Web development

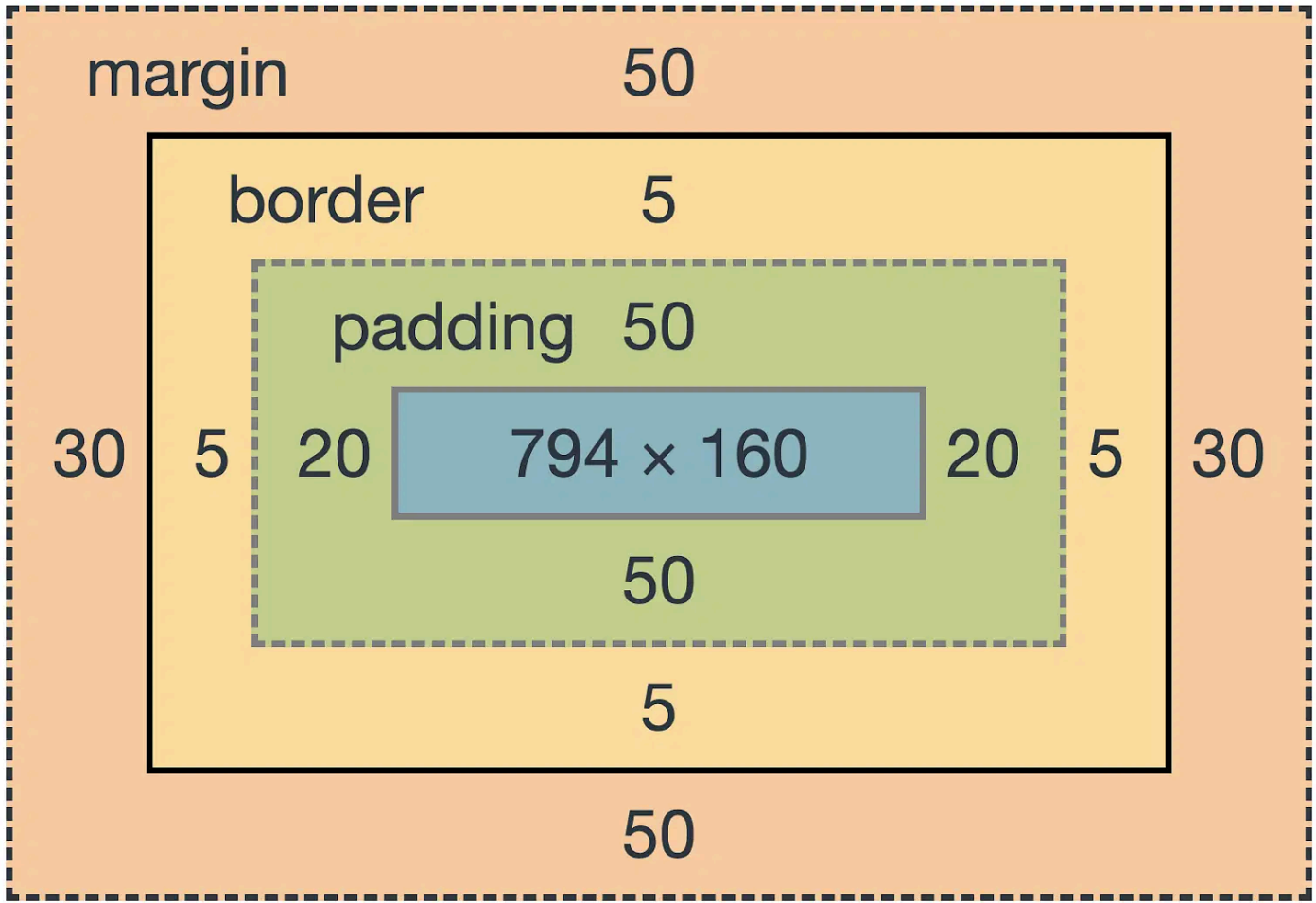
lesson 13

Nov 24, 2024

CSS Box Model: Introduction

What Is the CSS Box Model?

- Every element in CSS is a rectangular box that consists of the following areas:
  1. **Content**: The actual content of the element (e.g., text, images).
  2. **Padding**: Space between the content and the border.
  3. **Border**: A line surrounding the padding (or content if no padding).
  4. **Margin**: Space between the border and neighboring elements.


margin
50
border
5
padding
50
30
5
20
794 × 160
20
5
30
50
5
50

Content Area

What Is the Content Area?

- The innermost part of the box containing text, images, or other content.
- Defined by properties like `width` and `height`.

Example:

```
div {
  width: 200px;
  height: 100px;
  background-color: lightblue;
}
```

Note:

- The total size of the element is affected by padding, border, and margin unless the `box-sizing` property is used.

Padding

What Is Padding?

- The space between the content and the border.
- Adds internal spacing inside the element, expanding its visual size.

Example:

```
div {
  padding: 20px;
  background-color: lightcoral;
}
```

Shorthand for Padding:

- `padding: 10px;` (all sides).
- `padding: 10px 20px;` (top/bottom, left/right).
- `padding: 10px 20px 15px;` (top, left/right, bottom).
- `padding: 10px 20px 15px 5px;` (top, right, bottom, left).

Borders

What Is the Border?

- The line surrounding the padding or content of an element.
- Can have style, width, and color.

Example:

```
div {
  border: 2px solid black;
}
```

Border Shorthand:

- `border: 2px dashed red;`

Border Radius:

- Add rounded corners with `border-radius`.
- Example: `border-radius: 10px;`

Margin

What Is the Margin?

- The space outside the border that separates an element from its neighbors.
- Does not affect the element's size.

Example:

```
div {
  margin: 20px;
}
```

Shorthand for Margin:

- `margin: 10px;` (all sides).
- `margin: 10px 20px;` (top/bottom, left/right).
- `margin: 10px 20px 15px;` (top, left/right, bottom).
- `margin: 10px 20px 15px 5px;` (top, right, bottom, left).

Practical Example: Box Model in Action

Combining Margin, Padding, and Border:

- Example:

```
div {
  width: 200px;
  height: 100px;
  margin: 20px;
  padding: 10px;
  border: 5px solid blue;
  background-color: lightgreen;
}
```

Visual Breakdown:

- Content: 200px x 100px.
- Padding: Adds 10px inside the border.
- Border: 5px width, blue color.
- Margin: Adds 20px of space outside the element.

Summary and Best Practices

Key Points:

- The box model defines how element dimensions and spacing are calculated.
- Padding adds space inside the element; margin adds space outside.
- Border surrounds the padding and content.

Tips:

- Use `box-sizing: border-box;` to include padding and border in the total width/height calculation.
- Visualize the box model using browser dev tools for debugging.
- Use consistent spacing for better design.

CSS Animations: Introduction

What Are CSS Animations?

- CSS animations allow you to create smooth transitions between states without JavaScript.
- Two main components:
  1. **@keyframes**: Defines the animation behavior.
  2. **Animation Properties**: Control how the animation is applied.

Why Use CSS Animations?

- Improve user experience (e.g., loading indicators, hover effects).
- Enhance visual appeal with smooth transitions.

Example:

```
@keyframes slideIn {
  from {
    transform: translateX(-100%);
  }
  to {
    transform: translateX(0);
  }
}
```

Using `@keyframes`

Defining Keyframes:

- The `@keyframes` rule defines the intermediate steps of the animation.
- Use `from` and `to` or percentage values.

Example:

```
@keyframes fadeIn {
  0% {
    opacity: 0;
  }
  100% {
    opacity: 1;
  }
}
```

How It Works:

- `0%` marks the starting point.
- `100%` marks the endpoint.
- Intermediate steps (e.g., `50%`) can be defined for more control.

Animation Properties

Key Animation Properties:

1. `animation-name`: Specifies the name of the `@keyframes`.
2. `animation-duration`: Duration of the animation.
3. `animation-timing-function`: Speed curve of the animation.
4. `animation-delay`: Time before the animation starts.

Example:

```
div {
  animation-name: fadeIn;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: 1s;
}
```

Animation Timing Functions

Controlling Animation Speed:

- Predefined values: `linear`, `ease`, `ease-in`, `ease-out`, `ease-in-out`.
- Custom speed curves with `cubic-bezier()`.

Example:

```
div {
  animation-timing-function: cubic-bezier(0.42, 0, 0.58, 1);
}
```

Visualizing Timing Functions:

- Use tools like Cubic Bezier Generator to design curves.
- Default `ease` starts slow, accelerates, then slows down.

Combining Multiple Properties

Shorthand for Animations:

- Use `animation` shorthand to combine all animation properties.
- Syntax: `animation: name duration timing-function delay iteration-count direction;`

Example:

```
div {
  animation: fadeIn 2s ease-in-out 1s infinite alternate;
}
```

- `infinite` : Animation loops indefinitely.
- `alternate` : Reverses direction on every loop.

Practical Example: Bouncing Ball

Code Example:

```
@keyframes bounce {
  0%, 100% {
    transform: translateY(0);
  }
  50% {
    transform: translateY(-50px);
  }
}

.ball {
  animation: bounce 1s ease-in-out infinite;
  width: 50px;
  height: 50px;
  background-color: red;
  border-radius: 50%;
}
```

Practical Example: Bouncing Ball 2

HTML:

```
<div class="ball"></div>
```

What Happens:

- The ball moves up and down in a continuous loop, creating a bouncing effect.

Summary and Best Practices

Key Takeaways:

- Use `@keyframes` to define the animation steps.
- Combine animation properties with the `animation` shorthand for simplicity.
- Use timing functions to create smooth and natural effects.

Best Practices:

- Avoid excessive animations that may harm user experience.
- Test performance, especially on mobile devices.
- Use animations to guide user focus and improve usability.

CSS for Mobile Development: Introduction

What Is Mobile Development in CSS?

- CSS techniques used to create responsive, optimized, and touch-friendly designs for mobile devices.

Why It Matters:

- Over 50% of web traffic comes from mobile devices.
- Ensures usability and performance across various screen sizes and resolutions.

Key Focus Areas:

1. Responsive design.
2. Media queries.
3. Flexible layouts.
4. Performance optimization.

Responsive Design Basics

Responsive Design:

- Adapts the layout to different screen sizes and devices.
- Achieved through flexible grids, images, and media queries.

Core Concepts:

1. **Fluid Grids:** Use relative units like `%`, `vw`, and `vh` for layout elements.
2. **Flexible Images:** Set images to scale with the layout using `max-width: 100%;`.
3. Viewport Meta Tag:

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

Example:

```
.container {
  width: 90%;
  max-width: 1200px;
  margin: 0 auto;
}
```

Media Queries

What Are Media Queries?

- Enable conditional CSS based on device characteristics, like screen size or orientation.

Syntax:

```
@media (max-width: 768px) {
  body {
    font-size: 14px;
  }
}
```

- `max-width` : Targets screens smaller than or equal to the specified width.
- `min-width` : Targets screens larger than or equal to the specified width.

Media Queries 2

Common Breakpoints:

- Mobile: `480px`
- Tablets: `768px`
- Desktops: `1024px`

Best Practices:

- Start with a mobile-first approach, using `min-width` to add styles for larger devices.

Flexible Units for Mobile Design

Why Flexible Units Matter:

- Mobile devices have varying screen sizes and resolutions.
- Avoid fixed units (e.g., `px`) in favor of flexible ones.

Common Units:

1. `%` for relative sizing (e.g., width and height).
2. `em` and `rem` for scalable typography.
3. `vw` and `vh` for viewport-relative sizing.

- Example: `width: 50vw;` is 50% of the viewport width.

Using `clamp()` for Adaptive Sizing:

```
h1 {
  font-size: clamp(1rem, 5vw, 2rem);
}
```

Mobile-First Layouts

Key Principles:

- Start designing for small screens first and scale up for larger devices.
- Use simple layouts with vertical stacking for smaller screens.

CSS Techniques:

- Use flexbox for responsive layouts:

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

Mobile-First Layouts 2

- Grid layout for more complex designs:

```
.grid {
  display: grid;
  grid-template-columns: 1fr;
}
@media (min-width: 768px) {
  .grid {
    grid-template-columns: 1fr 1fr;
  }
}
```