Web development

lesson 12

Nov 17, 2024

CSS Grid Layout: Introduction

- **What Is CSS Grid?**
  - CSS Grid is a powerful, two-dimensional layout system for CSS.
  - Allows you to create complex, responsive layouts for rows and columns.
- **Key Components:**
  - **Grid Container:** Parent element with `display: grid`.
  - **Grid Items:** Direct children of the grid container, positioned in grid cells.

Setting Up the Grid Container

- **Creating a Grid Container:**
  - Set up the grid container with `display: grid;`.
  - Grid items will be automatically arranged in rows.
- **Example:**

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

Defining Rows and Columns

- **Using `grid-template-rows` and `grid-template-columns`:**
  - Define the number and size of rows/columns.
- **Example:**

```
.container {
  display: grid;
  grid-template-rows: 100px 200px;
  grid-template-columns: 1fr 2fr;
}
```

  - Here, the container has two rows and two columns.

Fractional Units (fr)

- **What Is `fr`?**
  - `fr` stands for "fractional unit," allocating space relative to other fr units.
- **Example:**

```
.container {
  display: grid;
  grid-template-columns: 1fr 3fr;
}
```

  - Column 1 gets one fraction, and column 2 gets three fractions of available space.

Grid Gap

- **Spacing Between Grid Items:**
  - `gap` adds space between rows and columns.
  - Previously, `grid-gap` was used but is now shortened to `gap`.
- **Example:**

```
.container {
  display: grid;
  gap: 10px;
}
```

Grid Lines and Cell Positioning

- **Understanding Grid Lines:**
  - Grid lines are the horizontal and vertical lines that separate grid cells.
  - Use line numbers to position grid items precisely.
- **Example:**

```
.item {
  grid-column: 1 / 3;
  grid-row: 1 / 2;
}
```

Spanning Rows and Columns

- **Using `grid-column` and `grid-row` to Span Cells:**
  - Control the size of items by spanning across multiple rows or columns.
- **Example:**

```
.item {
  grid-column: span 2;
  grid-row: span 2;
}
```

Implicit vs. Explicit Grids

- **Creating Explicit and Implicit Grids:**
  - Explicit grids are defined using `grid-template-rows` and `grid-template-columns`.
  - Implicit grids are created automatically for items without a defined place.
- **Example:**

```
.container {
  display: grid;
  grid-template-rows: 100px 100px;
}
```

Auto-placement of Items

- **Using the Grid's Auto-placement Algorithm:**
  - By default, items fill the grid in row-first order.
  - Use `grid-auto-flow` to change this order to column-first or other options.
- **Example:**

```
.container {
  display: grid;
  grid-auto-flow: column;
}
```

Auto-sizing Rows and Columns

- **Using `auto` to Dynamically Size Items:**
  - The `auto` keyword lets the browser decide the size based on content.
- **Example:**

```
.container {
  grid-template-columns: 200px auto;
}
```

Repeat Function

- **Using `repeat()` for Consistency:**
  - `repeat()` helps avoid redundancy by defining repetitive sizes.
- **Example:**

```
.container {
  grid-template-columns: repeat(3, 1fr);
}
```

Minmax Function

- **Creating Flexible Ranges with `minmax()`:**
  - `minmax(min, max)` sets a minimum and maximum size for a track.
- **Example:**

```
.container {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

Aligning Items in the Grid

- **Aligning Items Horizontally and Vertically:**
  - Use `justify-items` and `align-items` for horizontal and vertical alignment.
- **Example:**

```
.container {
  justify-items: center;
  align-items: start;
}
```

Aligning the Grid as a Whole

- **Aligning the Entire Grid Within Its Container:**
  - Use `justify-content` and `align-content` for full grid alignment.
- **Example:**

```
.container {
  justify-content: space-around;
  align-content: center;
}
```

Grid Template Areas

- **Using `grid-template-areas` to Name Grid Sections:**
  - Assign names to cells and control layout in a readable way.
- **Example:**

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

Placing Items Using Grid Areas

- **Using `grid-area` to Position Items:**
  - Place items by referencing named grid areas.
- **Example:**

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

Nested Grids

- **Creating Grids Within Grids:**
  - Flexibility to use grid containers within grid items for complex layouts.
- **Example:**

```
.item {
  display: grid;
  grid-template-columns: 1fr 1fr;
}
```

Responsive Grid Layouts

- **Using Media Queries with Grid Layouts:**
  - Adjust grid layouts based on screen size for responsiveness.
- **Example:**

```
@media (max-width: 600px) {
  .container {
    grid-template-columns: 1fr;
  }
}
```

Practical Example 1: Basic Grid Layout

- **Simple 2x2 Grid:**
  - A straightforward example with two rows and two columns.
  - Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 100px 100px;
}
```

Practical Example 2: Gallery Layout

- **Responsive Image Gallery with Grid:**
  - Use repeat and gap to create a dynamic gallery.
  - Example:

```
.gallery {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 10px;
}
```

Summary and Best Practices

- **Key Points:**
  - Use `grid-template` properties to control the layout.
  - Combine `fr`, `repeat()`, and `minmax()` for flexibility.
  - Test layouts on various screen sizes.
- **Tips:**
  - Start with simple layouts, and experiment with nested grids.
  - Use `grid-template-areas` for clear, semantic layouts.