Tag 3 CSS: Das Box-Modell, Display und Positionierung

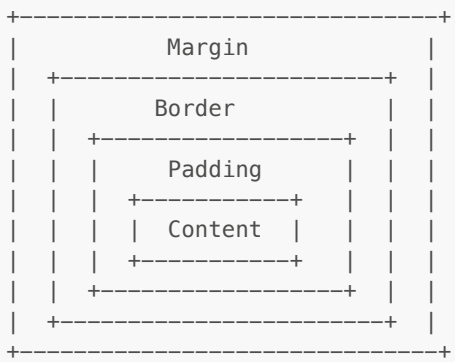
Das CSS-Box-Modell ist eines der wichtigsten Konzepte für Layout und Design im Web. Es beschreibt, wie der Platz und die Größe von Elementen berechnet wird. In diesem Kapitel behandeln wir alle Bestandteile dieses Modells, deren Zusammenspiel, moderne Best Practices und nützliche Erweiterungen wie Flexbox und `box-sizing`.

---

1. Das CSS Box-Modell – Grundstruktur

Jedes HTML-Element besteht im Layout aus vier Bereichen:

1. **Content** – Der Inhalt (z. B. Text oder Bild)
2. **Padding** – Der Innenabstand zwischen Inhalt und Rahmen
3. **Border** – Die sichtbare Umrandung
4. **Margin** – Der äußere Abstand zu anderen Elementen



---

2. `box-sizing`: `content-box` vs `border-box`

`content-box` (Standard)

```css
.box {
  width: 300px;
  padding: 20px;
  border: 10px solid black;
  margin: 10px;
  box-sizing: content-box;
}
```

> Ergebnis: Die tatsächliche Breite ist 300 + 20*2 (Padding)* + *10*2 (Border) = **360px**

`border-box` (Empfohlen)

```css
.box {
  width: 300px;
  padding: 20px;
  border: 10px solid black;
  box-sizing: border-box;
}
```

> Ergebnis: Die Gesamtbreite **bleibt bei 300px** – Padding und Border werden innerhalb eingerechnet.

> ✅ **Best Practice:** `box-sizing: border-box;` ist moderner Standard. Nutze am besten ein globales Reset:

```
*, *::before, *::after {
  box-sizing: border-box;
}
```

---

3. Margin und Padding

Margin – Außenabstand

```
.box {
  margin: 20px; /* alle Seiten gleich */
  margin-top: 30px;
  margin-left: 10px;
}
```

> Margins können auch negativ sein, was Elemente überlappen lässt.

Wirkung auf Inline-Elemente:

- Bei **inline-Elementen** (z. B. `<span>`, `<a>`) wirkt sich `margin` **horizontal** aus (`margin-left` & `margin-right`).
- Vertikale Margins (`margin-top` & `margin-bottom`) werden zwar akzeptiert, aber haben **keinen Einfluss** auf das Layout.
- Um Margin vollständig zu nutzen, kann man inline-Elemente mit `display: inline-block` behandeln.

Padding – Innenabstand

```
.box {
  padding: 15px;
  padding-right: 30px;
}
```

> Padding wirkt sich **innerhalb** der Box aus und vergrößert visuell den Raum um den Inhalt.

Wirkung auf Inline-Elemente:

- `padding-left` und `padding-right` wirken sich wie erwartet aus – sie erweitern die "Box" des Textes.
- `padding-top` und `padding-bottom` sind zwar definierbar, wirken aber **nicht immer sichtbar**, da Inline-Elemente sich nach der Textlinie richten.
- Auch hier kann `display: inline-block` helfen, um Padding in alle Richtungen sichtbar zu machen.

---

4. Border – Rahmen um das Element

```
.box {
  border: 5px solid black;
}
```

Stile:

- `solid`
- `dashed`
- `dotted`
- `double`
- `none`

Farben und Breiten sind frei kombinierbar:

```
```

```
p {
  border: 2px dashed red;
}
```

---

5. Display-Typen

`block`

```
div {
  display: block;
}
```

> Nimmt volle Breite ein, beginnt in neuer Zeile

`inline`

```
span {
  display: inline;
}
```

> Nimmt nur benötigte Breite ein, kein `width/height` möglich

`inline-block`

```
div {
  display: inline-block;
}
```

> Verhalten wie inline, erlaubt aber Größenangaben

`none`

```
.element {
  display: none;
}
```

> Element wird komplett ausgeblendet

---

6. Ausrichtung (Alignment)

Vertikal (inline-Elemente)

```
img {
  vertical-align: middle;
}
```

Horizontal (Text)

```
div {
  text-align: center;
```

```
}
```

Modern: Flexbox

```
.container {
  display: flex;
  justify-content: center;     /* horizontal */
  align-items: center;         /* vertikal */
  height: 100vh;
}
```

> **Flexbox** wird heute als Standardlösung für zentrierte Ausrichtung verwendet

---

7. Positionierung

Die `position`-Eigenschaft legt fest, **wie ein Element im Dokument positioniert** wird. Es gibt 5 Hauptwerte:

`static` (Standardwert)

```
div {
  position: static;
}
```

- Elemente erscheinen in der normalen Reihenfolge des HTML-Dokuments.
- Es kann kein Versatz über `top`, `left`, etc. erfolgen.

`relative`

```
div {
  position: relative;
  top: 10px;
  left: 20px;
}
```

- Element bleibt im Fluss, wird aber **relativ zu seiner normalen Position** verschoben.
- Umgebende Elemente nehmen **weiterhin seinen ursprünglichen Platz** ein.

> Praktisch für kleine Justierungen oder als Ausgangspunkt für `absolute` Positionierung von Kind-Elementen.

`absolute`

```
.outer {
  position: relative;
}

.inner {
  position: absolute;
  top: 0;
  right: 0;
}
```

- Das Element **verlässt den Fluss**.
- Positioniert sich relativ zum nächsten **positionierten Vorfahren** (d. h. einem Elternelement mit `relative`, `absolute`, `fixed` oder `sticky`).

> Häufige Fehlerquelle: Wenn kein Vorfahr positioniert ist, bezieht es sich auf das `<body>` oder `<html>`.

`fixed`

```
.banner {
  position: fixed;
  bottom: 0;
  width: 100%;
  background: yellow;
}
```

- Positioniert relativ zum **Viewport**.
- Bleibt beim Scrollen **immer sichtbar**.

> Praktisch für Navigationsleisten oder "Zurück nach oben"-Buttons.

`sticky`

```
header {
  position: sticky;
  top: 0;
  background: white;
  z-index: 100;
}
```

- Beginnt als `relative` – wird `fixed`, sobald eine Scrollposition erreicht ist.
- Sehr gut für "Sticky Headers" oder seitliche Navigationsleisten.

> Voraussetzung: Ein übergeordneter Container darf kein `overflow: hidden` haben.

8. Übungsaufgaben – Box-Modell, Display & Positionierung

Aufgabe 1: Box-Modell visualisieren

Erstelle eine Box mit folgenden Eigenschaften:

- `width: 300px`, `height: 150px`
- `padding: 20px`, `border: 5px solid black`, `margin: 30px`
- Verwende `box-sizing: content-box`
- Berechne die tatsächliche Gesamtbreite und Höhe

Aufgabe 2: Border-Stile testen

Erstelle vier `<div>`-Boxen mit verschiedenen Border-Stilen: `solid`, `dashed`, `dotted`, `double`. Teste jeweils Farbe und Breite.

Aufgabe 3: Display-Typen vergleichen

Erstelle drei `<div>`-Elemente mit Text und setze jeweils `display: block`, `inline` und `inline-block`. Vergleiche das Verhalten.

Aufgabe 4: Inline vs. Inline-Block

Füge mehreren `<span>`-Elementen Hintergründe und Breiten hinzu. Wechsle zwischen `display: inline` und `inline-block` und beobachte den Unterschied.

Aufgabe 5: Flexbox-Zentrierung

Erstelle einen Container mit `display: flex` und zentriere ein Kind-Element sowohl horizontal als auch vertikal (mit `justify-content` und `align-items`).

Aufgabe 6: Positionierung anwenden

Erstelle eine Seite mit vier Boxen und folgenden Positionierungen:

- Eine Box mit `static`
- Eine mit `relative`, die etwas verschoben wird
- Eine mit `absolute`, die innerhalb eines Containers oben rechts sitzt
- Eine mit `fixed`, die unten rechts am Bildschirm klebt

Aufgabe 7: Sticky Navigation

Erstelle ein Header-Element mit `position: sticky; top: 0`, das beim Scrollen oben sichtbar bleibt. Teste, ob es korrekt funktioniert.