Tag 4 CSS: Flexbox

Flexbox ist ein modernes CSS-Layoutmodell zur flexiblen Gestaltung von Benutzeroberflächen. Im Gegensatz zu älteren Techniken wie Floats bietet es eine intuitive Steuerung der Ausrichtung und Größenverteilung von Elementen.

---

1. Einführung in Flexbox

Flexbox (Flexible Box Layout) ist speziell dafür entwickelt worden, Container-Inhalte dynamisch und effizient auszurichten, sowohl horizontal als auch vertikal. Es eignet sich besonders gut für:

- Navigationen
- Karten (Cards)
- Footer-Layouts
- Zentrierungen
- Responsives Verhalten

---

2. Flexbox Grundlagen

Flexbox besteht aus:

- **Flex-Container**: das Elternelement mit `display: flex`
- **Flex-Items**: direkte Kindelemente im Flex-Kontext

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

---

3. Flex-Container Eigenschaften

a) Achsen definieren mit `flex-direction`

Die Eigenschaft `flex-direction` bestimmt, in welche Richtung die Flex-Items im Container angeordnet werden. Sie legt die Hauptachse fest, entlang derer die Items ausgerichtet werden.

```
.container {
  display: flex;
  flex-direction: row; /* oder column, row-reverse, column-reverse */
}
```

- `row` **(Standard)**: Hauptachse von links nach rechts
- `row-reverse`: Hauptachse von rechts nach links
- `column`: Hauptachse von oben nach unten (vertikale Ausrichtung)
- `column-reverse`: Hauptachse von unten nach oben
- **Hauptachse (main axis)**: bestimmt durch `flex-direction`
- **Querachse (cross axis)**: senkrecht zur Hauptachse

---

b) Ausrichtung entlang der Hauptachse mit `justify-content`

`justify-content` steuert die Verteilung der Flex-Items **entlang der Hauptachse** (je nach `flex-direction` horizontal oder vertikal).

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

Mögliche Werte:

- `flex-start`: Elemente beginnen am Start der Hauptachse
- `flex-end`: Elemente enden am Ende der Hauptachse
- `center`: Zentrierung entlang der Hauptachse
- `space-between`: Gleichmäßiger Abstand zwischen den Items
- `space-around`: Gleichmäßiger Abstand um jedes Item (auch außen)
- `space-evenly`: Gleicher Abstand zwischen allen Elementen inkl. Rand

> `justify-content` ist nützlich für horizontale Menüs oder das Verteilen von Boxen in einer Zeile.

---

c) Ausrichtung entlang der Querachse mit `align-items`

`align-items` steuert die vertikale Ausrichtung (bei `flex-direction: row`) oder die horizontale Ausrichtung (bei `flex-direction: column`) **entlang der Querachse**.

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

Mögliche Werte:

- `stretch`: Elemente füllen die Containerhöhe/breite aus (Standard)
- `flex-start`: Ausrichtung am Anfang der Querachse
- `flex-end`: Ausrichtung am Ende der Querachse
- `center`: Zentrierung entlang der Querachse
- `baseline`: Ausrichtung an der Text-Basislinie der Items

> Sehr hilfreich beim Zentrieren von Elementen in Cards oder Boxen.

---

d) Umbruch mit `flex-wrap`

Standardmäßig versucht Flexbox, alle Items in einer Zeile unterzubringen. Mit `flex-wrap` kann dieses Verhalten geändert werden.

```css
.container {
  flex-wrap: wrap;
}
```

Mögliche Werte:

- `nowrap`: Alles bleibt in einer Zeile (Standard)
- `wrap`: Elemente umbrechen in eine neue Zeile, wenn kein Platz mehr vorhanden ist
- `wrap-reverse`: Umbruch wie bei `wrap`, aber neue Zeile erscheint **oberhalb** der alten

> Flex-Wrap ist wichtig für responsives Verhalten: es verhindert, dass Items aus dem Container ragen.

---

e) Shorthand: `flex-flow`

Die Eigenschaft `flex-flow` kombiniert `flex-direction` und `flex-wrap` zu einer Kurzschreibweise.

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

```
}
```

Syntax:

```
flex-flow: <flex-direction> <flex-wrap>;
```

Beispiele:

- `flex-flow: column nowrap;`
- `flex-flow: row-reverse wrap;`

> Nützlich für schlanken Code in komplexeren Layouts.

---

4. Flex-Item Eigenschaften

Die Flex-Items sind die direkten Kindelemente eines Flex-Containers. Sie verhalten sich innerhalb des Flex-Layouts nach bestimmten Regeln und lassen sich individuell mit folgenden Eigenschaften steuern:

---

a) `flex-grow`

Diese Eigenschaft bestimmt, wie stark ein Item im Vergleich zu anderen wachsen kann, **wenn zusätzlicher Platz im Container vorhanden ist**.

```
.item {
  flex-grow: 1;
}
```

- Der Wert ist eine relative Zahl.
- Wenn ein Item `flex-grow: 2` hat und ein anderes `flex-grow: 1`, erhält das erste **doppelt so viel** freien Platz.
- Standardwert ist `0` → Item wächst **nicht** über seine Basisgröße hinaus.

> Beispiel: Drei Items mit `flex-grow: 1`, `1`, `2` → Verteilung: 25%, 25%, 50% des verbleibenden Raums

---

b) `flex-shrink`

Steuert, wie stark ein Item schrumpfen darf, **wenn nicht genügend Platz im Container vorhanden ist**.

```
.item {
  flex-shrink: 1;
}
```

- Standardwert: `1`
- Ein Wert von `0` bedeutet: Das Element **darf nicht schrumpfen**
- Funktioniert ähnlich wie `flex-grow`, nur im Negativfall (zu wenig Platz)

> Tipp: Bei responsiven Designs gezielt einsetzen, um wichtige Inhalte zu schützen

---

c) `flex-basis`

Bestimmt die **Startgröße** eines Flex-Items, **bevor** `flex-grow` oder `flex-shrink` angewendet werden.

```
.item {
  flex-basis: 200px;
}
```

- Kann in `px`, `%`, `em` usw. angegeben werden
- Ersetzt `width` im Flex-Kontext
- Standardwert: `auto`

> Wird `width` UND `flex-basis` gesetzt, hat `flex-basis` Vorrang.

---

d) `flex` (Shorthand)

Kombiniert die drei oben genannten Eigenschaften:

```
.item {
  flex: 1 1 200px; /* grow shrink basis */
}
```

Beispiele:

- `flex: 1;` → Entspricht `1 1 0%`
- `flex: 0 1 auto;` → Standardwert

> Verwende `flex: 1` für gleichmäßig flexible Elemente ohne feste Basisgröße

---

e) `align-self`

Überschreibt die `align-items`-Ausrichtung **nur für ein einzelnes Flex-Item**.

```
.item {
  align-self: flex-end;
}
```

Werte:

- `auto` (Standard): übernimmt Wert von `align-items`
- `flex-start`, `flex-end`, `center`, `baseline`, `stretch`

> Anwendungsfall: Ein Button, der am Ende einer Card ausgerichtet sein soll, unabhängig von anderen Items

---

f) Sichtbarkeit & Reihenfolge kombinieren

Auch Eigenschaften wie `order`, `visibility`, `display: none` oder `margin` beeinflussen das Verhalten von Flex-Items:

```
.item {
  order: 2;
  margin-right: auto; /* schiebt nach links */
  visibility: hidden;
}
```

> Flexbox funktioniert hervorragend mit dynamischen Interfaces (Tabs, Navigationen, Grids)

📌 **Zusammenfassung – Wann welche Eigenschaft?**

Ziel	Eigenschaft(en)
Gleichmäßige Verteilung	`flex: 1`
Feste Startbreite, aber flexibel	`flex: 1 1 200px`

Ziel	Eigenschaft(en)
Kein Schrumpfen	`flex-shrink: 0`
Eigene vertikale Ausrichtung	`align-self`
Reihenfolge ändern	`order`
Unterschiedliche Wachstumsverteilung	`flex-grow`

---

5. Weitere wichtige Eigenschaften und Konzepte

Neben den klassischen Container- und Item-Eigenschaften gibt es zwei zentrale Erweiterungen, die Flexbox noch mächtiger machen: `order` zur individuellen Reihenfolge und `align-content` für mehrzeilige Ausrichtung.

---

a) `order` – Reihenfolge der Flex-Items verändern

Mit der `order`-Eigenschaft kann die visuelle Reihenfolge von Flex-Items unabhängig von der **HTML-Reihenfolge** gesteuert werden. Standardmäßig haben alle Items `order: 0`. Je niedriger der Wert, desto früher erscheint das Element.

```
.item1 {
  order: 2;
}
.item2 {
  order: 1;
}
.item3 {
  order: 3;
}
```

> Ergebnis: Im Layout erscheint `.item2` zuerst, dann `.item1`, dann `.item3`.

Wann ist `order` sinnvoll?

- Für dynamische Interfaces, bei denen Inhalte sortierbar sind
- Für semantisch sauberen HTML-Code, aber visuell andere Reihenfolge
- In Kombination mit Media Queries für mobile-first Umstellungen

📌 **Hinweis**: `order` funktioniert nur innerhalb eines gemeinsamen Flex-Containers.

---

b) `align-content` – Ausrichtung mehrerer Flex-Zeilen

Diese Eigenschaft wird **nur relevant**, wenn `flex-wrap: wrap` aktiv ist und mehrere Zeilen vorhanden sind. Sie bestimmt, wie **ganze Zeilen** im Container verteilt werden – im Gegensatz zu `align-items`, das die **einzelnen Items innerhalb einer Zeile** ausrichtet.

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

Werte von `align-content`

- `stretch` (Standard): Zeilen dehnen sich aus, um den Container zu füllen
- `flex-start`: Zeilen am Anfang der Querachse ausrichten
- `flex-end`: Zeilen am Ende ausrichten
- `center`: Zeilen zentrieren
- `space-between`: Maximaler Abstand zwischen Zeilen

- `space-around`: Gleicher Abstand um jede Zeile
- `space-evenly`: Gleichmäßiger Abstand auch zu Randbereichen

Beispiel: Vergleich `align-items` vs. `align-content`

```
.container {
  flex-wrap: wrap;
  align-items: center;     /* Einzelne Items innerhalb Zeilen */
  align-content: center;   /* Ganze Zeilen im Container */
}
```

---

c) Kombination mit Media Queries (Praxisbezug)

Flexbox lässt sich hervorragend mit Media Queries kombinieren, z. B. für folgende Anwendungsfälle:

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

@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }
  .item {
    order: 2;
  }
  .item.special {
    order: 1;
  }
}
```

- Auf großen Bildschirmen: klassische horizontale Reihenfolge
- Auf kleineren Screens: Umschaltung auf `column`-Layout + geänderte Sortierung

---

d) Kombination mit Auto-Margin

In Flexbox kann `margin: auto` dazu verwendet werden, Items an bestimmten Stellen zu positionieren – ähnlich wie ein Push-System:

```
.item {
  margin-left: auto;
}
```

> Ergebnis: Das Item wird ganz nach rechts geschoben.

```
.item {
  margin: auto 0 auto auto;
}
```

> Ergebnis: Das Item wird rechts und vertikal zentriert.

---

e) Negative Abstände, Overflows und `min-width`

- **Negative Margins** können Flex-Verhalten stören

- `overflow: hidden` auf dem Container kann überstehende Items abschneiden
- `min-width: 0` ist essenziell, um das Schrumpfen innerhalb von Flexbox zu erlauben

```
.flex-item {
  min-width: 0;
}
```

> Manche Browser interpretieren das Verhalten von Items mit langen Wörtern (z. B. URLs) sonst fehlerhaft

---

Zusammenfassung `order` & `align-content`

Ziel	Eigenschaft	Bemerkung
Reihenfolge im Layout ändern	`order`	Funktioniert unabhängig vom HTML
Mehrere Zeilen vertikal ausrichten	`align-content`	Nur bei `flex-wrap: wrap`
Item nach rechts schieben	`margin-left: auto`	Simuliert „Push nach rechts“
Mobile-first Sortierung ändern	`order` + Media Query	Dynamisches Layout

---

6. Best Practices

- Verwende Flexbox für lineare Layouts (ein- oder mehrzeilig)
- Nutze `min-width: 0` bei flexiblen Elementen, um Überlauf zu vermeiden
- Verwende Flexbox mit Media Queries für responsive Layouts
- Vermeide komplexe verschachtelte Flexbox-Strukturen → lieber Flex + Grid kombinieren

---

7. Übungsaufgaben – Flexbox

Aufgabe 1: Flex-Zentrierung

Erstelle ein Flex-Container mit drei Boxen. Zentriere alle Items horizontal und vertikal im Container.

Aufgabe 2: Flex-Grow Vergleich

Erstelle drei Boxen mit unterschiedlichen `flex-grow`-Werten (1, 2, 3) und vergleiche die Breitenverteilung.

Aufgabe 3: Responsive Umbruch

Setze `flex-wrap: wrap` und lasse die Boxen bei kleinerem Bildschirm umbrechen. Nutze Media Query für `max-width: 600px`.

Aufgabe 4: Reihenfolge ändern

Setze bei drei Flex-Items unterschiedliche `order`-Werte und beobachte die Anzeigereihenfolge.

Aufgabe 5: Individuelle Ausrichtung

Nutze `align-self`, um ein einzelnes Item anders auszurichten als die anderen (z. B. `flex-end`).

Aufgabe 6: Flex mit Basisgröße

Vergebe `flex: 1 1 200px` für alle Items und beobachte die Verteilung in verschiedenen Containergrößen.