Unit 2: Styling more with CSS

CSS Selectors

CSS (Cascading Style Sheets) selectors are patterns used to select and style HTML elements in a web page. They allow you to target specific elements or groups of elements to apply styles such as colors, fonts, layout properties, and more

Here are some common CSS selectors:

1. **Element Selector**: Selects HTML elements by their element name.

```
p {
   /* Styles applied to all <p> elements */
}
```

2. **Class Selector:** Selects elements by their class attribute.

```
.classname {
   /* Styles applied to all elements with class="classname" */
}
```

3. **ID Selector:** Selects elements by their ID attribute.

```
#uniqueId {
   /* Styles applied to the element with id="uniqueId" */
}
```

4. **Universal Selector:** Selects all elements on a page.

```
* {
        /* Styles applied to all elements */
}
```

5. **Pseudo-classes and Pseudo-elements:** Selectors that define a special state of an element.

```
a:hover {
   /* Styles applied when hovering over a link */
}
```

```
p::first-line {
  /* Styles applied to the first line of <p> elements */
}
```

6. **Attribute Selector:** Selects elements based on their attribute values.

```
input[type="text"] {
  /* Styles applied to <input> elements with type="text" */
}
```

7. **Descendant Selecto**r: Selects an element that is a descendant of another specified element.

```
div p {
  /* Styles applied to all <p> elements inside a <div> */
}
```

8. **Child Selector:** Selects an element that is a direct child of another element.

```
ul > li {
  /* Styles applied to all <li> elements that are direct children of a <ul> */
}
```

9. **Grouping Selector:** Grouping selectors in CSS allows you to apply the same styles to multiple selectors simultaneously, reducing redundancy and making your code more concise. To group selectors, you simply separate them with a comma.'

```
h1, h2, h3 {
  font-family: "Arial", sans-serif;
  color: #333;
  /* Other common styles for heading elements */
}
```

CSS Properties and Values

CSS (Cascading Style Sheets) offers a wide range of properties and values to style HTML elements.

Here's an overview of some common CSS properties along with their potential values:

CSS properties and their associated values provide a wide range of options to style HTML elements. Here's a detailed explanation of some common CSS properties along with their respective values:

1. Color Properties:
    - `color`: Defines the text color of an element.
        - Values: Hexadecimal (#RRGGBB), RGB/RGBA (rgb/rgba(255, 0, 0, 0.5)), HSL/HSLA (hsl/hsla(120, 100%, 50%, 0.5)), color names ("red," "blue," etc.).
    - `background-color`: Sets the background color of an element.
        - Values: Same as `color` property values.
2. Typography:
    - `font-family`: Specifies the font family for text.
        - Values: Font names (e.g., "Arial," "Helvetica," sans-serif, serif, monospace).
    - `font-size`: Sets the size of the font.
        - Values: Pixels (px), em, rem, percentages (%), etc.
    - `font-weight`: Defines the thickness of the font.
        - Values: Normal, bold, bolder, lighter, 100-900.
3. Layout and Box Model:
    - `width` / `height`: Determines the width or height of an element.
        - Values: Pixels (px), percentages (%), em, rem, auto, etc.
    - `margin`: Controls the spacing outside an element.
        - Values: Lengths (px, em, rem), percentages (%), auto, etc.
    - `padding`: Defines the spacing inside an element.
        - Values: Same as `margin` property values.
    - `border`: Sets properties of an element's border.
        - Values: Border width, style (solid, dashed, dotted), color.
4. Positioning:
    - `position`: Specifies the type of positioning method used.
        - Values: Static, relative, absolute, fixed, sticky.
    - `top`, `right`, `bottom`, `left`: Adjusts the positioning of an element.

        - Values: Pixels (px), percentages (%), em, rem, auto, etc.
5. Display:
    - `display`: Defines how an element is displayed.
        - Values: Block, inline, inline-block, flex, grid, none, etc.
    - `visibility`: Controls the visibility of an element.
        - Values: Visible, hidden, collapse.
6. Flexbox and Grid:
    - `flex-direction`, `justify-content`, `align-items`: Flexbox properties for controlling layout.
        - Values: Various alignment and direction values like `flex-start`, `center`, `space-between`, etc.
    - `grid-template-columns`, `grid-template-rows`: Grid properties for defining columns and rows.
        - Values: Lengths, percentages, auto, fr (fractional unit), etc.
7. Text and Line Properties:
    - `text-align`: Aligns the text within an element.
        - Values: Left, right, center, justify, etc.
    - `line-height`: Sets the height of a line of text.
        - Values: Normal, number, length, percentages (%).
8. Animations and Transitions:
    - `animation`: Defines animations.
        - Values: Animation names, durations, timing functions, etc.
    - `transition`: Controls the transition effect for changing property values.
        - Values: Property to transition, duration, timing function, delay.

Browser Compatibility

Browser compatibility refers to the ability of a website, web application, or code to function correctly and consistently across different web browsers

Since various web browsers (such as Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, Opera, etc.) are developed independently, they may interpret HTML, CSS, and JavaScript code differently, leading to discrepancies in how a web page appears or behaves across these browsers.

Causes for Browser Incompatibility

- **Rendering Engine Differences:** Various browsers use different rendering engines (e.g., Blink, Gecko, WebKit, Trident) that interpret HTML, CSS, and JavaScript code differently. These engines may prioritize and handle certain aspects of web content in their unique ways, resulting in visual discrepancies.
- **CSS and HTML Interpretation:** Browsers may interpret CSS rules and HTML elements differently, causing variations in layout, spacing, and visual appearance. Some CSS properties or HTML elements might be supported or handled inconsistently across browsers.
- **JavaScript Executio**n: Differences in JavaScript engines among browsers can result in varying support for JavaScript features, performance discrepancies, and potential bugs or errors in scripts executed across different environments.
- **Vendor Prefixes and Non-Standard Features:** Historically, browser vendors introduced vendor prefixes as a way to implement experimental or non-standard CSS properties. These prefixes were intended to allow developers to use new and evolving CSS features that hadn't yet been fully standardized or were still under development. They were primarily used to test and experiment with new ideas, giving web developers early access to innovative styling options.

  The most common vendor prefixes include:

  - `-webkit-` (used by WebKit-based browsers like Safari and old versions of Chrome)
  - `-moz-` (used by Mozilla Firefox)
  - `-ms-` (used by Internet Explorer and Microsoft Edge before adopting the Blink rendering engine)
  - `-o-` (used by Opera)

  For example, a developer might use `-webkit-border-radius` to apply rounded corners in Safari or Chrome before the `border-radius` property became a standard feature.

  However, relying solely on these prefixed properties without providing standardized alternatives or versions for other browsers can lead to compatibility issues.

- **Standards Compliance and Interpretation:** Browsers might interpret web standards defined by organizations like W3C(World Wide Web Consortium) and

WHATWG ("Web Hypertext Application Technology Working Group.")differently, resulting in varied implementations of the same standards.

- **Legacy and Newer Browser Versions:** Older browser versions may lack support for newer web technologies, while newer versions may support advanced features not available in older ones. This discrepancy can cause compatibility problems when catering to a diverse user base.
- **Platform and Device Differences:** Browsers operate on various platforms (desktop, mobile, tablet) and devices with different screen sizes, resolutions, and capabilities. Optimizing for these diverse environments can lead to challenges in ensuring consistent rendering.
- **Update Cycles and Feature Adoption:** Different browsers have varying update cycles and timelines for adopting new web features or standards. This can lead to differences in feature support across browsers at any given time.

Strategies for Ensuring Cross-Browser Compatibility in Web Development"

- **Cross-Browser Testing:** Test on multiple browsers to identify issues and ensure consistent performance.
- **Standards Adherence:** Follow established HTML, CSS, and JavaScript standards to reduce compatibility problems.
- **Update Browsers and Libraries:** Encourage users to update browsers and ensure libraries are current for better support.
- **Browser-Specific Adjustments:** Consider browser-specific CSS or JavaScript when necessary, using conditional comments or feature detection.
- **Documentation and Community Suppo**rt: Seek guidance from browser vendor documentation and developer communities for solutions and insights into specific browser-related issues.

Dynamic vs Static Webpage

Aspect	Static Web Page	Dynamic Web Page
Content	Fixed content that remains unchanged until manually updated	Content is generated dynamically, often from a database
Data Handling	Predefined content, no interaction or data processing	Interacts with databases, APIs, or user input for content

Loading Speed	Generally faster as content is pre-built and cached	May have longer load times due to content generation
Interactivity	Limited or no user interaction	Supports user interaction, forms, and real-time updates
Technologies Used	HTML, CSS, possibly minimal JavaScript	Utilizes server-side scripting (e.g., PHP, Node.js), AJAX
Updates and Changes	Manual updates required for content changes	Content can update dynamically without manual intervention
Personalization	Limited or no ability for personalization	Can personalize content based on user preferences or actions
Examples	Brochure websites, informational pages	Social media feeds, online stores, interactive web apps

Page Layout Techniques

Page layout techniques refer to the methods used to organize and present content within a web page or document to enhance readability, usability, and visual appeal.

Here are some common page layout techniques:

1. Normal Flow (Static or Document Flow):
2. The Display Property:
3. Flexbox
4. Grid
5. Floats
6. Positioning
7. Table Layout
8. Multi-Column layout

Normal Flow(static or Document Flow)

The Normal Flow (also known as Static or Document Flow) in web design refers to the default behavior of elements within an HTML document when no specific layout or positioning is applied using CSS.

Key characteristics of the Normal Flow include:

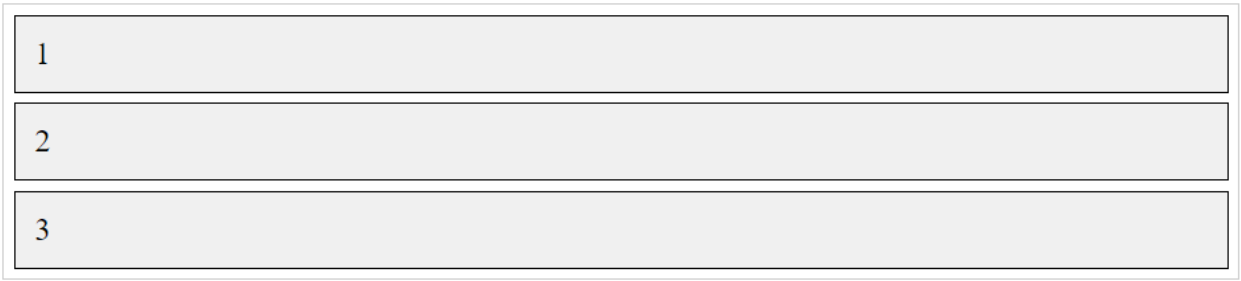
1. **Sequential Display:** Elements are displayed one after another according to their order in the HTML, flowing in either a horizontal (left to right) or vertical (top to bottom) direction.
2. **Width of Parent Container:** Elements within the normal flow generally occupy the entire width available within their parent container, expanding to fill the available space horizontally.
3. **Subsequent Element Flow**: Each subsequent element is positioned immediately after the previous one, without overlapping or breaking the flow unless specified otherwise.
4. **Inline and Block Elements**: Inline elements, such as spans or anchors, flow horizontally within the text, while block-level elements, such as divs or paragraphs, create a new line and occupy the full width available.
5. **Responsive Behavior:** Elements within the normal flow are responsive by default, adjusting to different screen sizes and resolutions while maintaining their flow order.
6. **Layout Flexibility:** The layout created by the Normal Flow can be altered using various CSS properties and layout models, like Flexbox, Grid, Floats, or Positioning, to achieve more complex or customized layouts.

Certainly! Here's an example illustrating the characteristics of the Normal Flow in HTML and CSS:

HTML Structure:

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
  <style>
    .container {
      border: 1px solid #ccc;
    }
```

```
    .box {
      border: 1px solid #000;
      background-color: #f0f0f0;
      margin: 5px;
      padding: 10px;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="box">1</div>
    <div class="box">2</div>
    <div class="box">3</div>
  </div>
</body>
</html>
```



Explanation of the characteristics:

- **Sequential Display:** The HTML consists of three `div` elements (with class `.box`) placed within a container (with class `.container`).
- **Width of Parent Container:** By default, the `.box` elements take up the width of the `.container` div.
- **Subsequent Element Flow:** Each `.box` div follows the previous one in the order they appear in the HTML, displayed one after another.
- **Inline and Block Elements**: Although `div` elements are block-level by default, they are positioned one below the other due to the Normal Flow behavior.
- **Responsive Behavior:** These elements will adjust their layout according to the available space, adjusting their flow within the container.

Example 2:

```
<!DOCTYPE html>
<html>
<head>
 <title>Normal Flow Layout Example</title>
 <style>
  /* Reset default margin and padding */
  * {
   margin: 0;
   padding: 0;
   box-sizing: border-box;
  }

  /* Basic styles for layout elements */
  .container {
   width: 80%;
   margin: 0 auto;
   border: 1px solid #ccc;
   padding: 20px;
   overflow: hidden;
  }

  header, nav, main, footer {
   margin-bottom: 20px;
  }

  header, footer {
   background-color: #f2f2f2;
   padding: 10px;
  }

  nav ul {
   list-style: none;
  }

  nav ul li {
   display: inline;
   margin-right: 10px;
  }
```

```
    aside {
      float: right;
      width: 30%;
      background-color: #f9f9f9;
      padding: 10px;
    }
  </style>
</head>
<body>
  <div class="container">
    <header>
      <h1>Normal Flow Layout Example</h1>
    </header>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Services</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
    <main>
      <article>
        <h2>Article Title</h2>
        <p>This is a sample article text. Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
      </article>
    </main>
    <footer>
      <p>&copy; 2023 Normal Flow Layout Example</p>
    </footer>
  </div>
</body>
</html>
```

2. The Display Property

The `display` property plays a significant role in defining how elements are rendered and laid out on a web page.
Different `display` property values influence the layout and behavior of HTML elements.
Here are several common values for the `display` property used for layout design:

`display: block;`

- Behavior:
  - Generates a block-level box.
  - Starts on a new line and occupies the full width available within its container.
  - Vertical space (height, margin, padding) and horizontal space (width) can be adjusted.
- Usage:
  - Suitable for structural elements like `<div>`, `<p>`, `<h1>`-`<h6>`, which need to start on new lines and have distinct blocks.
  - Typically used for elements that need to contain other elements or create sections on a webpage.

`display: inline;`

- Behavior:
  - Generates an inline-level box.

    - Does not start on a new line and only occupies the space required by its content.
    - Vertical space (height, margin, padding) are applied vertically, but width is ignored.
- Usage:
    - Ideal for inline text elements like `<span>`, `<a>`, `<strong>`, `<em>`.
    - Often used for styling elements within a line of text without causing line breaks.

`display: inline-block;`

- Behavior:
    - Combines aspects of both block and inline elements.
    - Creates an inline-level box but retains block-level properties like width, height, padding, margin.
    - Allows elements to flow inline while still respecting block-level properties.
- Usage:
    - Useful for elements that need block-level properties but also need to sit inline, like creating grids, navigation menus, or elements in a line with block features.

```
<!DOCTYPE html>
<html>
<head>
 <title>Display: Inline vs Block vs Inline-Block</title>
 <style>
  /* General styling */
  .container {
   width: 60%;
   margin: 20px auto;
  }

  /* Style for container boxes */
  .inline-example, .block-example, .inline-block-example {
   border: 1px solid #ccc;
   padding: 10px;
   margin-bottom: 20px;
  }

  /* Inline elements */
  .inline-example span {
   display: inline;
   margin: 5px;
```

```
    padding: 8px;
    background-color: #f2f2f2;
   }

   /* Block elements */
   .block-example div {
    display: block;
    margin: 5px;
    padding: 8px;
    background-color: #f2f2f2;
   }

   /* Inline-block elements */
   .inline-block-example div {
    display: inline-block;
    width: 100px;
    height: 100px;
    margin: 5px;
    padding: 8px;
    background-color: #f2f2f2;
    text-align: center;
    line-height: 100px;
   }
  </style>
 </head>
 <body>
  <div class="container">
   <h2>Display: Inline vs Block vs Inline-Block</h2>

   <!-- Applying borders to container divs -->
   <div class="inline-example">
    <h3>Inline Elements (Spans)</h3>
    <span>Span 1</span>
    <span>Span 2</span>
    <span>Span 3</span>
   </div>

   <div class="block-example">
    <h3>Block Elements (Divs)</h3>
    <div>Div 1</div>
    <div>Div 2</div>
    <div>Div 3</div>
   </div>

   <div class="inline-block-example">
    <h3>Inline-Block Elements (Divs)</h3>
    <div>Inline-Block 1</div>
```

```
    <div>Inline-Block 2</div>
    <div>Inline-Block 3</div>
   </div>
  </div>
</body>
</html>
```

Display: Inline vs Block vs Inline-Block

Inline Elements (Spans)

Span 1 Span 2 Span 3

Block Elements (Divs)

Div 1

Div 2

Div 3

Inline-Block Elements (Divs)

Inline-Block 1 Inline-Block 2 Inline-Block 3

Multi-Column Layout

The multi-column layout in CSS allows content to flow into multiple columns, creating a newspaper-like or magazine-like appearance. It's particularly useful for dividing long content into more digestible portions across multiple columns.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Multi-Column Layout Example</title>
  <style>
    .multi-column-container {
      column-count: 3; /* Number of columns */
      column-gap: 20px; /* Gap between columns */
      width: 80%; /* Width of the container */
      margin: 20px auto;
    }

    .multi-column-content {
      margin-bottom: 20px;
      padding: 10px;
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>
  <div class="multi-column-container">
    <div class="multi-column-content">
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut lacinia, lectus eget efficitur imperdiet,
eros nisl malesuada dui, id congue lorem leo sit amet velit.</p>
      <!-- Add more content here -->
    </div>
    <div class="multi-column-content">
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut lacinia, lectus eget efficitur imperdiet,
eros nisl malesuada dui, id congue lorem leo sit amet velit.</p>
      <!-- Add more content here -->
    </div>
    <div class="multi-column-content">
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut lacinia, lectus eget efficitur imperdiet,
eros nisl malesuada dui, id congue lorem leo sit amet velit.</p>
      <!-- Add more content here -->
    </div>

  </div>
</body>
</html>
```

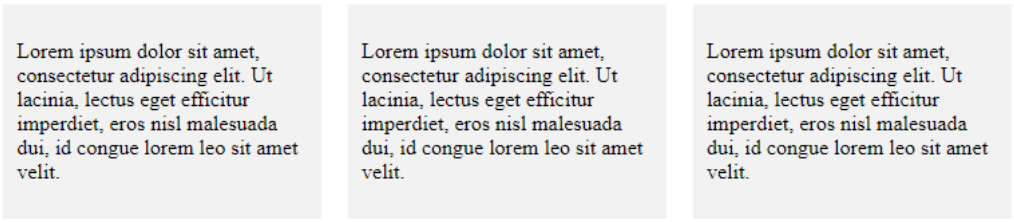
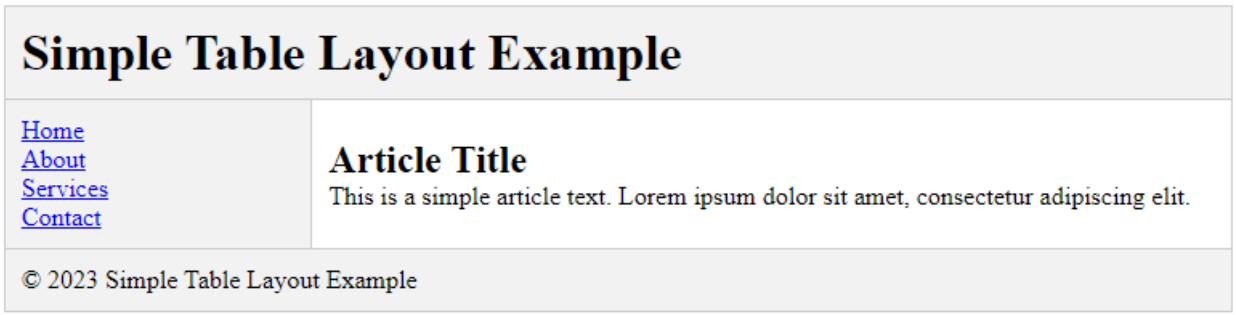


Table Layout

In web design, table layouts refer to structuring webpage content using HTML tables. This method involves organizing content into rows and columns using `<table>`, `<tr>`, `<th>`, and `<td>` HTML elements. However, using tables for layout purposes has become less common due to several drawbacks like accessibility issues, limited responsiveness, and complexity in maintaining the code

```
<!DOCTYPE html>
<html>
<head>
 <title>Simple Table Layout Example</title>
 <style>
  /* Reset default margin and padding */
  * {
   margin: 0;
   padding: 0;
  }

  /* Basic styles for layout elements */
  table {
   width: 80%;
   margin: 20px auto;
   border-collapse: collapse;
  }

  ul {
   list-style: none; /* Remove bullets */
  }

  th, td {
   border: 1px solid #ccc;
   padding: 10px;
  }
```

```
    header, footer {
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>
  <table>
    <tr>
      <td colspan="2" style="background-color: #f2f2f2;">
        <header>
          <h1>Simple Table Layout Example</h1>
        </header>
      </td>
    </tr>
    <tr>
      <td style="width: 25%; background-color: #f2f2f2;">
        <nav>
          <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">About</a></li>
            <li><a href="#">Services</a></li>
            <li><a href="#">Contact</a></li>
          </ul>
        </nav>
      </td>
      <td style="background-color: #fff;">
        <main>
          <article>
            <h2>Article Title</h2>
            <p>This is a simple article text. Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
          </article>
        </main>
      </td>
    </tr>
    <tr>
      <td colspan="2" style="background-color: #f2f2f2;">
        <footer>
          <p>&copy; 2023 Simple Table Layout Example</p>
        </footer>
      </td>
    </tr>
  </table>
</body>
</html>
```



Float Layout

- Float-based layout techniques involve using the CSS float property to position elements within a container.
- The CSS float property is used to specify how an element should float within its container. When an element is floated, it is taken out of the normal document flow and shifted to the left or right side of its containing element, allowing content to flow around it.

Syntax:

```
.element {
  float: left | right | none | inherit;
}
```

Values

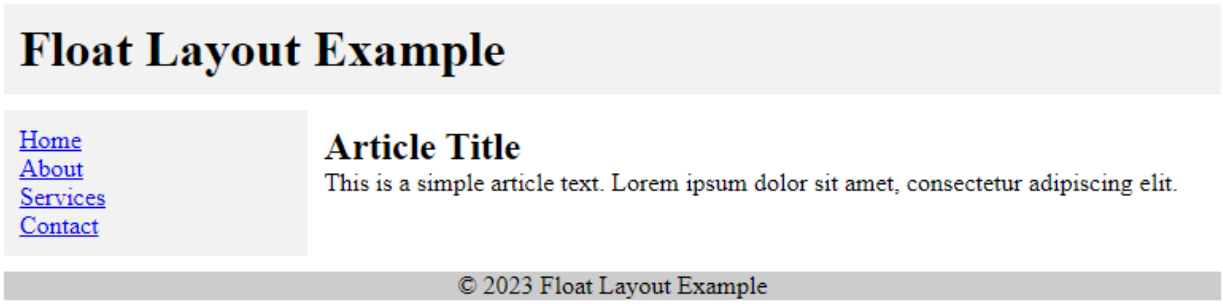
- `left`: The element floats to the left side of its container, allowing content to flow to its right side.
- `right`: The element floats to the right side of its container, allowing content to flow to its left side.
- `none`: The default value that ensures the element does not float.
- `inherit`: Inherits the float value from its parent element.

Example:

```
<!DOCTYPE html>
<html>
<head>
 <title>Float Layout Example</title>
 <style>
  /* Reset default margin and padding */
  * {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
```

```
    }

    .container {
      width: 80%;
      margin: 20px auto;
    }

    ul {
      list-style: none; /* Remove bullets */
    }

    header {
      background-color: #f2f2f2;
      padding: 10px;
      margin-bottom: 10px;
    }

    nav {
      float: left; /* Float the navigation to the left */
      width: 25%;
      background-color: #f2f2f2;
      padding: 10px;
    }

    main {
      float: right; /* Float the main content to the right */
      width: 75%;
      background-color: #fff;
      padding: 10px;
    }

    footer{
      float: left;
      width: 100%;
      text-align: center;
      background-color: #ccc;
      margin-top:10px ;
    }
  </style>
</head>
<body>
  <div class="container">
    <header>
      <h1>Float Layout Example</h1>
    </header>
    <nav>
      <ul>
```

```
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Services</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
  <main>
    <article>
      <h2>Article Title</h2>
      <p>This is a simple article text. Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
    </article>
  </main>
  <footer>
    <p>&copy; 2023 Float Layout Example</p>
  </footer>
 </div>
</body>
</html>
```



Positioning

In CSS, the position property determines how an HTML element is positioned within its parent or containing element. It offers several values that control the positioning behavior of an element on a web page.

The `position` property can take the following values:

1. `static`: This is the default value. Elements with `position: static;` follow the normal flow of the document. They are positioned according to the normal flow of the page, and any offsets (top, right, bottom, left) have no effect on statically positioned elements.
2. `relative`: Setting an element to `position: relative;` allows you to adjust its position relative to its default position. You can use `top`, `right`, `bottom`, or `left`

properties to shift the element from its original position, but it still affects the layout of surrounding elements.

3. `absolute`: Elements with `position: absolute;` are removed from the normal document flow and positioned relative to their closest positioned ancestor (an ancestor element with a position other than `static`). They're positioned using `top`, `right`, `bottom`, or `left` relative to this ancestor. If no positioned ancestor is found, the element is positioned relative to the document itself.
4. `fixed`: Elements with `position: fixed;` are removed from the normal flow and positioned relative to the viewport (browser window). They stay fixed in their position even when scrolling.

https://www.w3schools.com/css/css_positioning.asp

Grid Layout

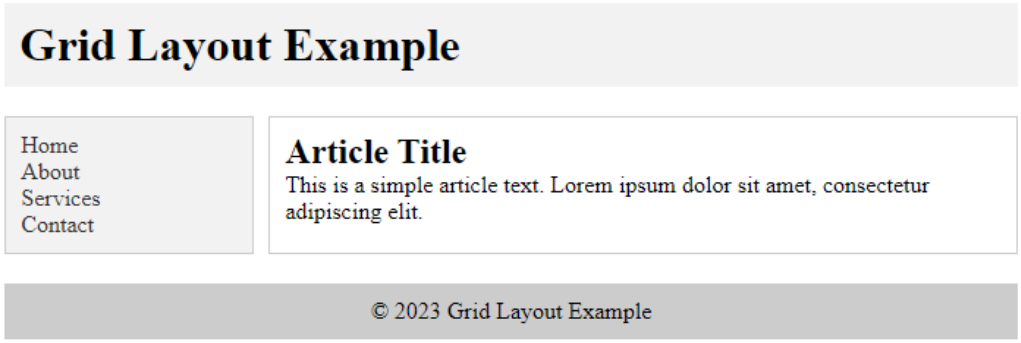
- The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.
- A grid layout consists of a parent element, with one or more child elements
- An HTML element becomes a grid container when its `display` property is set to `grid` or `inline-grid`.
- The difference between the values inline-grid and grid is that the inline-grid will make the element inline while grid will make it a block-level element.
- The vertical lines of grid items are called *columns*.
- The horizontal lines of grid items are called *rows*.
- The spaces between each column/row are called *gaps*.
- You can adjust the gap size by using one of the following properties:
    - `Column-gap: /* Defines the gap between columns */`
    - `Row-gap/* Defines the gap between rows */`
    - `Gap: set both the row gap and the column gap in one value`

https://www.w3schools.com/css/css_grid.asp

```
<!DOCTYPE html>
<html>
<head>
  <title>Grid Layout Example</title>
  <style>
    /* Reset default margin and padding */
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    /* Grid and layout styles */
    .container {
      width: 80%;
      margin: 20px auto;
      display: grid;
      grid-template-columns: 1fr 3fr; /* Two columns: 1/4 and 3/4 of the width */
      gap: 10px; /* Gap between grid columns and rows */
    }

    header {
      grid-column: 1 / -1; /* Span across both columns */
      background-color: #f2f2f2;
      padding: 10px;
      margin-bottom: 10px;
    }

    nav {
      background-color: #f2f2f2;
      padding: 10px;
    }

    main {
      background-color: #fff;
      padding: 10px;
    }

    footer {
      grid-column: 1 / -1; /* Span across both columns */
      text-align: center;
      background-color: #ccc;
      margin-top: 10px;
      padding: 10px;
```

```
    }

    nav, main {
      border: 1px solid #ccc; /* Adding border for visualization */
    }

    @media (max-width: 768px) {
      /* Responsive layout adjustments */
      .container {
        grid-template-columns: 1fr; /* Display as a single column on smaller screens */
      }

      nav, main {
        border: none; /* Remove border on smaller screens */
      }
    }

    /* Removing bullets from the navigation menu */
    nav ul {
      list-style: none; /* Remove bullets from the list */
      padding: 0; /* Remove default padding */
    }

    nav li {
      display: block; /* Display list items vertically */
      margin-right: 10px; /* Add spacing between list items */
    }

    nav li a {
      text-decoration: none; /* Remove underline from links */
      color: #333; /* Change link color */
    }
  </style>
</head>
<body>
  <div class="container">
    <header>
      <h1>Grid Layout Example</h1>
    </header>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Services</a></li>
```

```
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
    <main>
      <article>
        <h2>Article Title</h2>
        <p>This is a simple article text. Lorem ipsum dolor sit amet, consectetur
adipiscing elit.</p>
      </article>
    </main>
    <footer>
      <p>&copy; 2023 Grid Layout Example</p>
    </footer>
  </div>
</body>
</html>
```



Flex Layout

Flexbox, short for Flexible Box Layout, is a CSS (Cascading Style Sheets) layout model designed to create more efficient and responsive designs for user interfaces within web applications and websites.
The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.
To start using the Flexbox model, you need to first define a flex container.

The CSS Flexbox Container Properties

The following table lists all the CSS Flexbox Container properties:

Property	Description
align-content	Modifies the behavior of the flex-wrap property. It is similar to align-items, but instead of aligning flex items, it aligns flex lines
align-items	Vertically aligns the flex items when the items do not use all available space on the cross-axis
display	Specifies the type of box used for an HTML element
flex-direction	Specifies the direction of the flexible items inside a flex container
flex-flow	A shorthand property for flex-direction and flex-wrap
flex-wrap	Specifies whether the flex items should wrap or not, if there is not enough room for them on one flex line
justify-content	Horizontally aligns the flex items when the items do not use all available space on the main-axis

Child Elements (Items)

The direct child elements of a flex container automatically becomes flexible (flex) items.

The CSS Flexbox Items Properties

The following table lists all the CSS Flexbox Items properties:

Property	Description
align-self	Specifies the alignment for a flex item (overrides the flex container's align-items property)
flex	A shorthand property for the flex-grow, flex-shrink, and the flex-basis properties
flex-basis	Specifies the initial length of a flex item
flex-grow	Specifies how much a flex item will grow relative to the rest of the flex items inside the same container
flex-shrink	Specifies how much a flex item will shrink relative to the rest of the flex items inside the same container
order	Specifies the order of the flex items inside the same container