Day 9: CSS Selectors and Properties

Date: 14-06-24

---

1. CSS Selectors: Selectors are patterns used to select the elements you want to style.

- **Basic Selectors:**
    - **Element Selector:** Selects all elements of a given type.

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

 Class Selector: Selects elements with a specific class attribute.

```
 .my-class {
  font-size: 14px;
}
```

 ID Selector: Selects an element with a specific ID attribute.

-
- #my-id {
-   text-align: center;
- }

 Attribute Selectors: Select elements based on an attribute or attribute value.

```
 a[href] {
  color: green;
}
a[target="_blank"] {
  font-weight: bold;
}
```

 Combinator Selectors:

- **Descendant Selector:** Selects all elements that are descendants of a specified element.

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

 Child Selector: Selects all elements that are direct children of a specified element.

```
 div > p {
  font-size: 18px;
}
```

 Adjacent Sibling Selector: Selects an element that is the next sibling of a specified element.

```
 h1 + p {
  margin-top: 20px;
}
```

 General Sibling Selector: Selects all siblings of a specified element.

-
- h1 ~ p {
-   color: gray;
- }

 Pseudo-class Selectors: Apply styles to elements based on their state.

```
 a:hover {
  color: red;
}
input:focus {
  border: 2px solid blue;
}
```

 Pseudo-element Selectors: Apply styles to a part of an element.

- p::first-line {
-   font-weight: bold;
- }
- p::before {
-   content: "Note: ";
-   font-weight: bold;
- }
-

2. CSS Properties: Properties define the styles applied to the selected elements.

- **Text Properties:**

    - color: Sets the color of the text.

▯ p {

color: blue;

}

▯ font-size: Sets the size of the font.

▯ p {

font-size: 16px;

}

▯ text-align: Aligns the text inside an element.

- ▯
- h1 {
-   text-align: center;
- }

▯ **Box Model Properties:**

- width and height: Set the width and height of an element.

▯ div {

width: 100px;

height: 50px;

}

▯ padding: Adds space inside the element, around the content.

▯ div {

padding: 10px;

}

▯ margin: Adds space outside the element, around the border.

▯ div {

margin: 20px;

}

▯ border: Sets the border around an element.

- ▯
- div {
-   border: 1px solid black;

- }

 Background Properties:

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

 body {

 background-color: #f0f0f0;

}

 background-image: Sets a background image for an element.

-
- div {
- background-image: url('image.jpg');
- }

 Display and Positioning Properties:

- display: Specifies the display behavior of an element.

 .hidden {

 display: none;

}

 position: Specifies the positioning method used for an element (static, relative, absolute, fixed, sticky).

-
- .absolute {
- position: absolute;
- top: 50px;
- left: 50px;
- }

 Flexbox Properties:

- display: flex: Defines a flex container and enables a flex context for all its direct children.

 .container {

 display: flex;

}

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

 .container {

```
  justify-content: center;
}
```

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

-
    - .container {
    -   align-items: center;
    - }
    -

3. Practical Examples:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Selectors and Properties</title>
  <style>
    /* Element Selector */
    p {
      color: blue;
      font-size: 14px;
    }

    /* Class Selector */
    .highlight {
      background-color: yellow;
    }

    /* ID Selector */
    #unique {
      font-weight: bold;
      text-align: center;
    }

    /* Attribute Selector */
    a[href^="https"] {
      color: green;
    }

    /* Descendant Selector */
    div p {
      margin-left: 20px;
    }

    /* Child Selector */
    ul > li {
      list-style-type: square;
    }

    /* Adjacent Sibling Selector */
    h1 + p {
      font-style: italic;
    }

    /* General Sibling Selector */
    h1 ~ p {
      color: gray;
    }

    /* Pseudo-class Selector */
    a:hover {
      text-decoration: underline;
    }
```

```
    /* Pseudo-element Selector */
    p::first-letter {
      font-size: 20px;
      color: red;
    }
  </style>
</head>
<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph.</p>
  <p class="highlight">This is a highlighted paragraph.</p>
  <p id="unique">This is a unique paragraph.</p>
  <a href="https://example.com">This is a link.</a>
  <div>
    <p>This is a paragraph inside a div.</p>
  </div>
  <ul>
    <li>List item 1</li>
    <li>List item 2</li>
  </ul>
  <a href="https://example.com">Hover over this link.</a>
</body>
</html>
```

Day 10: CSS Box Model and Fluid Layouts

Date: 17-06-24

Summary of the Day: On the tenth day of our web development training, we explored two important concepts in CSS: the CSS Box Model and fluid layouts. Understanding these topics is essential for creating well-structured and responsive web pages. The session covered the components of the CSS Box Model and techniques for designing fluid, flexible layouts.

Detailed Notes:

1. CSS Box Model: The CSS Box Model is a fundamental concept that describes how elements are structured and spaced on a web page.

- **Components of the Box Model:**
  - **Content:** The actual content of the element, such as text or an image.
  - **Padding:** The space between the content and the border. It increases the size of the element without affecting its external dimensions.
  - **Border:** A line surrounding the padding (if any) and content.
  - **Margin:** The space outside the border, separating the element from other elements on the page.
- **Visual Representation:**

```
▯ element {
  width: 100px;
  height: 100px;
  padding: 10px;
  border: 5px solid black;
  margin: 15px;
}
```

This would result in:

- Content: 100px x 100px
- Padding: 10px on all sides (total size becomes 120px x 120px)
- Border: 5px on all sides (total size becomes 130px x 130px)
- Margin: 15px on all sides (total space occupied becomes 160px x 160px)

▯ **Example:**

- <style>
-   .box {
-     width: 100px;
-     height: 100px;
-     padding: 10px;
-     border: 5px solid black;
-     margin: 15px;
-     background-color: lightblue;
-   }

- </style>
- <div class="box">Box Model Example</div>
-

2. Fluid Layouts: Fluid layouts, also known as liquid layouts, adapt to the size of the user's viewport, making web pages more responsive.

- **Percentage-Based Widths:** Using percentages allows elements to resize relative to their parent container.

```
 .container {
  width: 80%; /* 80% of the parent container's width */
  margin: 0 auto; /* Center the container */
}
```

 Viewport Units: Viewport units (vw and vh) are relative to the size of the viewport.

- 1vw is 1% of the viewport width.
- 1vh is 1% of the viewport height.

```
 .responsive-box {
  width: 50vw; /* 50% of the viewport width */
  height: 50vh; /* 50% of the viewport height */
  background-color: lightgreen;
}
```

 Flexbox: Flexbox is a powerful layout module that allows for the creation of flexible and responsive layouts.

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  justify-content: space-around;
}

.flex-item {
  flex: 1 1 auto;
  margin: 10px;
  background-color: lightcoral;
}
```

Example:

```
•  <style>
•   .flex-container {
•    display: flex;
•    flex-wrap: wrap;
•    justify-content: space-around;
•   }
•
•   .flex-item {
•    flex: 1 1 auto;
•    margin: 10px;
•    background-color: lightcoral;
•    padding: 20px;
•    text-align: center;
•   }
•  </style>
•  <div class="flex-container">
•   <div class="flex-item">Item 1</div>
•   <div class="flex-item">Item 2</div>
•   <div class="flex-item">Item 3</div>
•  </div>
•
```

3. Practical Examples:

Example with Box Model:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Box Model Example</title>
 <style>
```

```
  .box {
   width: 200px;
   padding: 20px;
   border: 5px solid black;
   margin: 15px;
   background-color: lightblue;
  }
 </style>
</head>
<body>
 <div class="box">This is an example of the box model.</div>
</body>
</html>
```

Example with Fluid Layout:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Fluid Layout Example</title>
 <style>
  .container {
   width: 80%;
   margin: 0 auto;
   background-color: lightgray;
   padding: 20px;
  }

  .responsive-box {
   width: 50vw;
   height: 50vh;
```

```
      background-color: lightgreen;
      margin: 20px 0;
    }

    .flex-container {
      display: flex;
      flex-wrap: wrap;
      justify-content: space-around;
    }

    .flex-item {
      flex: 1 1 200px;
      margin: 10px;
      background-color: lightcoral;
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Fluid Layout Example</h1>
    <div class="responsive-box">Responsive Box</div>
    <div class="flex-container">
      <div class="flex-item">Flex Item 1</div>
      <div class="flex-item">Flex Item 2</div>
      <div class="flex-item">Flex Item 3</div>
    </div>
  </div>
</body>
</html>
```

Day 11: CSS Layouts

Date: 18-06-24

---

1. CSS Layout Basics:

- **Block Layout:**
  - Block-level elements occupy the full width of their container and start on a new line.
  - Examples: <div>, <p>, <h1>, <section>
  - Properties:
-
- div {
-   display: block;
-   width: 100%;
- }

 Inline Layout:

- Inline elements do not start on a new line and only occupy as much width as necessary.
- Examples: <span>, <a>, <strong>
- Properties:
-
- a {
-   display: inline;
- }

 Inline-Block Layout:

- Inline-block elements are similar to inline elements but can have width and height set.
- Examples: <img>, <button>
- Properties:
-
  - .inline-block {
  -   display: inline-block;
  -   width: 100px;
  -   height: 50px;

- }
-

2. Modern Layout Techniques:

- **Flexbox:**
  - Flexbox is designed for one-dimensional layouts. It allows items to align and distribute space within a container.
  - Properties:

```
 .flex-container {
  display: flex;
  justify-content: space-between; /* Align items horizontally */
  align-items: center; /* Align items vertically */
}
.flex-item {
  flex: 1; /* Grow items to fill available space */
  margin: 10px;
}
```

 Example:

-
- <style>
- .flex-container {
-   display: flex;
-   justify-content: space-between;
-   align-items: center;
-   background-color: lightgray;
-   padding: 20px;
- }
- .flex-item {
-   flex: 1;
-   margin: 10px;
-   background-color: lightcoral;
-   text-align: center;

- padding: 20px;
- }
- </style>
- <div class="flex-container">
- <div class="flex-item">Item 1</div>
- <div class="flex-item">Item 2</div>
- <div class="flex-item">Item 3</div>
- </div>

 CSS Grid:

- CSS Grid is a two-dimensional layout system that allows for both rows and columns.
- Properties:

 .grid-container {

```
  display: grid;
  grid-template-columns: repeat(3, 1fr); /* Three equal columns */
  grid-gap: 10px; /* Gap between items */
}
.grid-item {
  background-color: lightblue;
  text-align: center;
  padding: 20px;
}
```

 Example:

-
    - <style>
    - .grid-container {
    - display: grid;
    - grid-template-columns: repeat(3, 1fr);
    - grid-gap: 10px;
    - }
    - .grid-item {
    - background-color: lightblue;

    - text-align: center;
    - padding: 20px;
    - }
    - </style>
    - <div class="grid-container">
    - <div class="grid-item">Item 1</div>
    - <div class="grid-item">Item 2</div>
    - <div class="grid-item">Item 3</div>
    - <div class="grid-item">Item 4</div>
    - <div class="grid-item">Item 5</div>
    - <div class="grid-item">Item 6</div>
    - </div>
    -

3. Positioning Techniques:

- **Static Positioning:**
    - Default positioning of elements.
    - Example:
-
- .static {
- position: static;
- }

 Relative Positioning:

- Positioned relative to its normal position.
- Example:
-
- .relative {
- position: relative;
- top: 10px;
- left: 20px;
- }

 Absolute Positioning:

- Positioned relative to its nearest positioned ancestor.
- Example:
- □
- .absolute {
- position: absolute;
- top: 50px;
- left: 50px;
- }

□ **Fixed Positioning:**

- Positioned relative to the browser window.
- Example:
- □
- .fixed {
- position: fixed;
- bottom: 0;
- width: 100%;
- background-color: lightgray;
- }

□ **Sticky Positioning:**

- Switches between relative and fixed positioning based on the user's scroll position.
- Example:
-
    - .sticky {
    - position: -webkit-sticky; /* For Safari */
    - position: sticky;
    - top: 0;
    - background-color: yellow;
    - }
    -

Day 19-06-24

1. Introduction to Flexbox:

- Flexbox is designed for one-dimensional layouts, either in a row or a column.
- It consists of a flex container and flex items.

2. Flex Container Properties:

- **display: flex;** Defines a flex container and enables flex context for all its direct children.

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

 flex-direction: Specifies the direction of the flex items.

```
 .flex-container {
  flex-direction: row; /* Default */
}
/* Other values: row-reverse, column, column-reverse */
```

 flex-wrap: Determines whether flex items should wrap or not.

```
 .flex-container {
  flex-wrap: nowrap; /* Default */
}
/* Other values: wrap, wrap-reverse */
```

 flex-flow: A shorthand for setting both flex-direction and flex-wrap.

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

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

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

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

```
 .flex-container {
  align-items: stretch; /* Default */
```

}

/* Other values: flex-start, flex-end, center, baseline */

 align-content: Aligns flex lines when there is extra space in the cross axis.

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

3. Flex Item Properties:

- **order:** Controls the order of the flex items.

 .flex-item {

  order: 1; /* Default is 0 */

}

 flex-grow: Specifies how much a flex item will grow relative to the rest.

 .flex-item {

  flex-grow: 1; /* Default is 0 */

}

 flex-shrink: Specifies how much a flex item will shrink relative to the rest.

 .flex-item {

  flex-shrink: 1; /* Default */

}

 flex-basis: Defines the initial size of a flex item.

 .flex-item {

  flex-basis: 100px; /* Default is auto */

}

 flex: A shorthand for flex-grow, flex-shrink, and flex-basis.

 .flex-item {

  flex: 1 1 100px;

}

 align-self: Allows the default alignment (or the one specified by align-items) to be overridden for individual flex items.

- .flex-item {
-   align-self: auto; /* Default */
- }
- /* Other values: flex-start, flex-end, center, baseline, stretch */
-

Example with Flex Properties:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox Properties</title>
  <style>
    .flex-container {
      display: flex;
      flex-direction: column;
      flex-wrap: wrap;
      justify-content: center;
      align-items: flex-start;
      align-content: space-between;
      height: 300px;
      background-color: lightblue;
    }

    .flex-item {
      background-color: lightgreen;
      margin: 10px;
      padding: 20px;
      text-align: center;
      order: 2;
      flex: 1 1 100px;
      align-self: center;
    }

    .flex-item:first-child {
      order: 1;
      flex: 2 1 150px;
    }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">Item 1</div>
    <div class="flex-item">Item 2</div>
    <div class="flex-item">Item 3</div>
    <div class="flex-item">Item 4</div>
  </div>
</body>
</html>
```