10 Exciting CSS Learning Exercises

Practice Coding and learn more about applying CSS to your HTML code

Exercise 1: Basic Styling

Objective: Apply basic styling to an HTML document.

Steps:

1. Create an HTML file with a sample structure.
2. Link a CSS file to the HTML document.
3. Apply basic styling such as setting the background color, font size, and text color.

HTML:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width,
initial-scale=1.0">

  <link rel="stylesheet" href="styles.css">

  <title>Exercise 1</title>

</head>

<body>

  <h1>Hello, CSS!</h1>

  <p>This is a sample paragraph.</p>

</body>

</html>
```

CSS (styles.css):

```
body {

  background-color: #f0f0f0;
```

```
  font-size: 16px;
  color: #333;
}

h1 {
  color: #0066cc;
}
```

Exercise 2: Box Model

Objective: Understand and apply the CSS box model.

Steps:

1. Create an HTML file with a container div and several child elements.
2. Apply borders, margins, and padding to different elements.
3. Observe how the box model affects the layout.

HTML:

```
<!-- Same as Exercise 1 -->
```

CSS (styles.css):

```
.container {
  border: 1px solid #999;
```

```
  padding: 20px;
}

.child {
  margin: 10px;
  padding: 15px;
  border: 1px solid #ddd;
}
```

Exercise 3: Flexbox

Objective: Learn and implement Flexbox layout.

Steps:

1. Create a container with several child elements.
2. Use Flexbox to arrange the child elements horizontally.
3. Experiment with Flexbox properties like justify-content and align-items.

HTML:

```
<!-- Same as Exercise 1 -->
```

CSS (styles.css):

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

.child {
  flex: 1;
}
```

Exercise 4: Grid Layout

Objective: Learn and implement CSS Grid layout.

Steps:

1. Create a container with multiple child elements.
2. Apply CSS Grid to create a two-dimensional layout.
3. Experiment with properties like grid-template-columns and grid-gap.

HTML:

```
<!-- Same as Exercise 1 -->
```

CSS (styles.css):

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 20px;
}
```

Exercise 5: Responsive Design

Objective: Make your webpage responsive.

Steps:

1. Set up a simple webpage with a navigation bar and content.
2. Use media queries to adjust the layout for different screen sizes.
3. Test responsiveness by resizing the browser window.

HTML:

```
<!-- Same as Exercise 1 with added navigation bar -->
```

CSS (styles.css):

```
/* Same as Exercise 1 styles */
nav {
  background-color: #333;
  color: #fff;
```

```
  padding: 10px;
}

@media (max-width: 600px) {
  /* Adjust styles for small screens */
  nav {
    text-align: center;
  }
}
```

Exercise 6: CSS Transitions and Animations

Objective: Implement CSS transitions and animations.

Steps:

1. Create an HTML element (e.g., a button).
2. Apply styles to make the element visually appealing.
3. Use CSS transitions to create smooth hover effects.
4. Implement a simple CSS animation (e.g., rotation) on click.

HTML:

```
<!-- Same as Exercise 1 with added button -->
```

```
<button class="animated-button">Click me</button>
```

CSS (styles.css):

```
/* Same as Exercise 1 styles */


.animated-button {
  background-color: #4CAF50;
  color: #fff;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s ease-in-out, transform 0.3s ease-in-out;
}


.animated-button:hover {
  background-color: #45a049;
}
```

```
.animated-button:active {

  transform: rotate(360deg);

}
```

Exercise 7: CSS Variables

Objective: Use CSS variables to create a theme.

Steps:

1. Define CSS variables for colors, fonts, etc.
2. Apply these variables throughout your stylesheet.
3. Experiment with changing the theme by modifying variable values.

HTML:

```
<!-- Same as Exercise 1 -->
```

CSS (styles.css):

```
:root {

  --primary-color: #3498db;

  --secondary-color: #2ecc71;

}
```

```
body {
  background-color: var(--secondary-color);
  color: #fff;
}

h1 {
  color: var(--primary-color);
}
```

Exercise 8: Pseudo-classes and Pseudo-elements

Objective: Use pseudo-classes and pseudo-elements for styling.

Steps:

1. Apply styles to different states of a button using pseudo-classes (e.g., :hover).
2. Use pseudo-elements (e.g., ::before or ::after) to add decorative elements to an existing element.

HTML:

```
<!-- Same as Exercise 1 with additional button -->
```

```
<button class="styled-button">Hover me</button>
```

CSS (styles.css):

```
/* Same as Exercise 1 styles */

.styled-button {
  background-color: #e74c3c;
  color: #fff;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

.styled-button:hover {
  background-color: #c0392b;
}

.styled-button::before {
```

```
  content: '\2022'; /* Unicode for a bullet point */
  margin-right: 10px;
}
```

Exercise 9: Form Styling

Objective: Style HTML forms.

Steps:

1. Create an HTML form with various input types (text, checkbox, radio, etc.).
2. Apply styles to improve the form's visual appearance.
3. Use pseudo-classes for styling different states of form elements.

HTML:

```
<!-- Same as Exercise 1 with added form -->
<form>
  <label for="username">Username:</label>
  <input type="text" id="username" name="username">
  <br>
  <label for="password">Password:</label>
  <input type="password" id="password" name="password">
  <br>
```

```
  <input type="checkbox" id="remember" name="remember">
  <label for="remember">Remember me</label>
  <br>
  <input type="submit" value="Submit">
</form>
```

CSS (styles.css):

```
/* Same as Exercise 1 styles */

form {
  max-width: 400px;
  margin: 0 auto;
}

label {
  display: block;
  margin-bottom: 5px;
}
```

```
input {
  width: 100%;
  padding: 8px;
  margin-bottom: 10px;
}

input[type="submit"] {
  background-color: #2ecc71;
  color: #fff;
  cursor: pointer;
}

input[type="submit"]:hover {
  background-color: #27ae60;
}
```

Exercise 10: CSS Positioning

Objective: Practice different CSS positioning techniques.

Steps:

1. Create an HTML layout with multiple elements.
2. Experiment with different positioning values such as relative, absolute, and fixed.
3. Use the z-index property to control the stacking order of elements.

HTML:

```
<!-- Same as Exercise 1 with additional positioned elements -->

<div class="positioned-container">

  <div class="positioned-box">Box 1</div>

  <div class="positioned-box">Box 2</div>

</div>
```

CSS (styles.css):

```
/* Same as Exercise 1 styles */


.positioned-container {

  position: relative;

}


.positioned-box {
```

```
  position: absolute;
  top: 20px;
  left: 20px;
  width: 100px;
  height: 100px;
  background-color: #3498db;
  color: #fff;
  text-align: center;
  line-height: 100px;
  margin: 10px;
}

.positioned-box:nth-child(2) {
  top: 70px;
  left: 70px;
  background-color: #e74c3c;
}
```

These exercises cover a broad range of CSS concepts and should help you strengthen your skills.