BONNIE COMPUTER HUB

Empowering Through Technology

FRONTEND WEB DEVELOPMENT – BEGINNER TRACK

WEEK 2/8: CSS FUNDAMENTALS + LAYOUTS

SESSION 1: CSS Basics, Selectors, Box Model

SESSION 2: Flexbox, Positioning

SESSION 3: GRID + RESPONSIVE LAYOUT

PREPARED BY: *Bonnie Computer Hub Team*

SESSION 1: CSS BASICS, SELECTORS, BOX MODEL

1. Introduction

Objective: Learn the fundamental concepts of CSS, understand different types of selectors, and master the CSS box model.

Relevance: CSS (Cascading Style Sheets) is the language that brings websites to life with colors, layouts, and visual appeal. Without CSS, websites would be plain text documents. Understanding CSS is essential for creating professional, visually appealing web interfaces.

2. Lesson Overview

Topics Covered:

- What is CSS and why it matters
- CSS syntax and implementation methods
- CSS selectors and specificity
- The CSS box model
- Basic styling properties

3. Core Content

What is CSS?

CSS (Cascading Style Sheets) is a styling language used to describe the presentation of a document written in HTML. While HTML provides the structure and content of a webpage, CSS controls how that content looks and is arranged on the screen.

CSS Implementation Methods

There are three ways to add CSS to your HTML:

1. Inline CSS: Applied directly to HTML elements using the style attribute.

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

2. Internal CSS: Placed within the <style> tags in the HTML document's <head> section.

```
<head>
  <style>
    p {
      color: blue;
      font-size: 16px;
    }
  </style>
</head>
```

3. External CSS (Recommended): Stored in a separate .css file and linked to the HTML document.

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

In styles.css:

```
p {
  color: blue;
  font-size: 16px;
}
```

CSS Syntax

CSS consists of **selectors** and **declarations**:

```
selector {
  property: value;  /* This is a declaration */
}
```

CSS Selectors

Selectors determine which HTML elements the CSS rules will apply to.

1. Element Selector: Selects all instances of a specific HTML element.

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

2. Class Selector: Selects elements with a specific class attribute.

```
.highlight {
  background-color: yellow;
}
```

3. ID Selector: Selects a single element with a specific ID attribute.

```
#header {
  font-size: 24px;
}
```

4. Attribute Selector: Selects elements based on an attribute or attribute value.

```
input[type="text"] {
  border: 1px solid gray;
}
```

5. Combinators:

- Descendant selector (space): div p selects all <p> elements inside elements
- Child selector (>): div > p selects all <p> elements that are direct children of elements
- Adjacent sibling selector (+): h1 + p selects the first <p> element directly after an <h1> element
- General sibling selector (~): h1 ~ p selects all <p> elements that follow an <h1> element

6. Pseudo-classes: Select elements based on a specific state.

```
a:hover {
  color: red;
}

li:first-child {
  font-weight: bold;
}
```

7. Pseudo-elements: Select and style parts of an element.

```
p::first-letter {
  font-size: 2em;
}

p::before {
  content: "→ ";
}
```

CSS Specificity

When multiple conflicting CSS rules target the same element, the browser follows specificity rules to determine which style to apply.

Specificity hierarchy (from lowest to highest):

1. Element selectors (p, div, etc.)
2. Class selectors (.class), attribute selectors ([type="text"]), and pseudo-classes (:hover)
3. ID selectors (#id)
4. Inline styles (style="...")
5. !important (overrides all other declarations)

The CSS Box Model

Every HTML element is represented as a rectangular box. The CSS box model describes this box and consists of:

1. **Content:** The actual content of the element (text, images, etc.)
2. **Padding:** The space between the content and the border
3. **Border:** A line that surrounds the padding
4. **Margin:** The space outside the border

```
div {
  /* Content dimensions */
  width: 300px;
  height: 200px;

  /* Padding */
  padding-top: 10px;
  padding-right: 20px;
  padding-bottom: 10px;
  padding-left: 20px;
  /* Shorthand: padding: 10px 20px 10px 20px; or padding: 10px 20px; */

  /* Border */
  border-width: 2px;
  border-style: solid;
  border-color: black;
  /* Shorthand: border: 2px solid black; */

  /* Margin */
  margin-top: 15px;
  margin-right: 15px;
```

```
  margin-bottom: 15px;
  margin-left: 15px;
  /* Shorthand: margin: 15px; */
}
```

By default, the width and height properties in CSS define the dimensions of the content area only. The total width of an element includes the content, padding, and border:

Total width = width + padding-left + padding-right + border-left + border-right
Total height = height + padding-top + padding-bottom + border-top + border-bottom

To change this behavior, you can use the box-sizing property:

```
* {
  box-sizing: border-box; /* Makes width and height include content, padding, and border */
}
```

Basic Styling Properties

1. Text Properties:

```
p {
  color: #333333;              /* Text color */
  font-family: Arial, sans-serif; /* Font typeface */
  font-size: 16px;             /* Font size */
  font-weight: bold;           /* Font weight: normal, bold, 100-900 */
  font-style: italic;          /* Font style: normal, italic, oblique */
  text-align: center;          /* Text alignment: left, right, center, justify */
  text-decoration: underline;   /* Text decoration: none, underline, line-through */
  line-height: 1.5;            /* Line height (leading) */
  letter-spacing: 1px;         /* Space between letters */
  word-spacing: 2px;           /* Space between words */
```

```
  text-transform: uppercase;      /* Changes case: none, uppercase, lowercase, capitalize */
}
```

2. Background Properties:

```
div {
  background-color: #f0f0f0;    /* Background color */
  background-image: url('image.jpg'); /* Background image */
  background-repeat: no-repeat; /* Image repetition: repeat, repeat-x, repeat-y, no-repeat */
  background-position: center;  /* Image position */
  background-size: cover;          /* Image sizing: auto, cover, contain, specific values */
  /* Shorthand: background: #f0f0f0 url('image.jpg') no-repeat center/cover; */
}
```

3. Border Properties:

```
div {
  border-width: 2px;          /* Border width */
  border-style: solid;        /* Border style: none, solid, dotted, dashed, etc. */
  border-color: black;        /* Border color */
  border-radius: 10px;        /* Rounded corners */
  /* Shorthand: border: 2px solid black; */
}
```

4. Dimension Properties:

```
div {
  width: 300px;               /* Element width */
  height: 200px;              /* Element height */
  min-width: 100px;           /* Minimum width */
  max-width: 500px;           /* Maximum width */
```

```
  min-height: 100px;          /* Minimum height */
  max-height: 500px;          /* Maximum height */
}
```

4. Assignments

Practice Exercise 1: Basic CSS Styling

Create an HTML page with different text elements (headings, paragraphs, lists) and apply various CSS properties to style them. Experiment with:

- Different font families, sizes, and colors
- Text alignment and decorations
- Background colors and borders

Practice Exercise 2: CSS Selectors

Create an HTML document with various elements and practice using different types of selectors to style them:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Selectors Practice</title>
  <link rel="stylesheet" href="selectors.css">
</head>
<body>
  <header id="main-header">
    <h1>CSS Selectors</h1>
    <nav>
      <ul>
        <li><a href="#" class="active">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
```

```
  </nav>
 </header>

 <section class="content">
  <article>
   <h2>Article 1</h2>
   <p>This is the first paragraph.</p>
   <p>This is the second paragraph.</p>
  </article>

  <article>
   <h2>Article 2</h2>
   <p>Another set of paragraphs.</p>
   <p>The last paragraph.</p>
  </article>
 </section>

 <footer>
  <p>&copy; 2025 Selector Practice</p>
 </footer>
</body>
</html>
```

In your CSS file, write rules using different selectors to achieve:

1. Style the header with a background color
2. Make all links blue and remove the underline
3. Style active links differently
4. Make the first paragraph of each article bold
5. Add a border only to the second article

6. Change the color of the last paragraph in the document

Practice Exercise 3: Box Model

Create a set of nested div elements and experiment with different box model properties:

```
<!DOCTYPE html>
<html>
<head>
  <title>Box Model Practice</title>
  <style>
    /* Add your CSS here */
  </style>
</head>
<body>
  <div class="container">
    <div class="box box1">Box 1</div>
    <div class="box box2">Box 2</div>
    <div class="box box3">Box 3</div>
  </div>
</body>
</html>
```

Style these boxes with:

1. Different dimensions (width and height)
2. Various padding and margin values
3. Different border styles
4. Try changing the box-sizing property and observe the differences

5. Knowledge Check

Reflection Questions:

1. What is the primary difference between inline, internal, and external CSS?
2. Why is the box model essential for understanding CSS layouts?
3. What happens when two CSS rules conflict? How does the browser decide which style to apply?
4. What is the difference between margin and padding?

Quiz:

1. Which CSS property controls the space between an element's content and its border?
   a) Margin
   b) Padding
   c) Border
   d) Spacing
2. Which selector has the highest specificity?
   a) Element selector
   b) Class selector
   c) ID selector
   d) Attribute selector
3. What does the CSS property box-sizing: border-box; do?
   a) Makes the element a box shape
   b) Includes padding and border in the element's width and height
   c) Adds a box shadow to the element
   d) Creates a box around the content only

4. Which of the following is NOT a valid CSS combinator?
   a) Descendant selector (space)
   b) Child selector (>)
   c) Adjacent sibling selector (+)
   d) Parent selector (<)
5. What will p::first-letter select?
   a) The first paragraph in a document
   b) The first letter of each paragraph
   c) The first word of each paragraph
   d) The first line of each paragraph

6. Additional Resources

Further Reading:

- MDN Web Docs: CSS Basics
- CSS-Tricks: A Complete Guide to the Box Model
- W3Schools: CSS Selectors Reference

Tools:

- CSS Diner - A fun game to practice CSS selectors
- HTML Color Codes - For finding and generating color codes
- CSS Box Model Visualizer - Interactive box model demonstration

SESSION 2: FLEXBOX, POSITIONING

1. Introduction

Objective: Master CSS Flexbox for creating flexible layouts and understand CSS positioning methods for precise control of elements.

Relevance: Modern websites require responsive, flexible layouts that adapt to different screen sizes. Flexbox is one of the most powerful tools for creating these layouts, while positioning gives you precise control over where elements appear on the page. Together, these techniques form the foundation of modern CSS layout.

2. Lesson Overview

Topics Covered:

- CSS Flexbox layout model
- Understanding flex containers and flex items
- Flex properties for alignment and ordering
- CSS positioning methods
- Z-index and stacking contexts

3. Core Content

Understanding CSS Flexbox

Flexbox (Flexible Box Module) is a one-dimensional layout method designed for arranging items in rows or columns. It solves many layout problems that were difficult with traditional CSS methods.

Key Concepts:

- **Flex Container:** The parent element with display: flex or display: inline-flex
- **Flex Items:** The direct children of the flex container
- **Main Axis:** The primary axis along which flex items are placed (horizontal for row, vertical for column)
- **Cross Axis:** The axis perpendicular to the main axis

Creating a Flex Container:

```
.container {
  display: flex; /* or display: inline-flex */
}
```

Flex Container Properties

1. flex-direction: Defines the direction of the main axis.

```
.container {
  flex-direction: row; /* default: left to right */
  /* Other values: row-reverse, column, column-reverse */
}
```

2. flex-wrap: Controls whether items wrap to new lines when they run out of space.

```
.container {
  flex-wrap: nowrap; /* default: no wrapping */
  /* Other values: wrap, wrap-reverse */
}
```

3. flex-flow: Shorthand for flex-direction and flex-wrap.

```
.container {
  flex-flow: row wrap; /* direction wrap */
}
```

4. justify-content: Aligns items along the main axis.

```
.container {
  justify-content: flex-start; /* default: items at start */
  /* Other values: flex-end, center, space-between, space-around, space-evenly */
}
```

5. align-items: Aligns items along the cross axis.

```
.container {
  align-items: stretch; /* default: items stretch to fill container height */
  /* Other values: flex-start, flex-end, center, baseline */
}
```

6. align-content: Aligns flex lines within the container (only matters when there are multiple lines).

```
.container {
  align-content: stretch; /* default */
  /* Other values: flex-start, flex-end, center, space-between, space-around, space-evenly */
}
```

Flex Item Properties

1. order: Controls the order in which items appear.

```
.item {
  order: 0; /* default */
  /* Higher numbers appear later in the order */
}
```

2. flex-grow: Determines how much an item can grow relative to others.

```
.item {
  flex-grow: 0; /* default: does not grow */
  /* Higher values grow more */
}
```

3. flex-shrink: Determines how much an item can shrink relative to others.

```
.item {
  flex-shrink: 1; /* default: can shrink */
  /* Higher values shrink more */
}
```

4. flex-basis: Sets the initial main size of an item.

```
.item {
  flex-basis: auto; /* default: based on content */
  /* Can be specific length (e.g., 100px) or percentage */
}
```

5. flex: Shorthand for flex-grow, flex-shrink, and flex-basis.

```
.item {
  flex: 0 1 auto; /* default: grow=0, shrink=1, basis=auto */
  /* Common values: flex: 1 (grow=1, shrink=1, basis=0%) */
}
```

6. align-self: Overrides the align-items value for specific items.

```
.item {
  align-self: auto; /* default: inherit from parent */
  /* Other values: stretch, flex-start, flex-end, center, baseline */
}
```

Common Flexbox Patterns

1. Equal-Width Columns:

```
.container {
  display: flex;
}
.item {
  flex: 1; /* All items get equal width */
}
```

2. Navbar with Space Between:

```
.navbar {
  display: flex;
  justify-content: space-between;
}
```

3. Centering an Item:

```
.container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh; /* Full viewport height */
}
```

4. Responsive Grid:

```
.grid {
  display: flex;
  flex-wrap: wrap;
}

.grid-item {
  flex: 1 0 300px; /* Grow, don't shrink, basis 300px */
  margin: 10px;
}
```

CSS Positioning

Positioning allows you to control exactly where elements appear on the page, sometimes outside the normal document flow.

Position Property Values:

```
.element {
  position: static | relative | absolute | fixed | sticky;
}
```

1. static (default): Elements remain in the normal document flow.

```
.element {
  position: static;
  /* Offset properties (top, right, bottom, left) have no effect */
```

}

2. relative: Elements remain in the normal flow but can be offset from their original position.

```
.element {
  position: relative;
  top: 20px;    /* Move 20px down from original position */
  left: 30px;   /* Move a 30px right from original position */
}
```

3. absolute: Elements are removed from the normal flow and positioned relative to their nearest positioned ancestor (or the document body).

```
.container {
  position: relative; /* Creates a positioning context for absolute elements */
}
.element {
  position: absolute;
  top: 0;      /* 0px from the top of the container */
  right: 0;    /* 0px from the right of the container */
  /* Creates a top-right corner positioning */
}
```

4. fixed: Elements are removed from the normal flow and positioned relative to the viewport (browser window). They stay in place even when scrolling.

```
.header {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  /* Header stays at the top of the viewport while scrolling */
}
```

5. sticky: Elements act like relative until the user scrolls to a threshold point, then act like fixed.

```
.section-heading {
  position: sticky;
  top: 0;
  /* Heading scrolls normally until it reaches the top, then sticks */
}
```

Offset Properties

When an element has a position other than static, you can use offset properties to place it:

- top: Distance from the top edge
- right: Distance from the right edge
- bottom: Distance from the bottom edge
- left: Distance from the left edge

Z-index and Stacking Order

The z-index property controls the vertical stacking order of elements that overlap. Higher values appear on top of elements with lower values.

```
.back-element {
  position: relative; /* z-index only works on positioned elements */
  z-index: 1;
}

.front-element {
  position: relative;
  z-index: 2; /* Will appear on top of .back-element */
}
```

Important Z-index Notes:

- Only works on positioned elements (not position: static)
- Creates a new stacking context
- Elements are rendered in order of appearance in the HTML by default (later elements on top)
- Negative values are allowed

Common Positioning Use Cases

1. Modal/Dialog Box:

```
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
  z-index: 100;
}

.modal {
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  z-index: 101;
}
```

2. Tooltips:

```
.tooltip-container {
  position: relative;
```

```
}

.tooltip {
  position: absolute;
  bottom: 100%;
  left: 50%;
  transform: translateX(-50%);
  display: none;
}

.tooltip-container:hover .tooltip {
  display: block;
}
```

3. Sticky Header:

```
header {
  position: sticky;
  top: 0;
  background-color: white;
  z-index: 10;
}
```

4. Assignments

Practice Exercise 1: Flexbox Navigation Bar

Create a responsive navigation bar using flexbox with the following features:

- Logo on the left
- Navigation links in the center
- Login/signup buttons on the right
- Navigation links should wrap on small screens

HTML Structure:

```
<!DOCTYPE html>
<html>
<head>
  <title>Flexbox Navigation</title>
  <link rel="stylesheet" href="nav.css">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
  <nav class="navbar">
    <div class="logo">
      <h1>Site Logo</h1>
    </div>
    <ul class="nav-links">
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Services</a></li>
      <li><a href="#">Portfolio</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
    <div class="auth-buttons">
      <button class="login">Login</button>
      <button class="signup">Sign Up</button>
    </div>
  </nav>

  <main>
    <h2>Welcome to our website!</h2>
    <p>This page demonstrates a responsive navbar using flexbox.</p>
```

```
  </main>
</body>
</html>
```

Practice Exercise 2: Flexbox Card Layout

Create a responsive card layout for a product display:

HTML Structure:

```
<!DOCTYPE html>
<html>
<head>
  <title>Flexbox Cards</title>
  <link rel="stylesheet" href="cards.css">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
  <h1>Featured Products</h1>

  <div class="card-container">
    <!-- Card 1 -->
    <div class="card">
      <div class="card-image">Product Image 1</div>
      <div class="card-content">
        <h3>Product Title</h3>
        <p>Product description goes here. This is a short overview of the product.</p>
        <div class="card-price">$19.99</div>
        <button class="buy-button">Add to Cart</button>
      </div>
    </div>
```

```
    <!-- Add 5 more identical cards -->
    <!-- Card 2 -->
    <div class="card"><!-- Same structure --></div>
    <!-- Card 3 -->
    <div class="card"><!-- Same structure --></div>
    <!-- Card 4 -->
    <div class="card"><!-- Same structure --></div>
    <!-- Card 5 -->
    <div class="card"><!-- Same structure --></div>
    <!-- Card 6 -->
    <div class="card"><!-- Same structure --></div>
  </div>
</body>
</html>
```

Requirements:

1. Cards should display in a row on large screens
2. Cards should wrap to multiple rows as the screen gets smaller
3. Each card should have a minimum width
4. Cards should have equal height in each row
5. Card content should be arranged with flexbox inside each card

Practice Exercise 3: Positioning Elements

Create a webpage with different positioning scenarios:

1. A fixed header that stays at the top of the page
2. A sidebar that sticks to the viewport when scrolling
3. A modal dialog box that appears centered on the screen
4. Content sections with relative positioning
5. A floating "back to top" button in the bottom-right corner

5. Knowledge Check

Reflection Questions:

1. How does Flexbox differ from traditional CSS layout methods?
2. When would you choose position: absolute over position: relative?
3. What is the difference between justify-content and align-items in Flexbox?
4. How can you center an element both horizontally and vertically using Flexbox vs. using positioning?

Quiz:

1. Which property controls the direction of the main axis in a flex container?
   a) flex-alignment
   b) flex-direction
   c) flex-flow
   d) flex-axis
2. What is the default value of the flex-grow property?
   a) 0
   b) 1
   c) auto
   d) none
3. Which position value removes an element from the normal document flow?
   a) static
   b) relative
   c) absolute
   d) None of the above

4. The z-index property works on elements with which position value?
   a) Only static
   b) Only absolute
   c) Any position except static
   d) All position values
5. Which flexbox alignment property would you use to create equal space between items?
   a) justify-content: space-between
   b) align-items: space-between
   c) align-content: space-between
   d) space-between: flex

6. Additional Resources

Further Reading:

- CSS-Tricks: A Complete Guide to Flexbox
- MDN Web Docs: CSS Flexible Box Layout
- MDN Web Docs: CSS Position

Tools:

- Flexbox Froggy - A game for learning Flexbox
- Flexbox Defense - Another game for learning Flexbox
- CSS Layout Generator - Visual tool for creating CSS layouts

SESSION 3: GRID + RESPONSIVE LAYOUT

1. Introduction

Objective: Master CSS Grid for two-dimensional layouts and learn how to create responsive designs that work well on all devices.

Relevance: Modern websites must function well on devices of all sizes, from smartphones to large desktop monitors. CSS Grid provides powerful tools for creating complex layouts, while responsive design principles ensure your site looks good everywhere. These skills are essential for any frontend developer in today's multi-device world.

2. Lesson Overview

Topics Covered:

- CSS Grid fundamentals
- Creating grid layouts
- Grid placement and alignment
- Responsive design principles
- Media queries
- Mobile-first development

3. Core Content

CSS Grid Introduction

CSS Grid Layout is a two-dimensional layout system designed specifically for creating grid-based user interfaces. Unlike Flexbox, which is one-dimensional, Grid allows you to control both rows and columns simultaneously.

Key Concepts:

- **Grid Container:** The element with display: grid or display: inline-grid
- **Grid Items:** The direct children of the grid container
- **Grid Lines:** The horizontal and vertical dividing lines that make up the grid
- **Grid Tracks:** The spaces between grid lines (rows and columns)
- **Grid Cells:** The intersection of a row and column

- **Grid Areas:** Rectangular spaces surrounded by four grid lines

Creating a Grid Container:

```
.container {
  display: grid; /* or display: inline-grid */
}
```

Defining Grid Structure

1. grid-template-columns and grid-template-rows: Define the columns and rows of the grid with their track sizes.

```
.container {
  grid-template-columns: 100px 200px 100px; /* 3 columns with fixed widths */
  grid-template-rows: auto 100px auto; /* 3 rows, 1st and 3rd auto-sized */
}
```

Using the fr unit (fraction):

```
.container {
  grid-template-columns: 1fr 2fr 1fr; /* Proportional widths */
  /* First and third columns get 1 part each, middle column gets 2 parts */
}
```

Using repeat() function:

```
.container {
  grid-template-columns: repeat(3, 1fr); /* 3 equal columns */
  grid-template-rows: repeat(2, 100px); /* 2 rows of 100px each */
}
```

2. grid-template-areas: Define named grid areas.

```
.container {
  grid-template-areas:
    "header header header"
    "sidebar content content"
    "footer footer footer";
```

```
}

.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.content { grid-area: content; }
.footer { grid-area: footer; }
```

3. grid-template: Shorthand for grid-template-rows, grid-template-columns, and grid-template-areas.

```
.container {
  grid-template:
    "header header header" auto
    "sidebar content content" 1fr
    "footer footer footer" auto
    / 1fr 2fr 1fr;
}
```

4. grid-gap properties: Control spacing between grid items.

```
.container {
  column-gap: 10px; /* Space between columns */
  row-gap: 15px; /* Space between rows */
  /* Shorthand: gap: 15px 10px; (row-gap column-gap) */
}
```

5. Implicit grid: When items are placed outside the explicitly defined grid, the implicit grid is created automatically.

```
.container {
  grid-auto-rows: 100px; /* Sets height for implicitly created rows */
  grid-auto-columns: 1fr; /* Sets width for implicitly created columns */
  grid-auto-flow: row; /* Controls how auto-placement works (row, column, dense) */
```

```
}
```

Placing Items on the Grid

1. Using line numbers:

```
.item {
  grid-column-start: 1; /* Start at first column line */
  grid-column-end: 3;   /* End at third column line (spanning 2 columns) */
  grid-row-start: 2;    /* Start at second row line */
  grid-row-end: 4;      /* End at fourth row line (spanning 2 rows) */

  /* Shorthand properties */
  grid-column: 1 / 3;   /* column-start / column-end */
  grid-row: 2 / 4;      /* row-start / row-end */

  /* Alternative syntax using 'span' */
  grid-column: 1 / span 2; /* Start at line 1 and span 2 columns */
  grid-row: 2 / span 2;    /* Start at line 2 and span 2 rows */
}
```

2. Using grid-area:

```
.item {
  /* grid-area: row-start / column-start / row-end / column-end */
  grid-area: 2 / 1 / 4 / 3;
}
```

3. Using named grid areas: (From the grid-template-areas example above)

```
.header {
  grid-area: header;
}
```

4. Using auto-placement: Items are placed automatically according to the grid-auto-flow algorithm.

```
.item {
  /* No specific placement properties */
}
```

Alignment in Grid Layouts

1. Aligning grid items:

```
.container {
  /* Align items along the row axis (horizontal) */
  justify-items: start | end | center | stretch;

  /* Align items along the column axis (vertical) */
  align-items: start | end | center | stretch;

  /* Shorthand for both */
  place-items: align-items justify-items;
}

.item {
  /* Override alignment for specific item */
  justify-self: start | end | center | stretch;
  align-self: start | end | center | stretch;
  place-self: align-self justify-self;
}
```

2. Aligning the grid tracks within the container:

```
.container {
  /* Align grid tracks within container (when grid is smaller than container) */
```

```
  justify-content: start | end | center | stretch | space-around | space-between | space-evenly;
  align-content: start | end | center | stretch | space-around | space-between | space-evenly;
  place-content: align-content justify-content;
}
```

Common Grid Patterns

1. Basic 12-Column Grid:

```
.grid {
  display: grid;
  grid-template-columns: repeat(12, 1fr);
  gap: 20px;
}

.span-3 { grid-column: span 3; }
.span-4 { grid-column: span 4; }
.span-6 { grid-column: span 6; }
.span-12 { grid-column: span 12; }
```

2. Card Layout:

```
.card-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: 20px;
}
```

3. Holy Grail Layout:

```
.container {
  display: grid;
  grid-template-areas:
```

```
    "header header header"
    "nav main aside"
    "footer footer footer";
  grid-template-columns: 200px 1fr 200px;
  grid-template-rows: auto 1fr auto;
  min-height: 100vh;
}

header { grid-area: header; }
nav { grid-area: nav; }
main { grid-area: main; }
aside { grid-area: aside; }
footer { grid-area: footer; }
```

Responsive Web Design

Responsive web design (RWD) is an approach that makes web pages render well on a variety of devices and window or screen sizes.

Core Principles:

1. **Fluid Grids:** Using relative units like percentages or fr units instead of fixed pixel dimensions
2. **Flexible Images:** Ensuring images scale within their containers
3. **Media Queries:** Applying different styles based on device characteristics

Media Queries

Media queries allow you to apply different CSS based on device characteristics or browser window size.

Syntax:

```
@media mediatype and (condition) {
  /* CSS rules to apply */
}
```

Common media types:

- all (default)
- screen (for screens)
- print (for print preview and printed pages)

Common conditions:

- width, min-width, max-width
- height, min-height, max-height
- orientation (portrait or landscape)
- aspect-ratio
- resolution

Examples:

```
/* Apply when viewport width is 600px or less */
@media screen and (max-width: 600px) {
  .container {
    grid-template-columns: 1fr;
  }
}

/* Apply when viewport width is between 601px and 900px */
@media screen and (min-width: 601px) and (max-width: 900px) {
  .container {
    grid-template-columns: 1fr 1fr;
  }
}

/* Apply when viewport width is 901px or more */
@media screen and (min-width: 901px) {
  .container {
```

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

/* Apply when in landscape orientation */
@media screen and (orientation: landscape) {
  .container {
    grid-template-areas:
      "header header"
      "nav main"
      "footer footer";
  }
}
```

Mobile-First Development

Mobile-first is an approach where you design for mobile devices first, then enhance the design for larger screens.

Benefits:

- Forces you to focus on essential content and functionality
- Typically results in faster-loading sites
- Adapts well to the growing mobile user base

Example:

```
/* Base styles for mobile devices */
.container {
  display: grid;
  grid-template-columns: 1fr;
  grid-template-areas:
    "header"
    "nav"
```

```
    "main"
    "aside"
    "footer";
}

/* Tablet styles */
@media screen and (min-width: 768px) {
  .container {
    grid-template-columns: 200px 1fr;
    grid-template-areas:
      "header header"
      "nav main"
      "aside main"
      "footer footer";
  }
}

/* Desktop styles */
@media screen and (min-width: 1024px) {
  .container {
    grid-template-columns: 200px 1fr 200px;
    grid-template-areas:
      "header header header"
      "nav main aside"
      "footer footer footer";
  }
}
```

Responsive Images and Media

1. Flexible Images:

```
img {
  max-width: 100%;
  height: auto;
}
```

2. CSS Background Images:

```
.hero {
  background-image: url('small.jpg');
  background-size: cover;
  background-position: center;
}

@media screen and (min-width: 768px) {
  .hero {
    background-image: url('medium.jpg');
  }
}

@media screen and (min-width: 1200px) {
  .hero {
    background-image: url('large.jpg');
  }
}
```

3. Picture Element for Art Direction:

```
<picture>
  <source media="(min-width: 1200px)" srcset="large.jpg">
  <source media="(min-width: 768px)" srcset="medium.jpg">
```

```
  <img src="small.jpg" alt="Responsive image">
</picture>
```

Responsive Typography

1. Using relative units:

```
body {
  font-size: 16px; /* Base font size */
}

h1 {
  font-size: 2em; /* 2 times the parent font size (32px) */
}

p {
  font-size: 1rem; /* 1 times the root font size (16px) */
}
```

2. Viewport units for responsive text:

```
h1 {
  font-size: 5vw; /* 5% of viewport width */
}
```

3. Fluid typography with calc():

```
h1 {
  /* Base size + scaling factor * viewport width */
  font-size: calc(20px + 2vw);
}
```

4. Clamp for controlled scaling:

```
h1 {
  /* min, preferred, max */
  font-size: clamp(24px, 5vw, 60px);
}
```

4. Assignments

Practice Exercise 1: Grid Photo Gallery

Create a responsive photo gallery using CSS Grid with the following features:

- Display images in a grid layout
- Images should maintain their aspect ratio
- Gallery should be responsive, showing fewer columns on smaller screens
- Some images should span multiple grid cells for visual interest

HTML Structure:

```
<!DOCTYPE html>
<html>
<head>
  <title>Grid Photo Gallery</title>
  <link rel="stylesheet" href="gallery.css">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
  <h1>Photo Gallery</h1>

  <div class="gallery">
    <div class="gallery-item">
      <img src="https://via.placeholder.com/400x300" alt="Gallery image 1">
    </div>
    <div class="gallery-item featured">
```

```
    <img src="https://via.placeholder.com/600x400" alt="Gallery image 2">
  </div>
  <!-- Add 10 more gallery items -->
 </div>
</body>
</html>
```

CSS Requirements:

1. Create a basic grid layout for all devices
2. Make featured items span multiple grid cells
3. Use media queries to adjust the number of columns based on screen width
4. Ensure all images maintain their aspect ratio
5. Add hover effects to the gallery items

Practice Exercise 2: Responsive Dashboard Layout

Create a responsive dashboard layout with CSS Grid that includes:

- Header
- Sidebar navigation
- Main content area with multiple widgets/cards
- Footer
- Proper layout changes for mobile, tablet, and desktop views

HTML Structure:

```
<!DOCTYPE html>
<html>
<head>
 <title>Responsive Dashboard</title>
 <link rel="stylesheet" href="dashboard.css">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

```
<body>
  <div class="dashboard">
    <header class="header">Dashboard Header</header>
    <nav class="sidebar">Navigation Sidebar</nav>
    <main class="main-content">
      <div class="widget widget-large">Large Widget</div>
      <div class="widget">Widget 1</div>
      <div class="widget">Widget 2</div>
      <div class="widget">Widget 3</div>
      <div class="widget">Widget 4</div>
      <div class="widget widget-wide">Wide Widget</div>
    </main>
    <footer class="footer">Dashboard Footer</footer>
  </div>
</body>
</html>
```

CSS Requirements:

1. Use grid-template-areas to define the dashboard layout
2. Create a different layout for mobile (stacked), tablet (simplified grid), and desktop (full grid)
3. The main content area should be a nested grid for the widgets
4. Some widgets should span multiple grid cells

Practice Exercise 3: Responsive Typography System

Create a responsive typography system with the following features:

- Base font sizes that adjust with viewport width
- Proper heading hierarchy (h1-h6)
- Text that remains readable at all viewport sizes
- Special treatment for featured text

HTML Structure:

```
<!DOCTYPE html>
<html>
<head>
  <title>Responsive Typography</title>
  <link rel="stylesheet" href="typography.css">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
  <header>
    <h1>Main Heading</h1>
    <p class="subtitle">This is a subtitle for the main heading</p>
  </header>

  <main>
    <section>
      <h2>Section Heading</h2>
      <p>Regular paragraph text. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla facilisi. Sed euismod, nisl vel ultricies lacinia, nisl nisl aliquam nisl, nec aliquam nisl nisl nec nisl.</p>

      <h3>Subsection Heading</h3>
      <p>More paragraph text with <strong>bold text</strong> and <em>emphasized text</em>.</p>

      <blockquote class="featured-text">
        This is a featured quote or callout that should stand out from the rest of the text.
```

```
    </blockquote>

    <h4>Smaller Heading</h4>
    <p>Even more paragraph text to demonstrate the typography system.</p>

    <h5>Even Smaller Heading</h5>
    <p>And yet more text to show different heading sizes.</p>

    <h6>Smallest Heading</h6>
    <p>Final paragraph to complete the hierarchy demonstration.</p>
  </section>
 </main>
</body>
</html>
```

CSS Requirements:

1. Use relative units (rem, em) for typography
2. Implement responsive font scaling using media queries
3. Create a typographic scale for headings
4. Style the featured text to stand out
5. Ensure proper line heights and spacing for readability

5. Knowledge Check

Reflection Questions:

1. What are the key differences between Flexbox and Grid? When would you choose one over the other?
2. How does the "fr" unit work in CSS Grid? What advantages does it offer over percentages?
3. Why is mobile-first development considered a good practice? What challenges might it present?

4. How can you ensure typography remains readable across different device sizes?

Quiz:

1. Which CSS Grid property would you use to define named template areas?
   a) grid-template
   b) grid-template-areas
   c) grid-area-template
   d) grid-named-areas
2. Which CSS function creates repeating patterns in grid templates?
   a) repeat()
   b) replicate()
   c) cycle()
   d) copy()
3. Which media query condition would target tablet devices in landscape orientation?
   a) @media (min-width: 768px) and (orientation: landscape)
   b) @media (orientation: tablet)
   c) @media tablet and landscape
   d) @media (min-device: tablet) and (orientation: landscape)
4. What is the purpose of the 'minmax()' function in CSS Grid?
   a) To set minimum and maximum grid sizes
   b) To create responsive grids without media queries
   c) To define a size range from minimum to maximum
   d) To calculate the average of two values

5. Which CSS property ensures an image will not exceed its container's width?
   a) max-width: 100%
   b) width: contain
   c) fluid-image: true
   d) responsive: true

6. Additional Resources

Further Reading:

- CSS-Tricks: A Complete Guide to CSS Grid
- MDN Web Docs: CSS Grid Layout
- Smashing Magazine: Responsive Design Patterns
- A List Apart: Responsive Web Design

Tools:

- Grid Garden - A game for learning CSS Grid
- Griddy - A CSS Grid playground
- Responsive Design Checker - Test how a website looks at different screen sizes
- Can I Use - Check browser support for CSS features