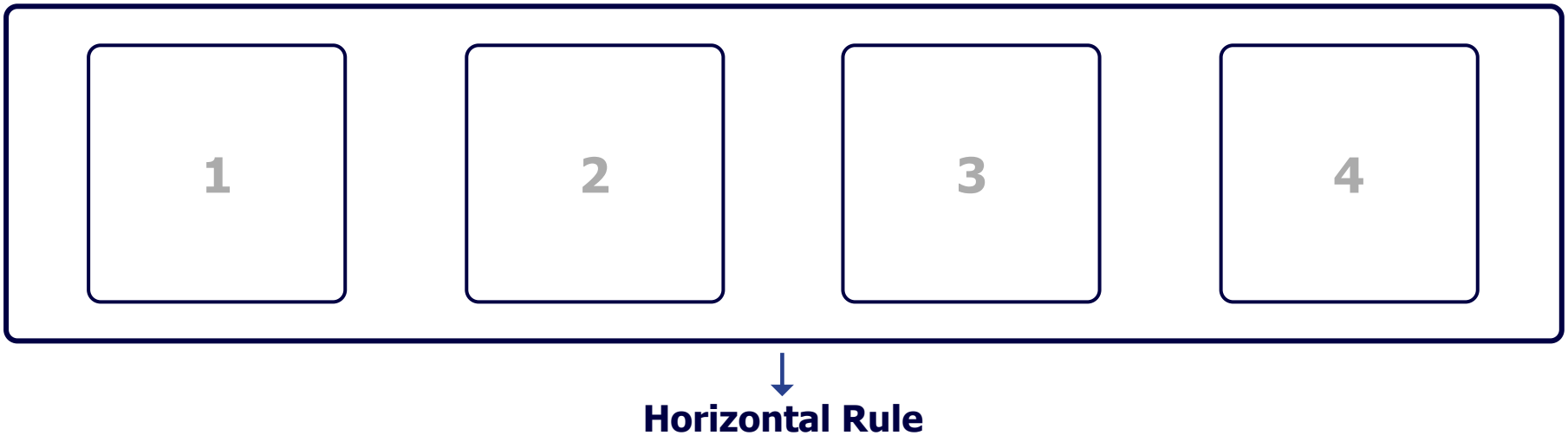
Each box inside the `.layout` has consistent styling:

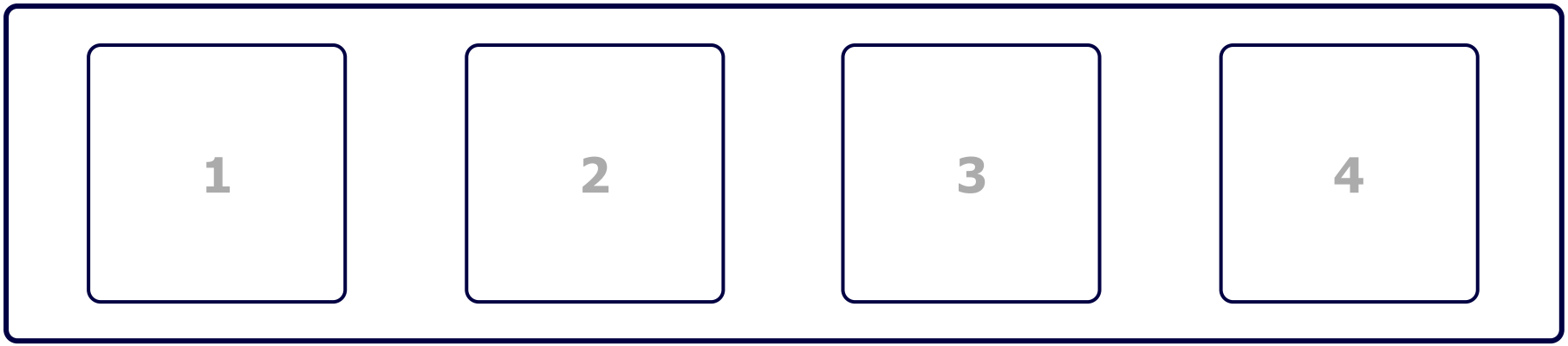
- **Fixed size:** `200px × 200px`
- **Self-centering:** Uses `display: flex` with `align-items: center` and `justify-content: center` to center its content
- **Styling:** Light-blue background, rounded corners, navy border

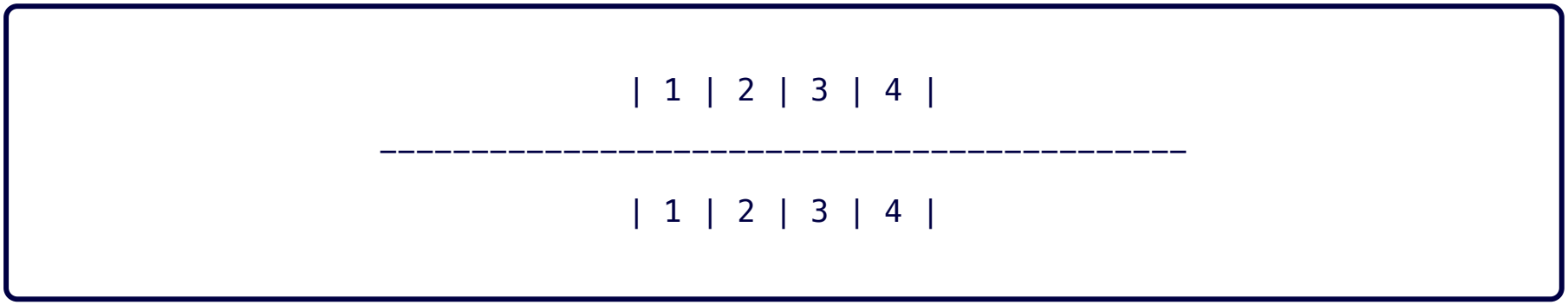
- **Text:** Numbers 1–4 appear white, bold, and centered inside each box

Final Visual Layout

Each section looks like this:







Each box is:

- Evenly spaced across the container
- Perfectly centered with consistent sizing
- Styled with borders and rounded corners

🔑

Understanding justify-content

The `justify-content: space-around;` property creates equal spacing. Here's how different values compare:

Value	Effect
`flex-start`	Items align to the start of the container
`flex-end`	Items align to the end of the container
`center`	Items are centered in the container
`space-between`	First item at start, last at end, equal space between
`space-around`	Equal space around each item (used in your code)

Value	Effect
`space-evenly`	Equal space between items and edges

Concept Summary

Concept	Description
Flexbox	Used to arrange elements horizontally (or vertically) with easy alignment and spacing
Parent .layout	Defines how child boxes are laid out (alignment, spacing)
Child elements	Styled boxes (content centered, fixed size, colored)

Concept	Description
Reusability	Because .layout is a class, the same style can be reused for multiple sections

Key Takeaways

🎯 Why Use Flexbox?

- **Simple alignment:** Center elements horizontally and vertically with ease
- **Flexible spacing:** Control space between items without complex calculations
- **Responsive by default:** Items adjust to container size automatically
- **One-dimensional:** Perfect for rows or columns (use Grid for 2D layouts)

🔧 Common Use Cases:

- Navigation bars with evenly spaced links
- Card layouts in a row
- Button groups
- Form layouts with aligned inputs
- Image galleries

🎉 You Now Understand CSS Flexbox!

Start building flexible, responsive layouts with confidence.

↓

CSS Grid Layout

A Complete Guide to Modern Web Layouts

↓

1. HTML Structure

Your HTML defines a simple page layout with five main semantic sections:

```
<header>Header</header>
<nav>Navigation</nav>
<main>Main Content</main>
<aside>Sidebar</aside>
<footer>Footer</footer>
```

Semantic HTML5 Tags: Each element describes the role of its content:

- **`<header>`** - Page headings and introductory content
- **`<nav>`** - Navigation links and menus
- **`<main>`** - Primary content of the page

- `<aside>` - Sidebar or secondary content
- `<footer>` - Footer information and links

All elements are direct children of the `<body>`, which becomes our grid container.

2. CSS Grid Layout

🔷 The Body as a Grid Container

```
body {
    display: grid;
    grid-template-areas:
        "header header"
        "nav nav"
```

```
        "main sidebar"
        "footer footer";
    grid-template-columns: 2fr 1fr;
    grid-template-rows: auto;
    gap: 10px;
    text-align: center;
}
```

This transforms the `<body>` into a grid container, dividing the page into rows and columns.

Key CSS Grid Properties

`display: grid;`

Activates the CSS Grid layout system for the element.

`grid-template-areas`

Defines a layout map using text labels. Each line represents a row:

```
"header header"   → Row 1: header spans 2 columns
"nav nav"         → Row 2: nav spans 2 columns
"main sidebar"    → Row 3: main (left), sidebar (right)
"footer footer"   → Row 4: footer spans 2 columns
```

`grid-template-columns: 2fr 1fr;`

Defines 2 columns:

- First column (main) gets **2 parts** of available width
- Second column (sidebar) gets **1 part**
- Result: Main area is twice as wide as the sidebar

`gap: 10px;`

Adds spacing between grid items.

🔷 Assigning Grid Areas

Each HTML element is assigned to one of the grid areas:

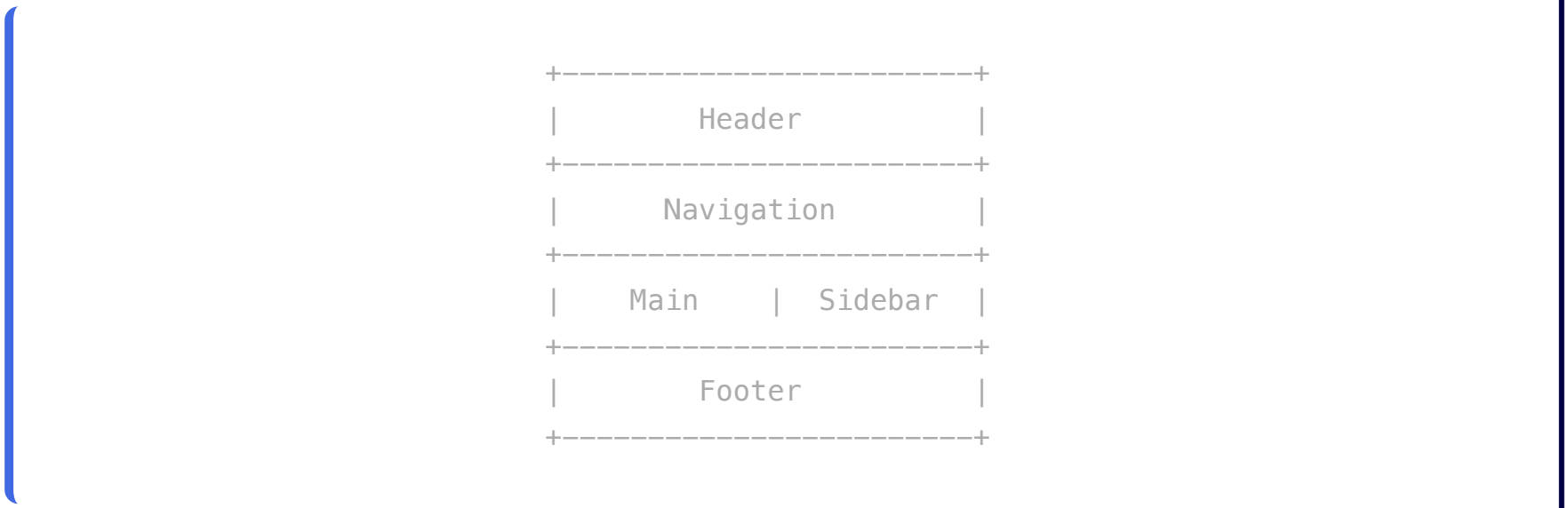
```
header { grid-area: header; background: lightblue; padding: 20px; }
nav { grid-area: nav; background: lightgray; padding: 15px; }
main { grid-area: main; background: lightgreen; padding: 20px; }
aside { grid-area: sidebar; background: lightcoral; padding: 20px; }
footer { grid-area: footer; background: lightgoldenrodyellow; padding: 15px; }
```

Visual Result:

Navigation

Main Content **Sidebar**

Footer



3. Responsive Design

Media queries adapt the layout for different screen sizes:

```
@media (max-width: 768px) {
    body {
        grid-template-areas:
            "header"
            "nav"
            "main"
            "sidebar"
            "footer";
        grid-template-columns: 1fr;
    }
}
```

When screen width ≤ 768px (mobile devices):

- Grid changes to a **single-column layout**
- Each section stacks **vertically**
- Ensures the layout works on tablets and phones

Try resizing your browser window to see the responsive layout in action! 🔄

4. How It Works Together

1. The browser loads `grid.html`

2. It links to `gridstyle.css`, which applies the grid layout

3. The `<body>` acts as a grid container, dividing space for each section

4. The `grid-template-areas` visually arranges the page elements

5. On smaller screens, the media query rearranges the layout vertically

Summary of Concepts

Concept	Description
CSS Grid	A powerful 2D layout system allowing precise control over rows and columns
Grid Template Areas	Text-based layout mapping for intuitive positioning

Concept	Description
fr unit	Fractional unit that divides space proportionally
Responsive Media Query	Adjusts layout based on device width
Semantic HTML	Improves readability, accessibility, and SEO

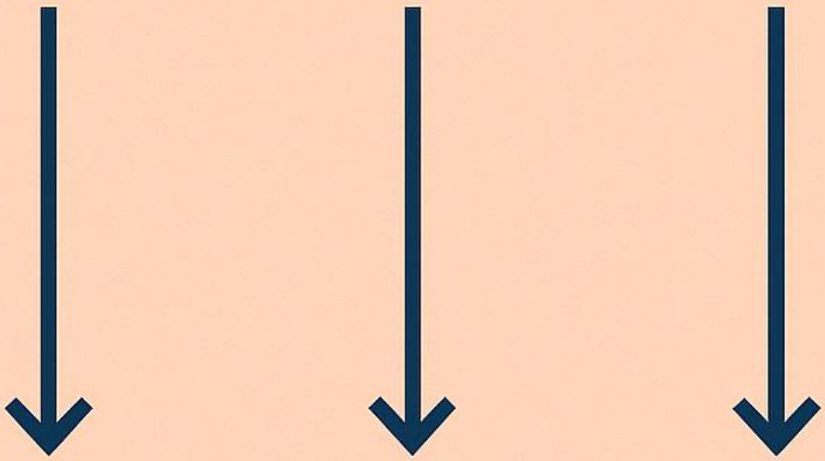
🎉 You Now Understand CSS Grid!

Start building beautiful, responsive layouts with confidence.

CSS SELECTOR

A CSS selector is used to target and select HTML elements.

```
selector {value;}
```



SELECTOR

Selects the HTML element to style

PROPERTY

Defines the style effect to apply

VALUE

Specifies the value of the property

📱 Understanding CSS Media Queries

> **Welcome to this interactive lesson!** Media queries are one of the most powerful features in CSS, allowing you to create responsive websites that adapt to different screen sizes, devices, and user preferences. By the end of this lesson, you'll understand how to use media queries to build modern, responsive web designs.

What Are Media Queries?

Media queries are a CSS technique that allows you to apply different styles depending on the characteristics of the device or viewport displaying your content. Think of them as conditional statements for your CSS—"if the screen is this size, then apply these styles."

> **Key Concept:** Media queries enable responsive design by allowing your website to adapt its layout and appearance based on factors like screen width, height, orientation, and resolution.

Why Are Media Queries Important?

In today's world, people access websites from a huge variety of devices:

- ✓ Smartphones with small touchscreens
- ✓ Tablets with medium-sized displays
- ✓ Laptops with various screen sizes
- ✓ Large desktop monitors
- ✓ Smart TVs and other devices

Media queries allow you to create a single website that looks great on all these devices, rather than building separate versions for each one.

Basic Syntax

> **The basic syntax of a media query:**

```
@media media-type and (condition) {
    /* CSS rules go here */
    selector {
        property: value;
    }
}
```

Breaking Down the Syntax:

@media - The keyword that starts a media query

media-type - Specifies the type of media (screen, print, speech, all)

and - Logical operator to combine multiple conditions

(condition) - The rule that must be true for the styles to apply

Common Media Query Examples

1. Max-Width (Mobile-First Approach)

This applies styles when the viewport is **up to** a certain width:

```
/* Styles for mobile devices (up to 600px) */
@media screen and (max-width: 600px) {
    body {
        font-size: 14px;
    }

    .container {
        padding: 10px;
    }
}
```

2. Min-Width (Desktop-First Approach)

This applies styles when the viewport is **at least** a certain width:

```
/* Styles for tablets and larger (601px and up) */
@media screen and (min-width: 601px) {
    .container {
        max-width: 900px;
        margin: 0 auto;
    }
}
```

3. Range Queries

This applies styles within a specific range:

```
/* Styles specifically for tablets (601px to 900px) */
@media screen and (min-width: 601px) and (max-width: 900px) {
    .sidebar {
        width: 30%;
    }
}
```

Live Demo: Resize Your Browser!

The box below changes color based on your screen size. Try resizing your browser window to see it in action!

RESIZE YOUR BROWSER WINDOW →

Responsive Grid Demo

This grid adjusts from 4 columns (desktop) to 2 columns (tablet) to 1 column (mobile):

Item 1 Item 2 Item 3 Item 4

Other Media Query Features

Orientation

You can target specific device orientations:

```
/* Landscape orientation */
@media screen and (orientation: landscape) {
```

```
    .header {
        height: 60px;
    }
}

/* Portrait orientation */
@media screen and (orientation: portrait) {
    .header {
        height: 80px;
    }
}
```

ROTATE YOUR DEVICE →

Resolution and Pixel Density

Target high-resolution displays (like Retina screens):

```
/* For high-DPI screens */
@media screen and (min-resolution: 2dppx) {
    .logo {
        background-image: url('logo@2x.png');
    }
}
```

Print Styles

Optimize your page for printing:

```
/* Print-specific styles */
@media print {
    .no-print {
        display: none;
    }

    body {
        color: black;
        background: white;
    }
}
```

Common Breakpoints

While you can set breakpoints at any width, here are some commonly used values:

```
/* Mobile (phones) */
@media screen and (max-width: 600px) { }

/* Tablet (portrait) */
@media screen and (min-width: 601px) and (max-width: 900px) { }

/* Desktop (small laptops) */
@media screen and (min-width: 901px) and (max-width: 1200px) { }
```

```
/* Desktop (large screens) */
@media screen and (min-width: 1201px) { }
```

Best Practices

- ✓ **Mobile-First:** Start with mobile styles, then use min-width queries to add complexity for larger screens
- ✓ **Use Relative Units:** Use em or rem instead of px for more flexible breakpoints
- ✓ **Test on Real Devices:** Browser resizing doesn't capture everything—test on actual phones and tablets
- ✓ **Don't Overdo It:** Too many breakpoints can make your CSS hard to maintain
- ✓ **Content-Based Breakpoints:** Set breakpoints where your content needs them, not just at device sizes
- ✓ **Use Developer Tools:** Modern browsers have responsive design modes to test different screen sizes

Logical Operators

You can combine multiple conditions using logical operators:

AND Operator

```
/* Both conditions must be true */
@media screen and (min-width: 600px) and (max-width: 900px) {
    /* Styles for tablets */
}
```

OR Operator (Comma)

```
/* Either condition can be true */
@media screen and (max-width: 600px), (orientation: portrait) {
    /* Styles for mobile OR portrait orientation */
}
```

NOT Operator

```
/* Applies to everything except print */
@media not print {
    .header {
        background: #333;
    }
}
```

Modern Media Query Features

Prefers-Color-Scheme

Detect if the user prefers light or dark mode:

```
/* Dark mode styles */
@media (prefers-color-scheme: dark) {
    body {
        background: #1a1a1a;
        color: #ffffff;
    }
}

/* Light mode styles */
@media (prefers-color-scheme: light) {
    body {
        background: #ffffff;
        color: #000000;
    }
}
```

Prefers-Reduced-Motion

Respect users who prefer less animation:

```
/* Reduce animations for users who prefer it */
@media (prefers-reduced-motion: reduce) {
    * {
        animation-duration: 0.01ms !important;
        transition-duration: 0.01ms !important;
    }
}
```

Practice Exercise

Try This: Create a simple webpage with a navigation menu that:

- Displays horizontally on desktop (≥768px)
- Becomes a vertical list on mobile (<768px)
- Changes the font size based on screen width

```
/* Solution */
.nav {
    list-style: none;
}

.nav li {
    display: inline-block;
    margin: 0 15px;
    font-size: 18px;
}

@media screen and (max-width: 767px) {
    .nav li {
        display: block;
        margin: 10px 0;
        font-size: 16px;
    }
}
```

Key Takeaways

- ✓ Media queries make your website responsive and adaptable
- ✓ Use `max-width` for mobile-first, `min-width` for desktop-first approaches
- ✓ Common breakpoints are around 600px (mobile), 900px (tablet), and 1200px (desktop)
- ✓ You can query many features: width, height, orientation, resolution, and user preferences
- ✓ Always test on multiple devices and screen sizes
- ✓ Modern media queries can detect dark mode, motion preferences, and more

Additional Resources

To learn more about media queries:

- ✓ MDN Web Docs: CSS Media Queries
- ✓ CSS-Tricks: A Complete Guide to CSS Media Queries
- ✓ Can I Use: Check browser support for media features
- ✓ Practice with responsive design tools in browser DevTools

🎉 Congratulations!

You now understand the fundamentals of CSS media queries. Start practicing by adding responsive styles to your own projects!