White Space und Text-Wrapping in CSS – eine ausführliche Anleitung

Wenn es um Textdarstellung in CSS geht, sind zwei Themen besonders wichtig:

1. **Wie Leerzeichen, Zeilenumbrüche und Tabulatoren behandelt werden**
2. **Wie und wo Text umbrochen werden darf**

Genau dafür gibt es in CSS mehrere Eigenschaften rund um **White Space** und **Text Wrap**. Manche davon sind altbekannt, andere stammen aus moderneren CSS-Spezifikationen. Zusammen steuern sie, ob Text in einer Zeile bleibt, umbricht, Leerzeichen zusammenfasst, lange Wörter trennt oder über den Container hinausragt.

Überblick: Was gehört thematisch dazu?

Die wichtigsten CSS-Eigenschaften in diesem Bereich sind:

1. `white-space`
2. `overflow-wrap`
3. `word-break`
4. `line-break`
5. `hyphens`
6. `text-wrap`
7. `tab-size`

Außerdem gibt es verwandte Themen, die oft damit verwechselt oder gemeinsam eingesetzt werden:

1. `writing-mode`
2. `direction`

3. `text-overflow`
4. `overflow`
5. `display` und verfügbare Breite des Elements

Das Grundproblem: Warum braucht man diese Eigenschaften?

Standardmäßig behandelt der Browser normalen Fließtext ungefähr so:

- Mehrere Leerzeichen hintereinander werden meist zu **einem** Leerzeichen zusammengefasst.
- Zeilenumbrüche im HTML-Quelltext werden meist nicht als sichtbare neue Zeilen dargestellt.
- Text darf an geeigneten Stellen automatisch umbrechen.
- Sehr lange Wörter oder URLs können Probleme machen, wenn kein sinnvoller Umbruchpunkt existiert.

Beispiel:

```
<p>
  Das    ist      Text
  mit mehreren Leerzeichen
  und einem Zeilenumbruch im HTML.
</p>
```

Ohne besondere CSS-Regeln wird das im Browser ungefähr wie ein normaler Satz dargestellt – also nicht mit exakt denselben Leerzeichen und Zeilenumbrüchen wie im Quelltext.

`white-space` – die zentrale Eigenschaft

Die wichtigste Eigenschaft für White Space ist `white-space`. Sie steuert vor allem:

- ob Leerzeichen zusammengefasst werden
- ob Zeilenumbrüche aus dem Quelltext erhalten bleiben
- ob automatischer Zeilenumbruch erlaubt ist

Syntax

```
.element {
  white-space: normal;
}
```

Die wichtigsten Werte

`white-space: normal`

Das ist das Standardverhalten.

- Mehrere Leerzeichen werden zusammengefasst.
- Zeilenumbrüche im Quelltext werden ignoriert bzw. wie normale Leerzeichen behandelt.
- Automatischer Zeilenumbruch ist erlaubt.

```
p {
  white-space: normal;
}
```

Typischer Einsatz: normaler Fließtext.

`white-space: nowrap`

- Leerzeichen werden weiterhin zusammengefasst.
- Zeilenumbrüche aus dem Quelltext werden nicht als echte neue Zeilen behandelt.
- **Automatischer Zeilenumbruch wird verhindert.**

```
.badge {
  white-space: nowrap;
}
```

Effekt: Der gesamte Text bleibt in einer Zeile, sofern nicht explizit ein `<br>` oder ähnliches vorhanden ist.

Typische Einsätze:

- Buttons
- Labels
- Tabellenzellen
- Navigationseinträge
- „Nicht umbrechen“-Bereiche wie Preisangaben oder Icons mit Text

Achtung: In schmalen Containern kann das zu horizontalem Overflow führen.

`white-space: pre`

Dieses Verhalten ähnelt dem HTML-Element `<pre>`.

- Leerzeichen bleiben erhalten.
- Zeilenumbrüche bleiben erhalten.
- Automatischer Umbruch findet **nicht** statt.

```
pre,
.code-like {
  white-space: pre;
}
```

Typischer Einsatz:

- Codeblöcke
- ASCII-Layouts
- Inhalte, bei denen exakte Einrückung relevant ist

`white-space: pre-wrap`

- Leerzeichen bleiben erhalten.
- Zeilenumbrüche bleiben erhalten.
- Automatischer Zeilenumbruch ist **zusätzlich** erlaubt.

```
.message {
  white-space: pre-wrap;
}
```

Das ist sehr praktisch für Inhalte wie:

- Benutzereingaben
- Chat-Nachrichten
- Kommentare
- Texte aus Textareas

Denn damit bleiben manuelle Zeilenumbrüche erhalten, aber lange Zeilen können dennoch umbrechen.

`white-space: pre-line`

- Mehrere Leerzeichen werden zusammengefasst.
- Zeilenumbrüche bleiben erhalten.
- Automatischer Zeilenumbruch ist erlaubt.

```
.poem {
  white-space: pre-line;
}
```

Unterschied zu `pre-wrap`:

- `pre-wrap` bewahrt auch mehrere Leerzeichen
- `pre-line` bewahrt nur Zeilenumbrüche, aber nicht die exakte Anzahl von Leerzeichen

---

`white-space: break-spaces`

Ein moderner Wert mit sehr speziellem Verhalten.

- Leerzeichen bleiben erhalten.
- Zeilenumbrüche bleiben erhalten.
- Umbruch kann auch an erhaltenen Leerzeichen stattfinden.
- Nachfolgende Leerzeichen am Zeilenende bleiben relevant.

```
.output {
  white-space: break-spaces;
}
```

Dieser Wert ist nützlich, wenn wirklich die **sichtbare Struktur von Leerzeichen** wichtig ist.

Vergleichstabelle zu `white-space`

Wert	Leerzeichen erhalten?	Zeilenumbrüche erhalten?	automatischer Umbruch?
`normal`	Nein	Nein	Ja
`nowrap`	Nein	Nein	Nein
`pre`	Ja	Ja	Nein
`pre-wrap`	Ja	Ja	Ja
`pre-line`	Nein	Ja	Ja
`break-spaces`	Ja	Ja	Ja

Moderne Aufteilung: White-Space-Untereigenschaften

In neueren CSS-Spezifikationen wird `white-space` konzeptionell in feinere Teilaspekte aufgeteilt. Dazu gehören unter anderem:

1. `white-space-collapse`
2. `text-wrap-mode`
3. `white-space-trim`

Diese sind konzeptionell wichtig, aber **noch nicht überall gleich gut etabliert** wie `white-space`. In der Praxis verwendet man deshalb meist weiterhin `white-space`.

`white-space-collapse`

Diese Eigenschaft steuert, wie Leerraum zusammengefasst oder erhalten wird.

Mögliche Werte sind je nach Spezifikation unter anderem:

- `collapse`
- `preserve`
- `preserve-breaks`

- `preserve-spaces`
- `break-spaces`

Beispielidee:

```
.element {
  white-space-collapse: preserve;
}
```

Praxis-Hinweis: Diese Eigenschaft ist interessant für moderne CSS-Modelle, aber für produktive, breit kompatible Websites ist `white-space` oft die sicherere Wahl.

`text-wrap-mode`

Sie beschreibt grundsätzlich, ob Zeilenumbruch erlaubt ist.

Typische Werte:

- `wrap`
- `nowrap`

Beispiel:

```
.element {
  text-wrap-mode: nowrap;
}
```

Auch hier gilt: In der Praxis ist meist `white-space: nowrap;` der bekanntere und breiter eingesetzte Weg.

`white-space-trim`

Diese Eigenschaft soll beeinflussen, ob bestimmte Whitespaces an Anfang oder Ende entfernt werden.

Beispielhaft konzeptionell:

```
.element {
  white-space-trim: discard-before;
```

```
}
```

Auch das ist eher ein fortgeschrittenes bzw. modernes Thema mit eingeschränkter Relevanz im Alltag.

`overflow-wrap` – was passiert mit langen Wörtern?

`overflow-wrap` bestimmt, ob der Browser **lange Wörter oder Zeichenketten umbrechen darf**, wenn sie sonst den Container sprengen würden.

Früher war dafür oft `word-wrap` im Einsatz. Das ist heute im Grunde ein Alias für `overflow-wrap`.

Syntax

```
.element {
  overflow-wrap: normal;
}
```

Werte

`overflow-wrap: normal`

Der Browser bricht nur an normalen Umbruchstellen um.

```
.element {
  overflow-wrap: normal;
}
```

Lange URLs oder zusammengesetzte Wörter können dann überlaufen.

`overflow-wrap: break-word`

Falls nötig, darf ein langes Wort umgebrochen werden, um Overflow zu verhindern.

```
.article {
  overflow-wrap: break-word;
}
```

Typischer Einsatz:

- CMS-Inhalte
- Foren
- Kommentare
- UGC („user generated content“)
- lange URLs

`overflow-wrap: anywhere`

Noch aggressiver: Der Browser darf an praktisch jeder Stelle umbrechen, wenn nötig.

```
.article {
  overflow-wrap: anywhere;
}
```

Wann sinnvoll?

- bei extrem langen Tokens
- bei technischen IDs
- bei URLs ohne sinnvolle Trennpunkte
- in sehr schmalen Layouts

Unterschied zu `break-word`:
`anywhere` erlaubt Umbrüche sehr frei und berücksichtigt diese Möglichkeiten auch stärker bei der Zeilenberechnung.

`word-wrap` – historischer Alias

```
.element {
  word-wrap: break-word;
```

```
}
```

Das funktioniert oft noch, aber modern und sauber ist:

```
.element {
  overflow-wrap: break-word;
}
```

`word-break` – wie aggressiv darf in Wörtern gebrochen werden?

`word-break` beeinflusst den Umbruch **innerhalb von Wörtern** stärker als `overflow-wrap`.

Syntax

```
.element {
  word-break: normal;
}
```

Werte

`word-break: normal`

Normales Umbruchverhalten nach Sprach- und Schriftsystemregeln.

```
.element {
  word-break: normal;
}
```

`word-break: break-all`

Wörter dürfen praktisch an beliebigen Stellen getrennt werden.

```
.element {
  word-break: break-all;
}
```

Effekt: Auch normale Wörter können mitten im Wort umbrechen.

Das löst Overflow-Probleme zuverlässig, sieht aber oft unschön aus.

Einsatz nur mit Vorsicht, z. B. bei:

- sehr schmalen technischen Layouts
- Tabellen mit langen Schlüsseln
- maschinenlesbaren Zeichenfolgen

`word-break: keep-all`

Verhindert Wortumbrüche innerhalb von Wörtern, besonders relevant für ostasiatische Schriftsysteme.

```
.element {
  word-break: keep-all;
}
```

Für deutschsprachige Seiten ist dieser Wert seltener relevant, kann aber in internationalen Projekten wichtig sein.

Unterschied zwischen `overflow-wrap` und `word-break`

Das ist einer der häufigsten Stolperpunkte.

`overflow-wrap`

- greift vor allem dann, wenn ein Wort **sonst überlaufen würde**
- ist eher eine „Notfalllösung“

`word-break`

- beeinflusst allgemeiner, **wie innerhalb von Wörtern getrennt werden darf**
- ist meist aggressiver

Faustregel

1. Erst `overflow-wrap` prüfen
2. Nur wenn das nicht reicht, `word-break` einsetzen
3. `break-all` nur bewusst und sparsam verwenden

`hyphens` – automatische Silbentrennung

Mit `hyphens` kann der Browser Wörter an geeigneten Stellen trennen – oft mit Bindestrich.

Syntax

```
.element {
  hyphens: auto;
}
```

Werte

`hyphens: none`

Keine Silbentrennung.

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

`hyphens: manual`

Nur manuell vorgegebene Trennstellen werden verwendet.

```
.element {
  hyphens: manual;
}
```

Manuelle Trennstellen können z. B. im HTML gesetzt werden.

`hyphens: auto`

Automatische Silbentrennung nach Sprachregeln, sofern der Browser und die Spracheinstellungen das unterstützen.

```
p {
  hyphens: auto;
}
```

Wichtig ist dabei oft ein korrekt gesetztes `lang`-Attribut:

```
<html lang="de">
```

oder

```
<p lang="de">Donaudampfschifffahrtsgesellschaftskapitän</p>
```

Ohne passende Sprache kann automatische Silbentrennung unzuverlässig sein.

Wann ist `hyphens` nützlich?

- schmale Textspalten
- Magazine-Layouts
- lange deutsche Wörter
- bessere Blocksatzdarstellung

Beispiel

```
.article-text {
  hyphens: auto;
  overflow-wrap: normal;
}
```

Das führt oft zu schöneren Ergebnissen als brutal mit `word-break: break-all` zu arbeiten.

`line-break` – Regeln für Zeilenumbrüche, besonders in asiatischen Schriften

`line-break` steuert die Strenge der Umbruchregeln, insbesondere für chinesische, japanische und koreanische Texte.

Syntax

```
.element {
  line-break: auto;
}
```

Typische Werte

- `auto`
- `loose`
- `normal`
- `strict`
- `anywhere`

Beispiel:

```
.element {
  line-break: strict;
}
```

Für deutschsprachige Seiten ist diese Eigenschaft meist **nicht zentral**, aber in mehrsprachigen Projekten kann sie wichtig sein.

`text-wrap` – moderne Steuerung für Zeilenumbruch

`text-wrap` ist eine modernere Eigenschaft für Strategien des Textumbruchs.

Syntax

```
.element {
  text-wrap: wrap;
}
```

Mögliche Werte

Je nach aktuellem Implementierungsstand sind insbesondere diese relevant:

- `wrap`

- `nowrap`
- `balance`
- `pretty`
- `stable`

Nicht jeder Wert wird in jedem Browser gleich unterstützt.

`text-wrap: wrap`

Normales Umbruchverhalten.

```
.element {
  text-wrap: wrap;
}
```

`text-wrap: nowrap`

Kein automatischer Umbruch.

```
.element {
  text-wrap: nowrap;
}
```

Praktisch ähnlich zu `white-space: nowrap`, aber konzeptionell moderner auf den Umbruch fokussiert.

`text-wrap: balance`

Versucht, Zeilen ausgewogener zu verteilen. Besonders nützlich für Überschriften.

```
h1, h2 {
  text-wrap: balance;
}
```

Sehr sinnvoll für:

- Headlines

- Hero-Texte
- Card-Titel

Statt einer sehr langen und einer sehr kurzen Zeile versucht der Browser, ein harmonischeres Ergebnis zu erzeugen.

`text-wrap: pretty`

Zielt auf optisch angenehmere Umbrüche ab und versucht unschöne Ein-Zeilen-Wörter oder ungünstige Brüche zu vermeiden.

```
p {
  text-wrap: pretty;
}
```

Die Unterstützung kann je nach Browserstand variieren.

`text-wrap: stable`

Soll Umbruchverhalten stabil halten, z. B. bei bearbeitbaren Inhalten. Auch das ist eher ein fortgeschrittenes Thema.

`tab-size` – Breite von Tabulatoren

Wenn ein Text Tab-Zeichen enthält, kann `tab-size` deren visuelle Breite bestimmen.

Syntax

```
.element {
  tab-size: 4;
```

```
}
```

oder

```
.element {
  tab-size: 2;
}
```

Beispiel

```
pre {
  white-space: pre;
  tab-size: 4;
}
```

Das ist vor allem für Code, Logs oder vorformatierte Texte nützlich.

Wichtige verwandte Eigenschaften

`text-overflow`

Wenn Text nicht umbrochen wird und der Container zu klein ist, kann `text-overflow` festlegen, wie abgeschnittener Text dargestellt wird.

Typisch:

```
.ellipsis {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

Das erzeugt die bekannte Darstellung mit „…“.

Wichtig: `text-overflow` funktioniert typischerweise nur in Kombination mit:

1. begrenzter Breite
2. `overflow: hidden`
3. meist `white-space: nowrap`

`overflow`

Wenn Text nicht in den Container passt, beeinflusst `overflow`, ob er sichtbar bleibt, abgeschnitten wird oder Scrollbars erscheinen.

```
.box {
  overflow: auto;
}
```

Mögliche Werte:

- `visible`
- `hidden`
- `clip`
- `scroll`
- `auto`

`display` und Breite

Textumbruch hängt nicht nur von Text-Eigenschaften ab, sondern auch davon, **ob überhaupt eine begrenzte Breite existiert**.

Beispiel:

```
.inline-label {
  display: inline;
}
```

Ein rein inline dargestelltes Element verhält sich anders als ein Block mit fester oder maximaler Breite.

Oft braucht man für sichtbaren Umbruch:

```
.card-title {
  display: block;
  max-width: 20rem;
}
```

Typische Praxisrezepte □□

1. Normaler Fließtext

```
p {
  white-space: normal;
  overflow-wrap: break-word;
  hyphens: auto;
}
```

Gut für: Artikel, Blogposts, CMS-Inhalte

Vorteil: normale Darstellung, aber robuste Behandlung langer Wörter.

---

2. Lange URLs oder unkontrollierter User-Content

```
.user-content {
  overflow-wrap: anywhere;
}
```

Gut für: Kommentare, Foren, Chat, Markdown-Content

---

3. Eine Zeile mit Auslassungspunkten

```
.one-line-ellipsis {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

---

4. Vorformatierter Text oder Code

```
pre {
  white-space: pre;
  tab-size: 4;
  overflow: auto;
}
```

Wenn Code stattdessen umbrechen soll:

```
pre.wrap {
  white-space: pre-wrap;
  overflow-wrap: anywhere;
}
```

---

5. Benutzereingaben mit erhaltenen Zeilenumbrüchen

```
.user-message {
  white-space: pre-wrap;
  overflow-wrap: break-word;
```

```
}
```

Das ist ein sehr typisches und sinnvolles Setup.

---

6. Schöne Überschriften umbrechen

```
h1, h2, h3 {
  text-wrap: balance;
}
```

Fallback-orientiert kann man einfach zusätzlich auf normales Verhalten vertrauen, falls der Browser den Wert nicht unterstützt.

Häufige Stolperfallen

1. `nowrap` „funktioniert zu gut“

```
.badge {
  white-space: nowrap;
}
```

Dann bleibt wirklich alles in einer Zeile. Wenn der Container schmal ist, läuft der Text möglicherweise heraus.

Lösung: Nur dort einsetzen, wo es wirklich gewünscht ist.

---

2. `text-overflow: ellipsis` zeigt keine Punkte

Nur diese Regel reicht nicht:

```
.element {
  text-overflow: ellipsis;
}
```

Meist braucht man zusätzlich:

```
.element {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

Und oft auch eine definierte Breite oder max. Breite.

---

3. Lange Wörter brechen trotzdem nicht um

Wenn ein langes Wort oder eine URL herausragt, hilft oft:

```
.element {
  overflow-wrap: break-word;
}
```

oder noch robuster:

```
.element {
  overflow-wrap: anywhere;
}
```

---

4. `hyphens: auto` bringt nichts

Dann fehlt oft eines der folgenden Dinge:

1. Das richtige `lang`-Attribut
2. Browser-Unterstützung
3. Geeigneter Fließtext-Kontext mit tatsächlichem Umbruchbedarf

---

5. In Flex- oder Grid-Layouts bricht Text nicht wie erwartet um

Das Problem liegt oft **nicht** an `white-space`, sondern an den Mindestgrößen der Items.

Ein klassischer Fall:

```
.flex-item {
  overflow-wrap: break-word;
}
```

Und trotzdem bricht der Text nicht schön um.

Dann kann zusätzlich nötig sein:

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

Oder in Grid-Layouts ebenfalls eine passende Größenlogik.

Das ist ein sehr häufiger Praxisfehler.

Unterschiede kurz und prägnant zusammengefasst

Wenn du Leerzeichen und Zeilenumbrüche aus dem Quelltext steuern willst

→ `white-space`

Wenn lange Wörter oder URLs den Container sprengen

→ `overflow-wrap`

Wenn innerhalb von Wörtern aggressiver getrennt werden soll

→ `word-break`

Wenn automatische Silbentrennung gewünscht ist

→ `hyphens`

Wenn Überschriften schöner umbrechen sollen

→ `text-wrap: balance`

Wenn Tabs in vorformatiertem Text korrekt aussehen sollen

→ `tab-size`

Empfehlenswerte Standardstrategien

Für normalen Content

```
.content {
  white-space: normal;
  overflow-wrap: break-word;
  hyphens: auto;
}
```

Für User-Generated Content

```
.user-content {
  white-space: pre-wrap;
  overflow-wrap: anywhere;
}
```

Das bewahrt Eingabe-Zeilenumbrüche und verhindert Layoutbruch durch lange Tokens.

Für Titel

```
.title {
  text-wrap: balance;
}
```

Für einzeilige UI-Elemente

```
.chip {
  white-space: nowrap;
}
```

Für abgeschnittene Ein-Zeilen-Texte

```
.truncate {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

Beispiel: alles in einer kleinen Demo

```
<div class="demo">
  <p class="normal">
    Das ist ein normaler Fließtext mit einer
```

```
sehrlangenbeispielzeichenketteohnesinnvollenumbruch.
  </p>

  <p class="message">
    Hallo!
    Dies ist eine Nachricht
    mit manuellem Zeilenumbruch.
  </p>

  <p class="truncate">
    Dies ist ein sehr langer Titel, der in einer Zeile abgeschnitten werden soll.
  </p>

  <pre class="code">function test() {
\tconsole.log("Hallo");
}</pre>
</div>
```

```
.demo {
  max-width: 20rem;
  font-family: system-ui, sans-serif;
}

.normal {
  white-space: normal;
  overflow-wrap: break-word;
  hyphens: auto;
}

.message {
  white-space: pre-wrap;
  overflow-wrap: break-word;
}

.truncate {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

```
.code {
  white-space: pre;
  tab-size: 4;
  overflow: auto;
}
```

Browser-Kompatibilität – was ist besonders sicher?

Sehr gut etabliert und in der Praxis „sicher“:

- `white-space`
- `overflow-wrap`
- `word-break`
- `hyphens` mit gewissen sprach- und browserabhängigen Unterschieden
- `tab-size`
- `text-overflow`

Moderner bzw. teils mit Vorsicht zu prüfen:

- `text-wrap` mit Werten wie `balance`, `pretty`, `stable`
- Untereigenschaften wie `white-space-collapse`, `text-wrap-mode`, `white-space-trim`

Empfehlung: Für produktive Projekte moderne Features gern nutzen, aber mit sinnvollen Fallbacks.

Eine praktische Merkhilfe □□

Du kannst dir das so merken:

1. **`white-space`**
   „Wie behandle ich Leerzeichen und Zeilenumbrüche grundsätzlich?“
2. **`overflow-wrap`**
   „Was mache ich mit langen Wörtern, wenn sie sonst überlaufen?“
3. **`word-break`**
   „Darf ich Wörter auch mitten drin zerlegen?“

4. `hyphens`
   „Kann ich Wörter sprachgerecht trennen?“
5. `text-wrap`
   „Welche Strategie soll der Browser für den Umbruch wählen?“
6. `tab-size`
   „Wie breit sind Tabs in vorformatiertem Text?“

Fazit

Die wichtigste Eigenschaft für White Space ist `white-space`. Sie legt die Basis dafür, wie Text, Leerzeichen und Zeilenumbrüche verarbeitet werden. Für robuste Layouts reicht das allein aber oft nicht aus.

In der Praxis ist diese Kombination besonders häufig sinnvoll:

```
.element {
  white-space: normal;
  overflow-wrap: break-word;
  hyphens: auto;
}
```

Für Benutzereingaben dagegen oft eher:

```
.element {
  white-space: pre-wrap;
  overflow-wrap: break-word;
}
```

Und für moderne, schön umbrechende Überschriften:

```
h1, h2 {
  text-wrap: balance;
}
```

Wenn du möchtest, kann ich dir im nächsten Schritt noch eine **kompakte Cheat Sheet-Tabelle**, eine **Browser-Kompatibilitätsübersicht** oder eine **visuelle Demo mit HTML/CSS-Beispielen für jeden Wert** erstellen.

Revision #1
Created 2026-05-18 14:55:00 UTC by art10m
Updated 2026-05-18 14:56:16 UTC by art10m