



POSITIONIERUNG UND LAYOUTING

ELEMENTFLUSS

- Der **Elementfluss** ist die Aneinanderreichung von Elementen in einem HTML-Dokument
- Wir haben bereits zwei Klassen von HTML-Elementen bzgl. ihrer Eigenschaften im Elementfluss kennengelernt:

Block-Elemente (z.B. `p`, `table`, `h1`)

erwirken Zeilenumbruch, nehmen vollen horizontalen Platz ein

Inline-Elemente (z.B. `strong`, `img`, `input`)

kein Zeilenumbruch, nehmen nur soviel horizontalen Platz ein wie nötig

NORMALER ELEMENTFLUSS

style.css


```
/* Inline- und Box-Elemente markieren */  
.inline { border: 2px dotted blue; }  
.box { border: 1px solid orange; }
```

seite.html

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Titel meiner Web-Seite</title>  
    <meta charset="utf-8">  
    <link rel="stylesheet" type="text/css" href="style.css">  
  </head>  
  
  <body>  
    <h1 class="box">Überschrift</h1>  
    <p class="box">  
      Lorem ipsum <em class="inline">dolor sit amet</em>,  
      consectetur adipisicing elit, <code class="inline">  
      sed do eiusmod tempor</code> incididunt ut  
      labore  et  
      dolore magna aliqua.  
    </p>  
    <ul class="box">  
      <li class="box">JavaScript</li>  
      <li class="box">HTML</li>  
      <li class="box">CSS</li>  
    </ul>  
  </body>  
</html>
```

ÜBERSCHRIFT

Lorem ipsum *dolor sit amet*, consectetur adipisicing elit, `sed do`

`eiusmod tempor` incididunt ut labore  et dolore magna aliqua.

- JavaScript
- HTML
- CSS

VERHALTEN VON ELEMENTEN ÄNDERN

Eigenschaft `display`

- Erlaubt die Änderung des Anzeigeverhaltens von HTML-Elementen
- Mögliche Werte sind u.A.:
 - `inline`: Element verhält sich wie ein Inline-Element
 - `block`: Element verhält sich wie ein Block-Element
 - `inline-block`: Element verhält sich wie ein Inline-Element, kann jedoch wie ein Block formatiert werden (z.B. mit Breite, Höhe, Außenabstand)
 - `none`: Element wird ausgeblendet

! Wir werden im Bereich der Layouts noch weitere Werte für `display` kennenlernen.

display: BEISPIEL

style.css

```
.inline {  
  display: inline;  
  border: 2px dotted blue;  
}  
.block {  
  display: block;  
  border: 1px solid orange;  
}  
.ib {  
  display: inline-block;  
  width: 35px;  
}  
.none { display: none; }
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
    <h1 class="block">Überschrift</h1>  
    <p class="block">  
      Lorem ipsum <em class="inline ib">dolor sit amet</em>,  
      consectetur adipisicing elit, <code class="inline">  
        sed do eiusmod tempor</code> incididunt ut  
        labore  et  
        dolore magna aliqua.  
    </p>  
    <ul class="block">  
      <li class="inline">JavaScript</li>  
      <li class="inline">HTML</li>  
      <li class="inline">CSS</li>  
    </ul>  
  </body>  
</html>
```

ÜBERSCHRIFT

>Lorem ipsum *dolor sit amet*, consectetur adipisicing elit, `sed do eiusmod`
tempor incididunt ut labore et dolore magna aliqua.

JavaScript HTML CSS

POSITIONIERUNG VON ELEMENTEN

- CSS bietet mehrere Möglichkeiten, die Positionierung und Anordnung von Elementen zu bestimmen
- Wir betrachten im Folgenden:
 1. Positionierungsmodell (`position`)
 2. Float-Modell
 3. Flexbox
 4. Grid

POSITIONIERUNG VON ELEMENTEN

- CSS bietet mehrere Möglichkeiten, die Positionierung und Anordnung von Elementen zu bestimmen
- Wir betrachten im Folgenden:
 1. Positionierungsmodell (`position`)
 2. Float-Modell
 3. Flexbox
 4. Grid

POSITIONIERUNGSMODELL

Eigenschaft **position**

Erlaubt die Positionierung von Elementen auf fünf Arten:

1. Statische Positionierung (Standard): `static`= normaler Elementfluss
2. Relative Positionierung: `relative`
3. Absolute Positionierung: `absolute`
4. Fixe Positionierung: `fixed`
5. "Sticky" Positionierung: `sticky`

! Ein Element gilt als **positioniert**, wenn es die Positionsart `relative`, `absolute`, `fixed` oder `sticky` hat.

POSITIONIERUNGSMODELL (2)

Eigenschaften **top**, **right**, **bottom**, **left**

- Ermöglichen die Positionierung von Elementen gemäß der gewählten Positionierungsart
- Erwarten einen numerischen Wert, hierbei gilt:
 - Ein positiver Wert bewirkt eine Verschiebung nach innen
 - Ein negativer Wert bewirkt eine Verschiebung nach außen

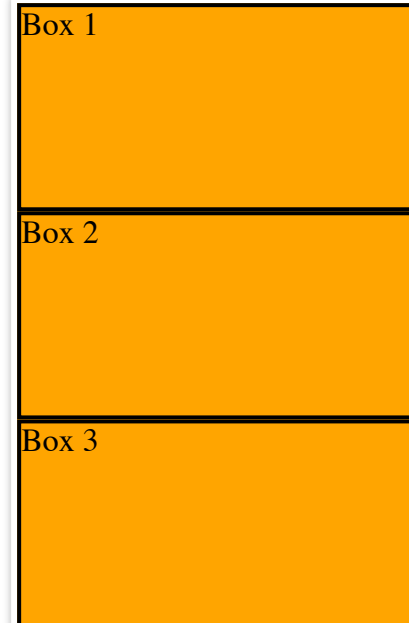
STATISCHE POSITIONIERUNG: BEISPIEL

style.css

```
div {  
  width: 200px;  
  height: 100px;  
  border: 2px solid;  
  background-color: orange;  
}  
.positioned {  
  /* Kann auch weggelassen werden,  
    da static Standard ist */  
  position: static;  
  /* top, right, bottom und left  
    haben bei static keine Auswirkung */  
  top: 50px;  
}
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
    <div>Box 1</div>  
    <div class="positioned">Box 2</div>  
    <div>Box 3</div>  
  </body>  
</html>
```



RELATIVE POSITIONIERUNG

Deklaration **position: relative**

- Element wird relativ zu seiner normalen Position im Elementfluss verschoben
- Alle anderen Elemente ignorieren die Verschiebung und bleiben im normalen Elementfluss

! Effekte:

1. Es entsteht zumeist ein Leerraum
2. Elemente können sich gegenseitig überlagern

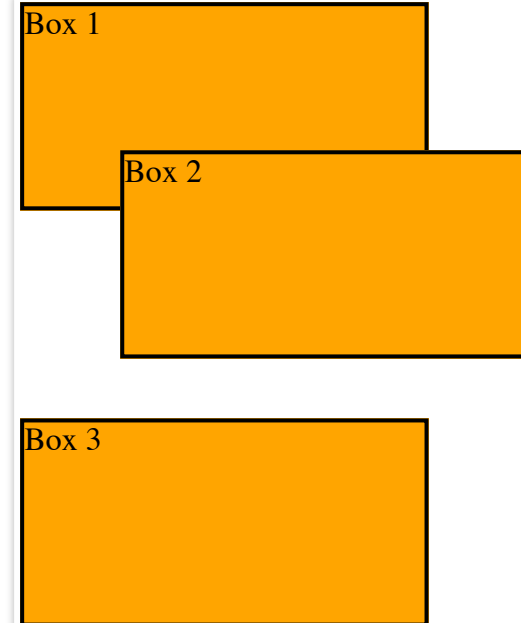
RELATIVE POSITIONIERUNG: BEISPIEL

style.css

```
div {  
  width: 200px;  
  height: 100px;  
  border: 2px solid;  
  background-color: orange;  
}  
.positioned {  
  position: relative;  
  top: -30px;  
  left: 50px;  
}
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
    <div>Box 1</div>  
    <div class="positioned">Box 2</div>  
    <div>Box 3</div>  
  </body>  
</html>
```



ABSOLUTE POSITIONIERUNG

Deklaration **position: absolute**

- Element wird aus dem normalen Elementfluss herausgenommen und relativ zum nächsten positionierten Vorfahrenelement verschoben (falls kein solches existiert → relativ zum Wurzelement `html`)
- Alle anderen Elemente tun so, als sei das verschobene Element gar nicht vorhanden

! Effekte:

1. Leerraum wird aufgefüllt
2. Elemente können sich gegenseitig überlagern

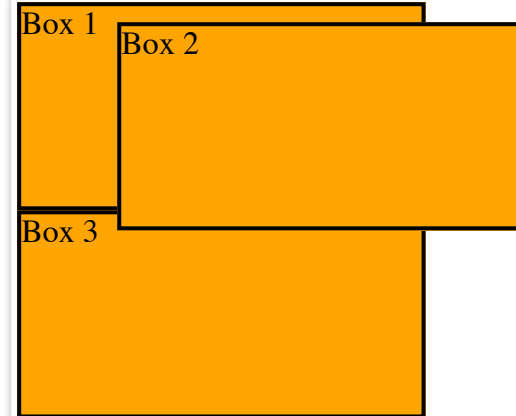
ABSOLUTE POSITIONIERUNG: BEISPIEL

style.css

```
div {  
  width: 200px;  
  height: 100px;  
  border: 2px solid;  
  background-color: orange;  
}  
.positioned {  
  position: absolute;  
  top: 10px;  
  left: 50px;  
}
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
    <div>Box 1</div>  
    <div class="positioned">Box 2</div>  
    <div>Box 3</div>  
  </body>  
</html>
```



FIXE POSITIONIERUNG

Deklaration **position: fixed**

- Entspricht `absolute`, jedoch wird das Element hier so fixiert, dass es nicht mehr mitgescrollt wird
- ! Anwendungsbeispiel: fixierte Kopfbereiche, Fußbereiche oder Menüs

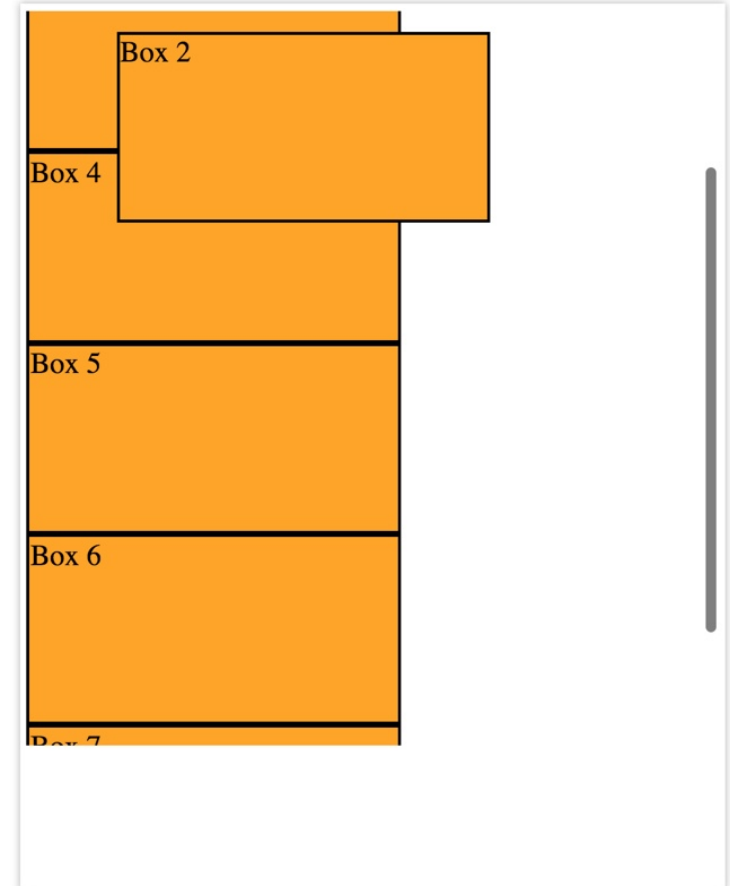
FIXE POSITIONIERUNG: BEISPIEL

style.css

```
div {  
  width: 200px;  
  height: 100px;  
  border: 2px solid;  
  background-color: orange;  
}  
.positioned {  
  position: fixed;  
  top: 10px;  
  left: 50px;  
}
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
    <div>Box 1</div>  
    <div class="positioned">Box 2</div>  
    <div>Box 3</div>  
    <div>Box 4</div>  
    <div>Box 5</div>  
    <div>Box 6</div>  
    <div>Box 7</div>  
  </body>  
</html>
```



STICKY POSITIONIERUNG

Deklaration **position: sticky**

- Kombination aus `relative` und `fixed`: Element scrollt bis zu einem gewissen Punkt mit und bleibt danach fixiert ("festgeklebt"), bis es an den unteren Rand seines Elternelements "stößt"
- Beispiel:

```
.positioned {  
  /* Element scrollt mit, bis es 10px vom oberen Rand  
    des Scrollbereiches entfernt ist - danach bleibt es  
    "kleben" */  
  position: sticky;  
  top: 10px;  
}
```

POSITIONIERUNG VON ELEMENTEN

- CSS bietet mehrere Möglichkeiten, die Positionierung und Anordnung von Elementen zu bestimmen
- Wir betrachten im Folgenden:
 1. Positionierungsmodell (`position`)
 2. Float-Modell
 3. Flexbox
 4. Grid

FLOAT-MODELL

Eigenschaft `float`

- Element wird teilweise aus dem normalen Elementfluss herausgenommen und an der linken (Wert `left`) oder rechten (Wert `right`) Innenkante des umgebenen Elements platziert
- Nur teilweises Herausnehmen aus dem Elementfluss, denn nachfolgende Elemente "umfließen" das Element

FLOAT-MODELL: BEISPIEL

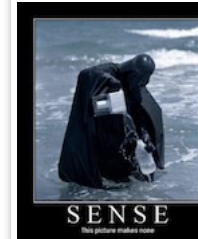
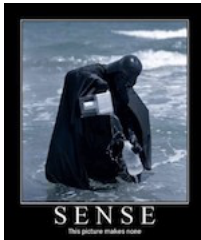
style.css

```
#pic1 { float: right; }  
#pic2 { float: left; }
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
      
    <p>Lorem ipsum [...]</p>  
      
    <p>Lorem ipsum [...]</p>  
  </body>  
</html>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

UMFLIESSEN AUFHEBEN

Eigenschaft `clear`

- Beendet das Umfließen "floatierter" Elemente
- Mögliche Werte:
 - `left`: Beendet `float:left`
 - `right`: Beendet `float:right`
 - `both`: Beendet beides

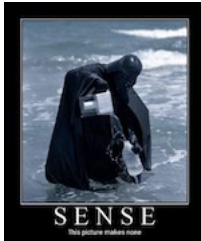
UMFLIESSEN BEENDEN: BEISPIEL

style.css

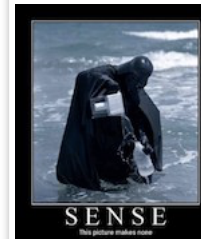
```
#pic1 { float: right; }  
#pic2 { float: left; }  
#p1 { clear: right; }  
#p2 { clear: left; }
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
      
    <p id="p1">Lorem ipsum [...]</p>  
      
    <p id="p2">Lorem ipsum [...]</p>  
  </body>  
</html>
```



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

LAYOUTS MIT FLOAT

- `float` kann grundsätzlich auch verwendet werden, um Layouts zu erstellen
 - Layouting mit `float` beinhaltet ein paar Stolpersteine*, z.B.:
 - Breite (`width`) sollte für gefloatete Elemente gesetzt sein, da sonst der Inhalt die Breite bestimmt (→ weniger Kontrolle über das Layout)
 - Bei spaltenbasierten Layouts muss mit "Workarounds" gearbeitet werden, um Spalten mit gleicher Höhe zu erhalten
 - Vertikales Zentrieren von Elementen ist schwierig
- 💡 Tipp: `float` nicht zur Erstellung von Layouts verwenden → Flexbox, Grid

* für die meisten gibt es mehr oder weniger elegante Lösungen 😊

POSITIONIERUNG VON ELEMENTEN

- CSS bietet mehrere Möglichkeiten, die Positionierung und Anordnung von Elementen zu bestimmen
- Wir betrachten im Folgenden:
 1. Positionierungsmodell (`position`)
 2. Float-Modell
 - 3. Flexbox**
 4. Grid

FLEXBOX

- **Flexbox** (kurz für: *Flexible Box*) ist ein [CSS-Modul](#) zur Anordnung und Positionierung von Elementen
- Grundprinzip:
 - Flexbox benötigt ein Elternelement (den **Flex-Container**)
 - Der Flex-Container hat die Fähigkeit, die enthaltenen Kindelemente (die **Flex-Items**) bzgl. ihrer Höhe, Breite und Anordnung anzupassen
 - Für die Flex-Items wird daher auf Größenangaben verzichtet - sie sind *flexibel* und werden vom Flex-Container verwaltet

FLEX-CONTAINER

Flex-Container definieren

Ein Element wird zum Flex-Container durch Setzen der Eigenschaft `display: flex`

Richtung der Flex-Items festlegen: **flex-direction**

- Wert `row` (Standard): horizontal von links nach rechts
- Wert `column`: vertikal von oben nach unten
- Wert `row-reverse`: horizontal von rechts nach links
- Wert `column-reverse`: vertikal von unten nach oben

FLEX-CONTAINER: BEISPIEL

style.css

```
#container1, #container2 {
  display: flex;
  background-color: darkorange;
  padding: 10px;
  margin: 5px;
  height: 100px;
}

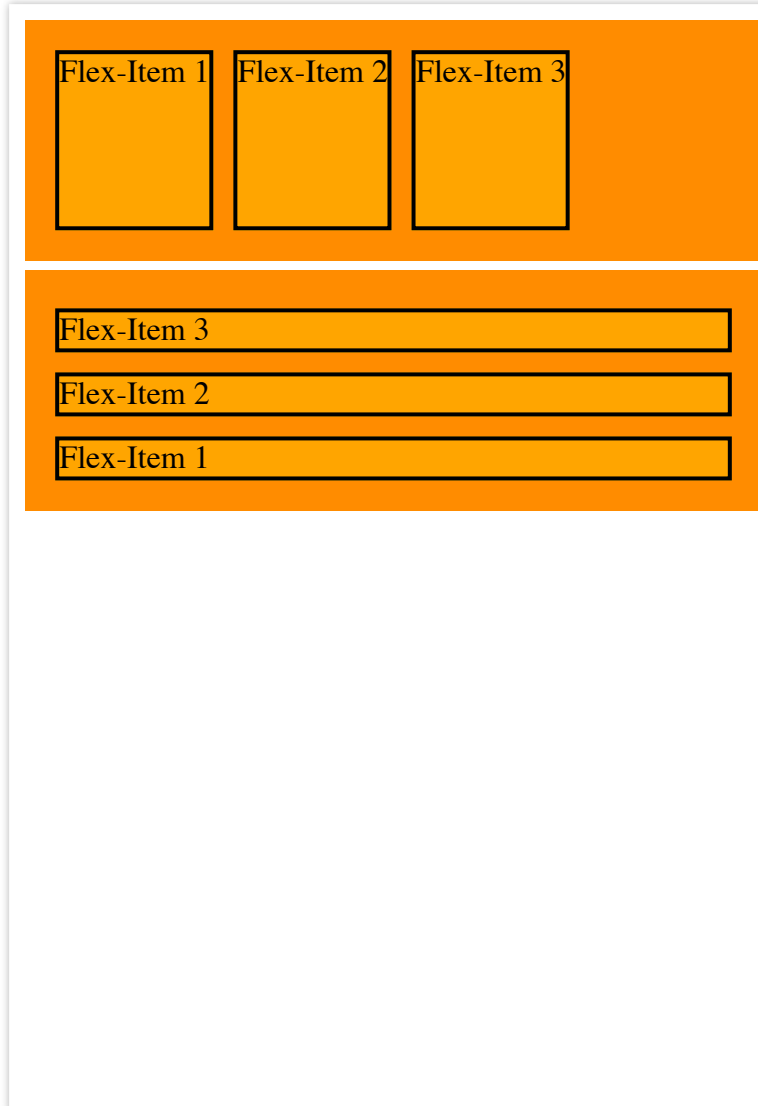
#container2 { flex-direction: column-reverse; }

div div {
  border: 2px solid;
  background-color: orange;
  margin: 5px;
}
```

seite.html

```
<!DOCTYPE html>
<html>
  [...]

  <body>
    <div id="container1">
      <div>Flex-Item 1</div>
      <div>Flex-Item 2</div>
      <div>Flex-Item 3</div>
    </div>
    <div id="container2">
      <div>Flex-Item 1</div>
      <div>Flex-Item 2</div>
      <div>Flex-Item 3</div>
    </div>
  </body>
</html>
```



ZEILEN/SPALTEN UMBRECHEN

Flex-Items umbrechen: **flex-wrap**

- Wert `nowrap` (Standard): Flex-Items bleiben auf einer Zeile/Spalte
- Wert `wrap`: Flex-Items verteilen sich auf mehrere Zeilen/Spalten
- Wert `wrap-reverse`: wie `wrap`, jedoch werden die Zeilen/Spalten in umgekehrter Richtung angeordnet

flex-flow

Kurzschreibweise, kombiniert `flex-direction` und `flex-wrap`,
z.B. `flex-flow: column wrap;`

UMBRECHEN: BEISPIEL

style.css

```
#container {  
  display: flex;  
  flex-flow: row wrap;  
  background-color: darkorange;  
  padding: 10px;  
  margin: 5px;  
  height: 100px;  
}  
  
div div {  
  border: 2px solid;  
  background-color: orange;  
  margin: 5px;  
}
```

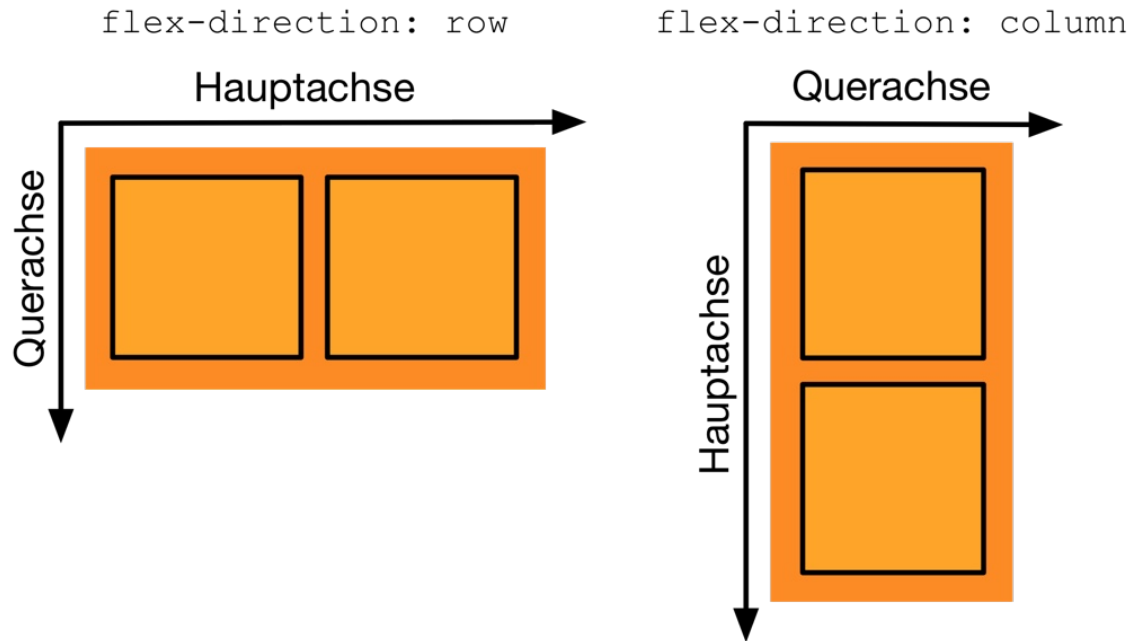
seite.html

```
<!DOCTYPE html>  
<html>  
  [...]   
  
  <body>  
    <div id="container">  
      <div>Flex-Item 1</div>  
      <div>Flex-Item 2</div>  
      <div>Flex-Item 3</div>  
      <div>Flex-Item 4</div>  
      <div>Flex-Item 5</div>  
      <div>Flex-Item 6</div>  
    </div>  
  </body>  
</html>
```



FLEX-ITEMS ANORDNEN

- Innerhalb des Flex-Containers wird zwischen der Hauptachse (*main axis*) und der Querachse (*cross axis*) unterschieden
- Der Verlauf der Achsen hängt von der eingestellten `flex-direction` ab:



FLEX-ITEMS ANORDNEN (2)

justify-content

Legt fest, wie Flex-Items entlang der Hauptachse angeordnet werden

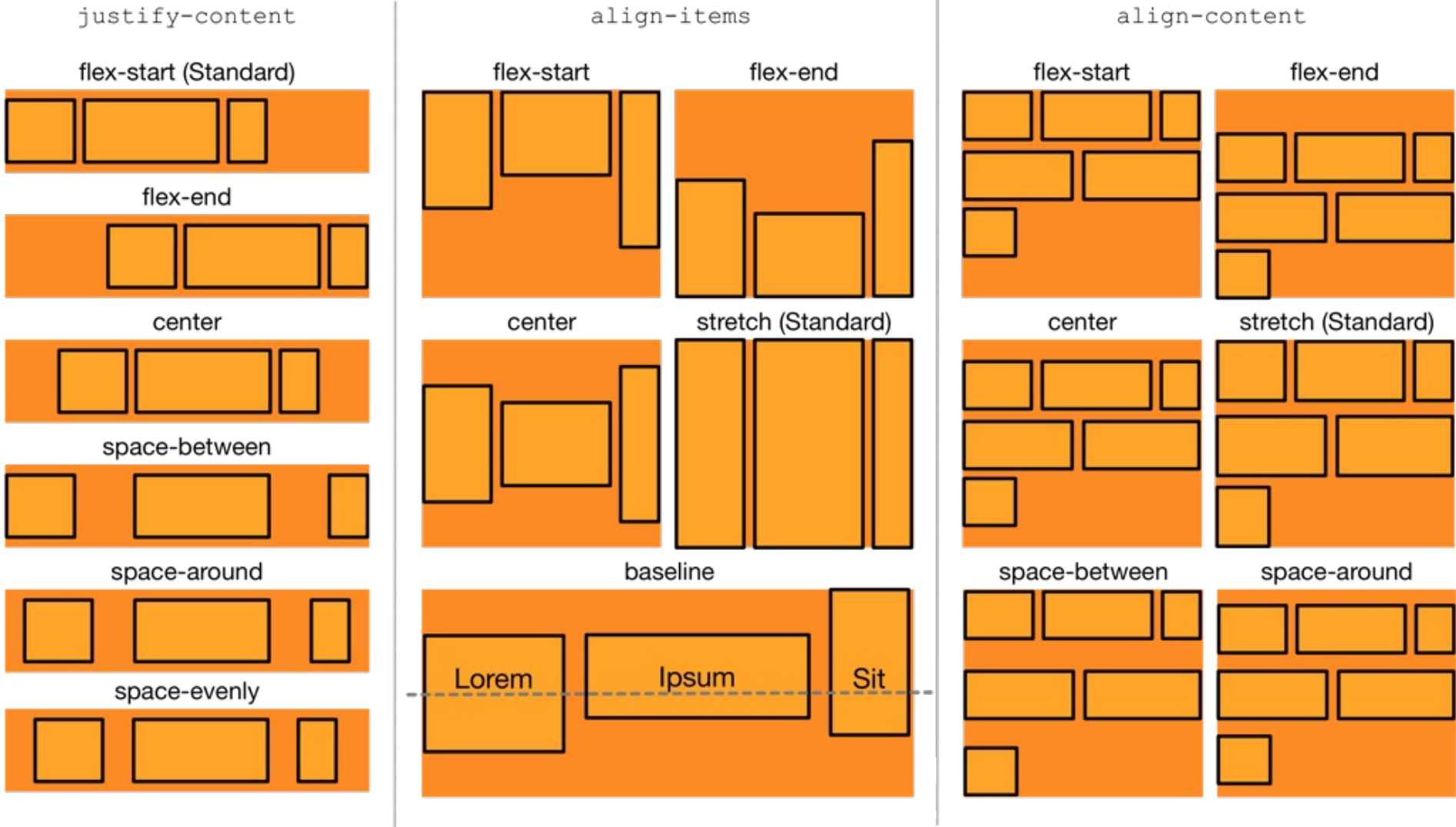
align-items

Legt fest, wie Flex-Items entlang der Querachse der aktuellen Zeile/Spalte angeordnet werden

align-content

Legt fest, wie Flex-Items entlang der Querachse angeordnet werden (nur bei mehreren Zeilen/Spalten)

Flex-Items anordnen (exemplarisch für `flex-direction: row`):



Darstellung nach: [A Complete Guide to Flexbox](#)

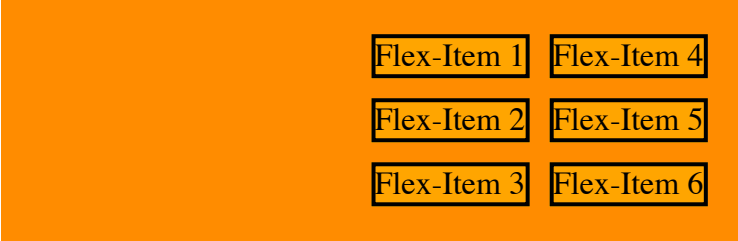
FLEX-ITEMS ANORDNEN: BEISPIEL

style.css

```
#container {  
  display: flex;  
  flex-flow: column wrap;  
  justify-content: center;  
  align-content: flex-end;  
  background-color: darkorange;  
  padding: 10px;  
  margin: 5px;  
  height: 100px;  
}  
  
div div {  
  border: 2px solid;  
  background-color: orange;  
  margin: 5px;  
}
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]   
  
  <body>  
    <div id="container">  
      <div>Flex-Item 1</div>  
      <div>Flex-Item 2</div>  
      <div>Flex-Item 3</div>  
      <div>Flex-Item 4</div>  
      <div>Flex-Item 5</div>  
      <div>Flex-Item 6</div>  
    </div>  
  </body>  
</html>
```



Flex-Item 1 Flex-Item 4
Flex-Item 2 Flex-Item 5
Flex-Item 3 Flex-Item 6

FLEXIBILITÄT DER FLEX-ITEMS

- Der Grad der Flexibilität kann für einzelne Flex-Items gezielt festgelegt werden
 - Dies ist insbesondere wichtig zum Umgang mit verschiedenen Größenverhältnissen (z.B. Zoom, unterschiedliche Anzeigegrößen und -formate)
- ! Alle bisherigen Eigenschaften wurden am Flex-Container notiert - die folgenden Eigenschaften beziehen sich jedoch auf Flex-Items.

FLEXIBILITÄT DER FLEX-ITEMS (2)

flex-grow

- Wachstumsfaktor als numerischer Wert ohne Einheit (Standard: 0)
- Legt fest, wie stark das Flex-Items relativ zu den anderen Flex-Items wachsen soll

flex-shrink

- Schrumpffaktor als numerischer Wert ohne Einheit (Standard: 1)
- Legt fest, wie stark das Flex-Items relativ zu den anderen Flex-Items schrumpfen soll

FLEXIBILITÄT DER FLEX-ITEMS (3)

flex-basis

- Basisbreite für das Flex-Item als numerischer Wert
- Standardwert ist auto (flexible Anpassung der Breite)

flex

- Kurzschreibweise, kombiniert `flex-grow`, `flex-shrink` und `flex-basis`, z.B. `flex: 2 1 auto;`
- Standardwert: `0 1 auto`

FLEXIBILITÄT DER FLEX-ITEMS: BEISPIEL

style.css

```
#container {
  display: flex;
  flex-flow: row wrap;
  background-color: darkorange;
  padding: 10px;
  margin: 5px;
  height: 100px;
}

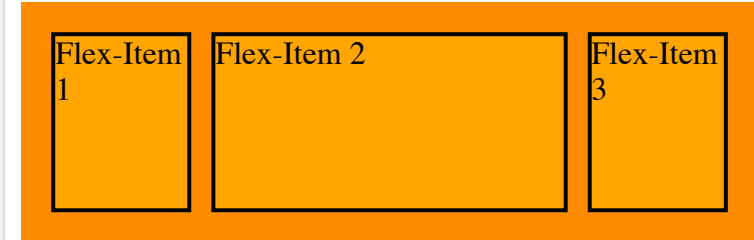
div div {
  border: 2px solid;
  background-color: orange;
  margin: 5px;
}

/* Schrumpf- und Wachstumsfaktor 1, automatische Breite */
#flex1, #flex3 { flex: 1 1 auto; }
/* Element wächst viermal so stark wie die anderen Elemente
   und schrumpft nicht, Basisbreite 50% */
#flex2 { flex: 4 0 50%; }
```

seite.html

```
<!DOCTYPE html>
<html>
  [...]

  <body>
    <div id="container">
      <div id="flex1">Flex-Item 1</div>
      <div id="flex2">Flex-Item 2</div>
      <div id="flex3">Flex-Item 3</div>
    </div>
  </body>
</html>
```



LAYOUTS MIT FLEXBOX

Vorteile, insbesondere gegenüber `float`, z.B.:

- ➕ Flexbox vereinfacht die Erstellung von Layouts, die flexibel auf verschiedene Bildschirmgrößen und Größenänderungen reagieren können
- ➕ Bei spaltenbasierten Layouts sind Spalten mit gleicher Höhe kein Problem
- ➕ Flexbox ermöglicht das vertikale Zentrieren von Elementen (und viele weitere Anordnungen)

Studieninteressierte
Studierende
Promovierende
Presse
Forschung und Transfer
IDIAL
Hochschule
Jobs & Karriere

Navigationsmenü



Hauptinhalt

Hochschulrat
Ministerin übergab Ernennungsurkunden
Tag der offenen Tür
Besuchen Sie uns am 5. Mai
Spring School
Studierende bauen kulturelle Brücken
TV-Projekt
KOMM AN! Der Migrations-Talk auf nrwision
Neubau
Lernen im Lichthof
TalentKolleg Ruhr
Neue App „Mathematik Vorkurs“ gestartet
Frühstücksfrühstück
Einladung ins Rathaus
Studieninteressierte
Freie Plätze in Master-Studiengängen
IT-Mittelstandsstipendium
Förderung für 35 neue Stipendiat*innen
Dortmunder Wirtschaftspreis
FH-Alumnus als Unternehmer erfolgreich
News
Weitere aktuelle Beiträge

Eingeloggt als Sven Jörges.

TAG DER
OFFENEN TÜR
5. MAI 2018

BildungsOffensive

Erstsemester
FHDO hilft

„Zusatz- informationen“

Studienplatzportal

FH DORTMUND

Über uns
Campus Emil-Figge-Straße
Campus Max-Ophüls-Platz
Campus Sonnenstraße
Jobs & Karriere

BILDUNGSANGEBOTE

Bachelor Studium
Master Studium
Duales Studium
Berufsbegleitendes Studium

SERVICE

Studienberatung
Studienleistungen
IT-Dienste
Redaktion
Hochschule intern

KONTAKT

Personen
Presse
VIA* Feedback
Ersthelfer
Impressum & Datenschutz

SOCIAL MEDIA



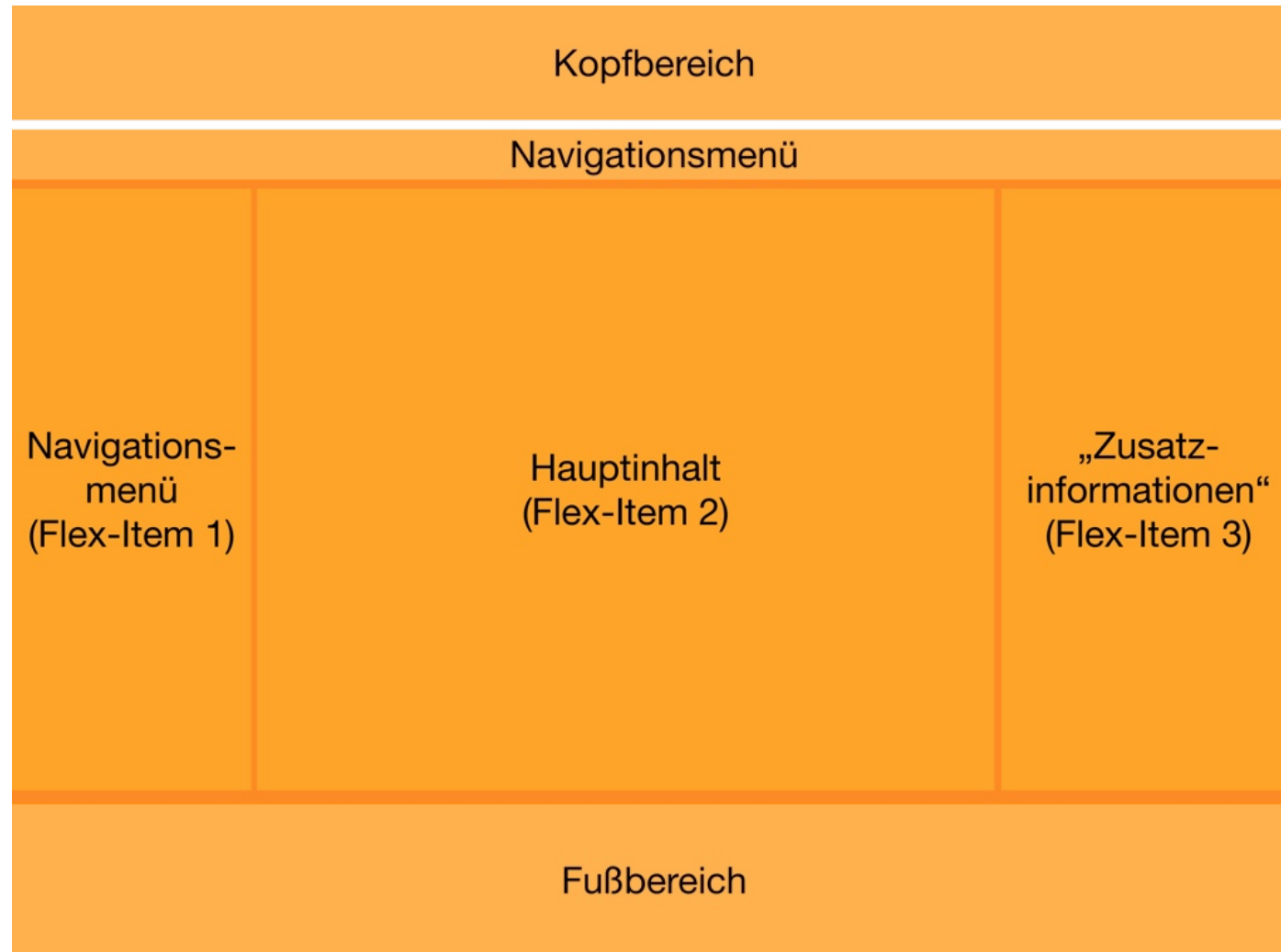
Fußbereich

Kopfbereich

Navigationsmenü

Flex-Container
(`flex-direction: row`)

Fußbereich



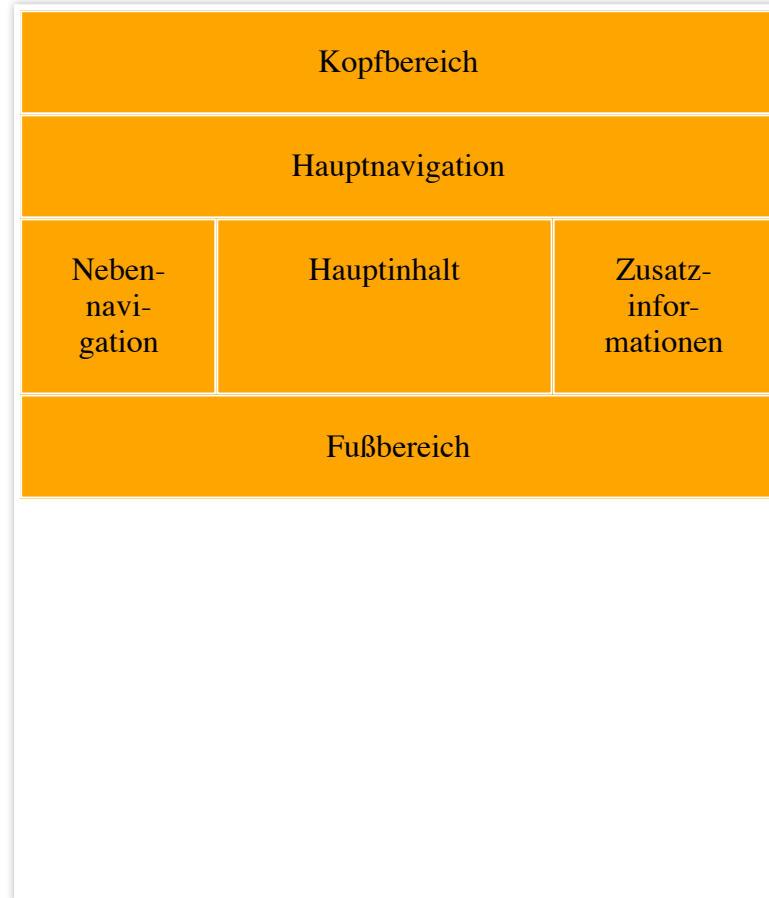
BEISPIEL: 3-SPALTEN-LAYOUT MIT FLEXBOX

style.css

```
header, nav, main, aside, footer {  
  border: 1px solid white;  
  background-color: orange;  
  padding: 1em;  
  text-align: center;  
}  
  
#container { display: flex; }  
  
main { flex: 3 60%; }
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
    <header>Kopfbereich</header>  
    <nav>Hauptnavigation</nav>  
    <div id="container">  
      <nav id="nebenNav">Nebennavigation</nav>  
      <main>Hauptinhalt</main>  
      <aside>Zusatzinformationen</aside>  
    </div>  
    <footer>Fußbereich</footer>  
  </body>  
</html>
```



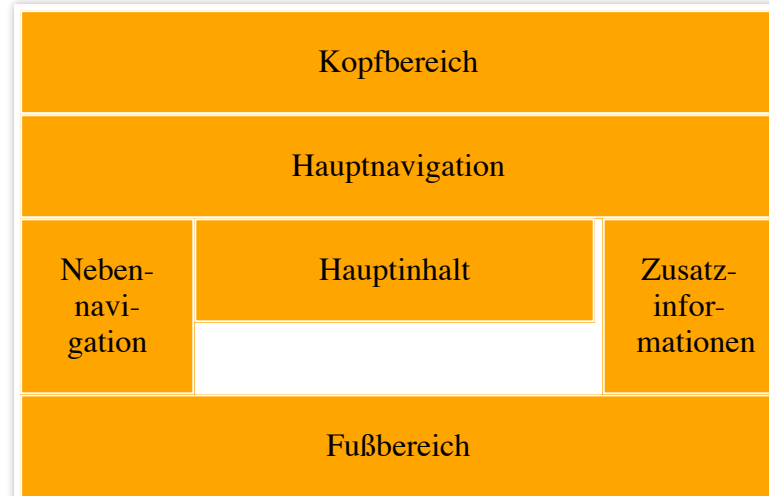
ZUM VERGLEICH: 3-SPALTEN-LAYOUT MIT FLOAT

style.css

```
header, nav, main, aside, footer {  
  border: 1px solid white;  
  background-color: orange;  
  padding: 1em;  
  text-align: center;  
}  
  
#nebenNav, main { float: left; }  
  
#nebenNav { width: 20%; }  
  
main { width: 55%; }  
  
aside {  
  float: right;  
  width: 25%;  
}  
  
footer { clear: both; }
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
    <header>Kopfbereich</header>  
    <nav>Hauptnavigation</nav>  
    <nav id="nebenNav">Nebennavigation</nav>  
    <main>Hauptinhalt</main>  
    <aside>Zusatzinformationen</aside>  
    <footer>Fußbereich</footer>  
  </body>  
</html>
```



VON FLEXIBLEN BOXEN ZU RASTERN...

- Flexbox ist ein **eindimensionales** Modell - Elemente werden in Zeilen oder Spalten positioniert

❓ Was passiert, wenn unser Layout komplexer wird?

Studieninteressierte
Studierende
Promovierende
Presse
Forschung und Transfer
IDIAL
Hochschule
Jobs & Karriere

Navigationsmenü



Hauptinhalt

Hochschulrat
Ministerin übergab Ernennungsurkunden
Tag der offenen Tür
Besuchen Sie uns am 5. Mai
Spring School
Studierende bauen kulturelle Brücken
TV-Projekt
KOMM AN! Der Migrations-Talk auf nrwision
Neubau
Lernen im Lichthof
TalentKolleg Ruhr
Neue App „Mathematik Vorkurs“ gestartet
Frühstücksfrühstück
Einladung ins Rathaus
Studieninteressierte
Freie Plätze in Master-Studiengängen
IT-Mittelstandsstipendium
Förderung für 35 neue Stipendiat*innen
Dortmunder Wirtschaftspreis
FH-Alumnus als Unternehmer erfolgreich
News
Weitere aktuelle Beiträge

Eingeloggt als Sven Jörges.

TAG DER
OFFENEN TÜR
5. MAI 2018

BildungsOffensive

Erstmaliger
FHDO hilft

„Zusatz- informationen“

Studienplatzportal

FH DORTMUND

Über uns
Campus Emil-Figge-Straße
Campus Max-Ophüls-Platz
Campus Sonnenstraße
Jobs & Karriere

BILDUNGSANGEBOTE

Bachelor Studium
Master Studium
Duales Studium
Berufsbegleitendes Studium

SERVICE

Studienberatung
Studienleistungen
IT-Dienste
Redaktion
Hochschule intern

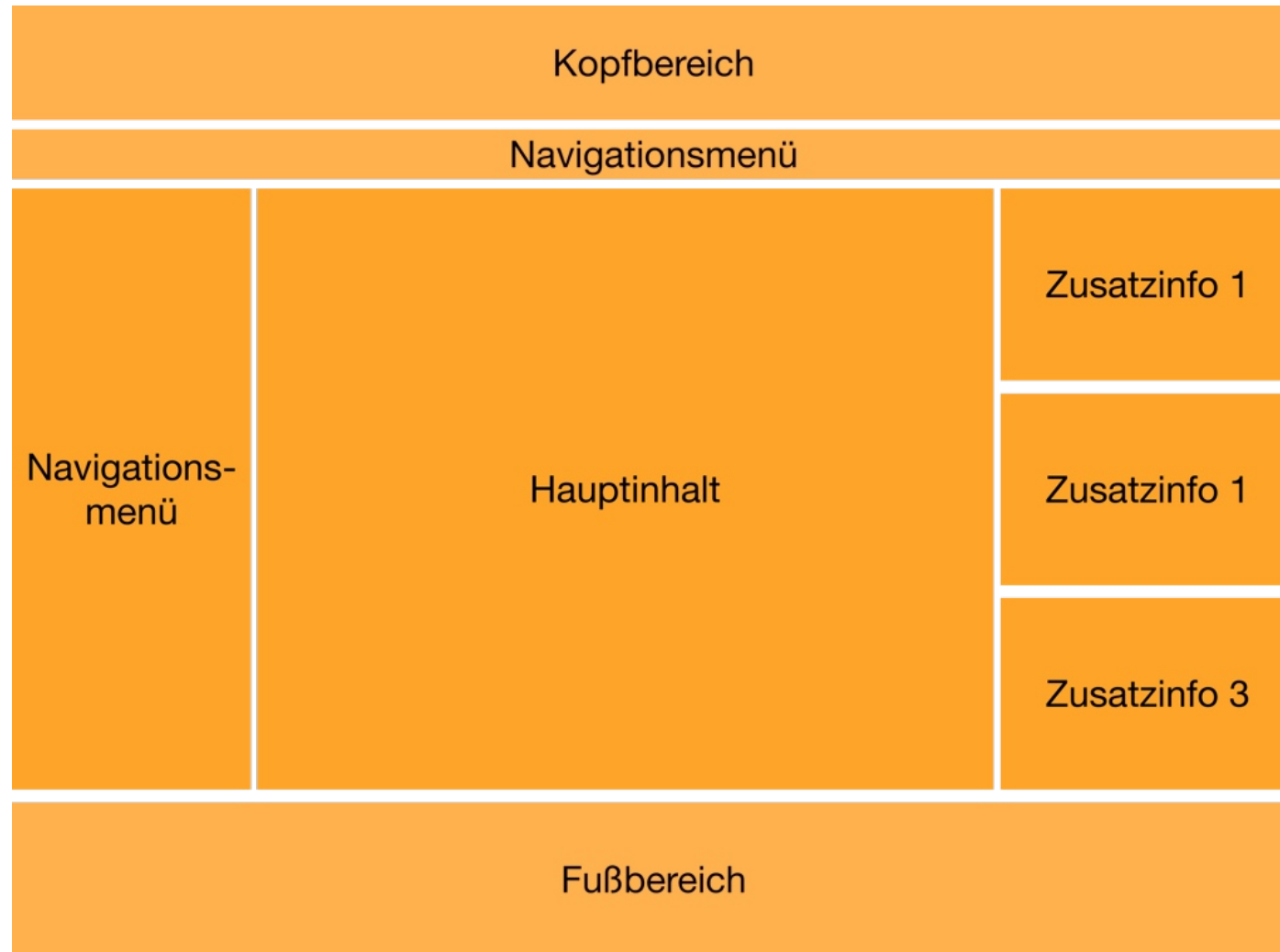
KONTAKT

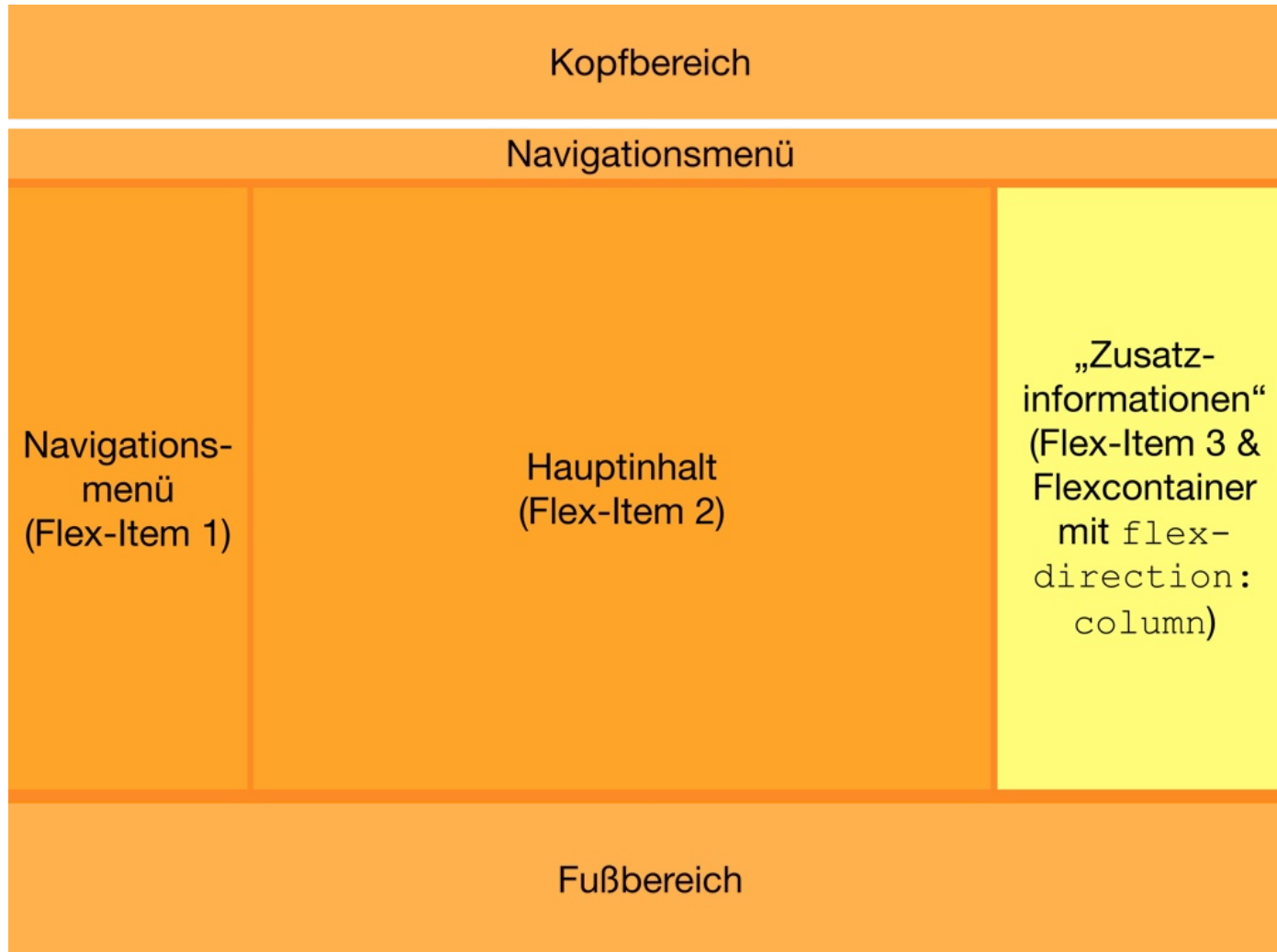
Personen
Presse
VIA* Feedback
Ersthelfer
Impressum & Datenschutz

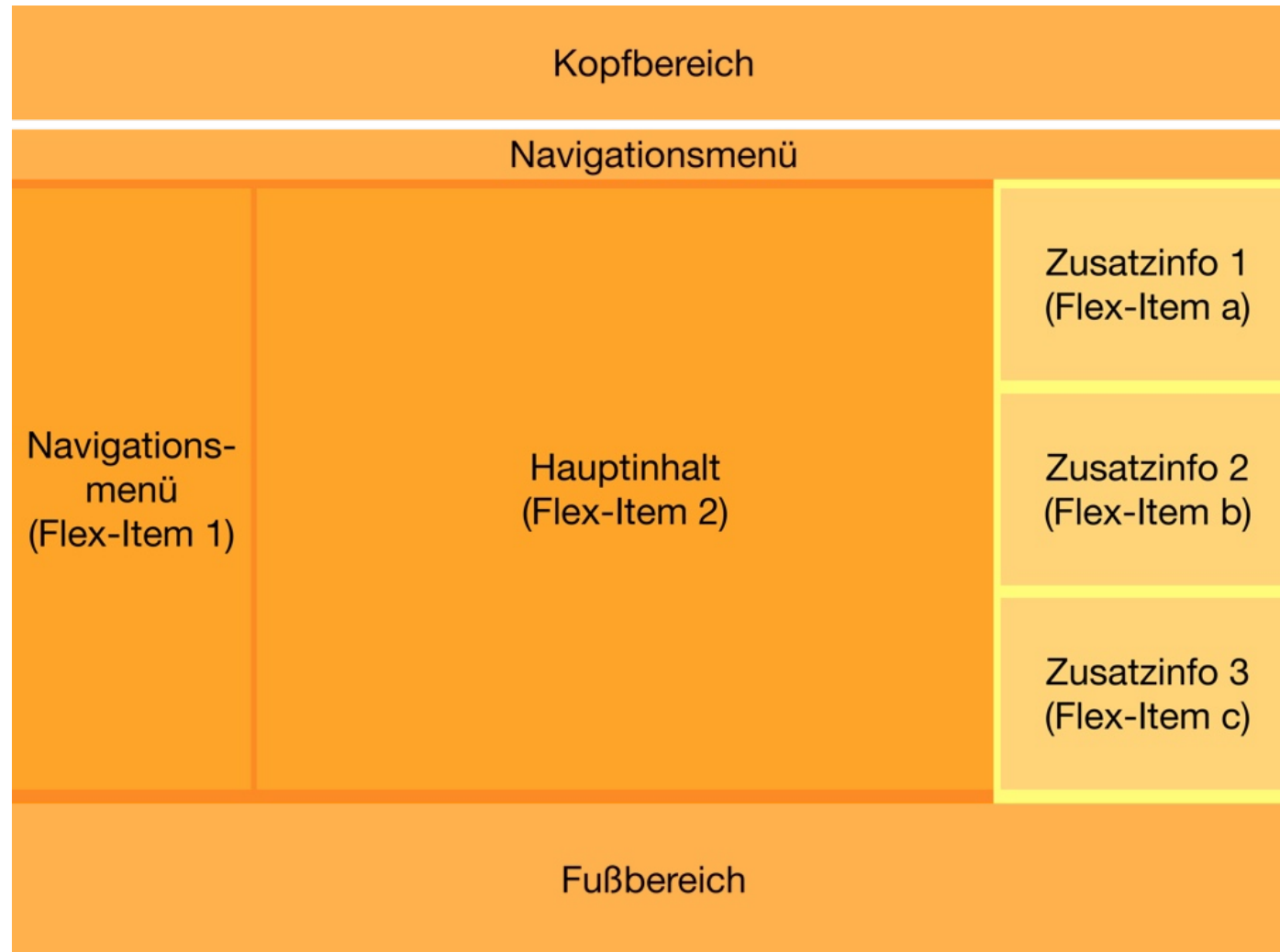
SOCIAL MEDIA



Fußbereich





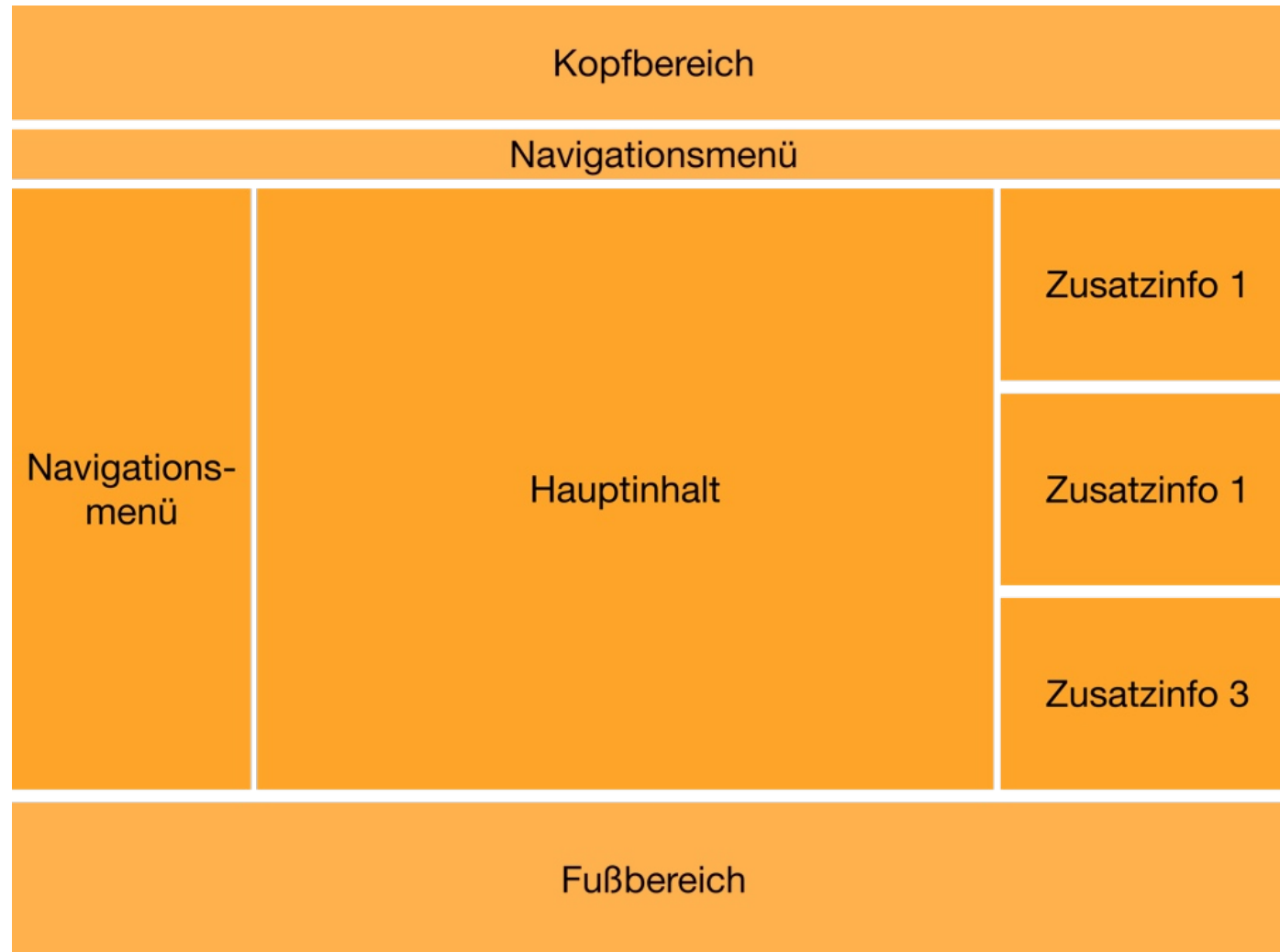


VON FLEXIBLEN BOXEN ZU RASTERN...

- Flexbox ist ein **eindimensionales** Modell - Elemente werden in Zeilen oder Spalten positioniert
- Komplexere Layouts kann man z.B. über verschachtelte Flex-Container aufbauen
- Mit dem Grid-Modell bietet CSS hier eine weitere Alternative

POSITIONIERUNG VON ELEMENTEN

- CSS bietet mehrere Möglichkeiten, die Positionierung und Anordnung von Elementen zu bestimmen
- Wir betrachten im Folgenden:
 1. Positionierungsmodell (`position`)
 2. Float-Modell
 3. Flexbox
 4. Grid



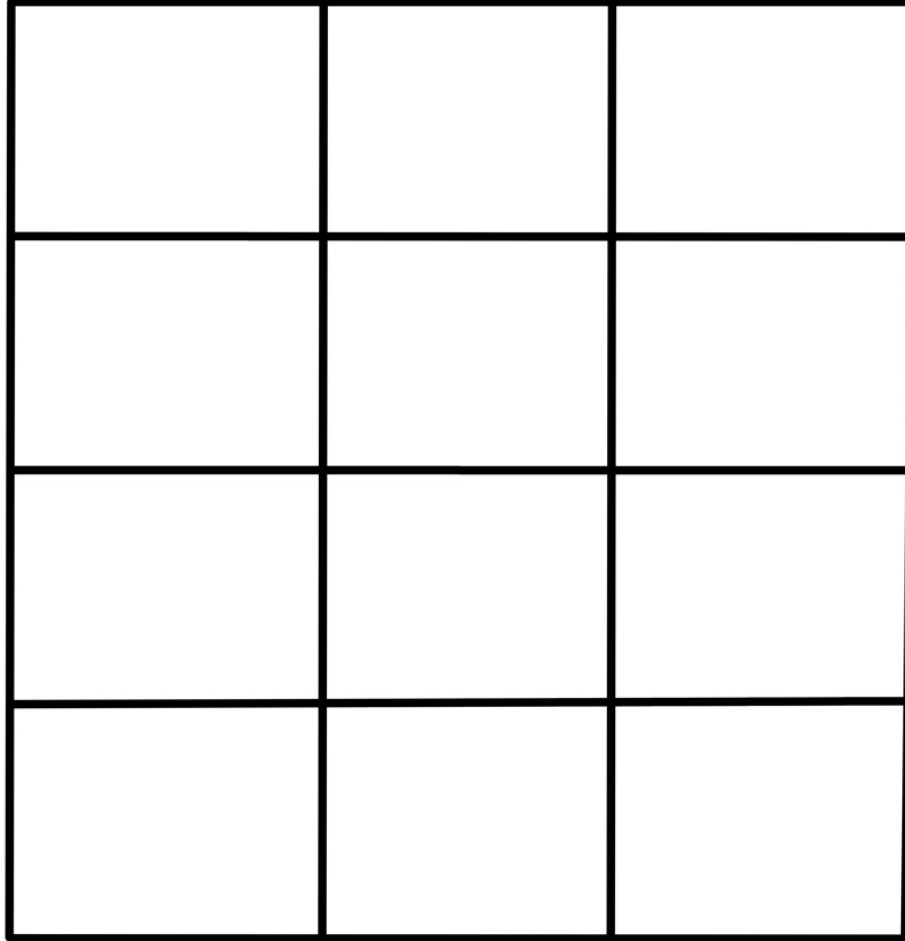
	Kopfbereich	
	Navigationsmenü	
		Zusatzinfo 1
Navigationsmenü	Hauptinhalt	Zusatzinfo 1
		Zusatzinfo 3
	Fußbereich	

	Spalte 1	Spalte 2	Spalte 3
Zeile 1		Kopfbereich	
Zeile 2		Navigationsmenü	
Zeile 3			Zusatzinfo 1
Zeile 4	Navigationsmenü	Hauptinhalt	Zusatzinfo 1
Zeile 5			Zusatzinfo 3
Zeile 6		Fußbereich	

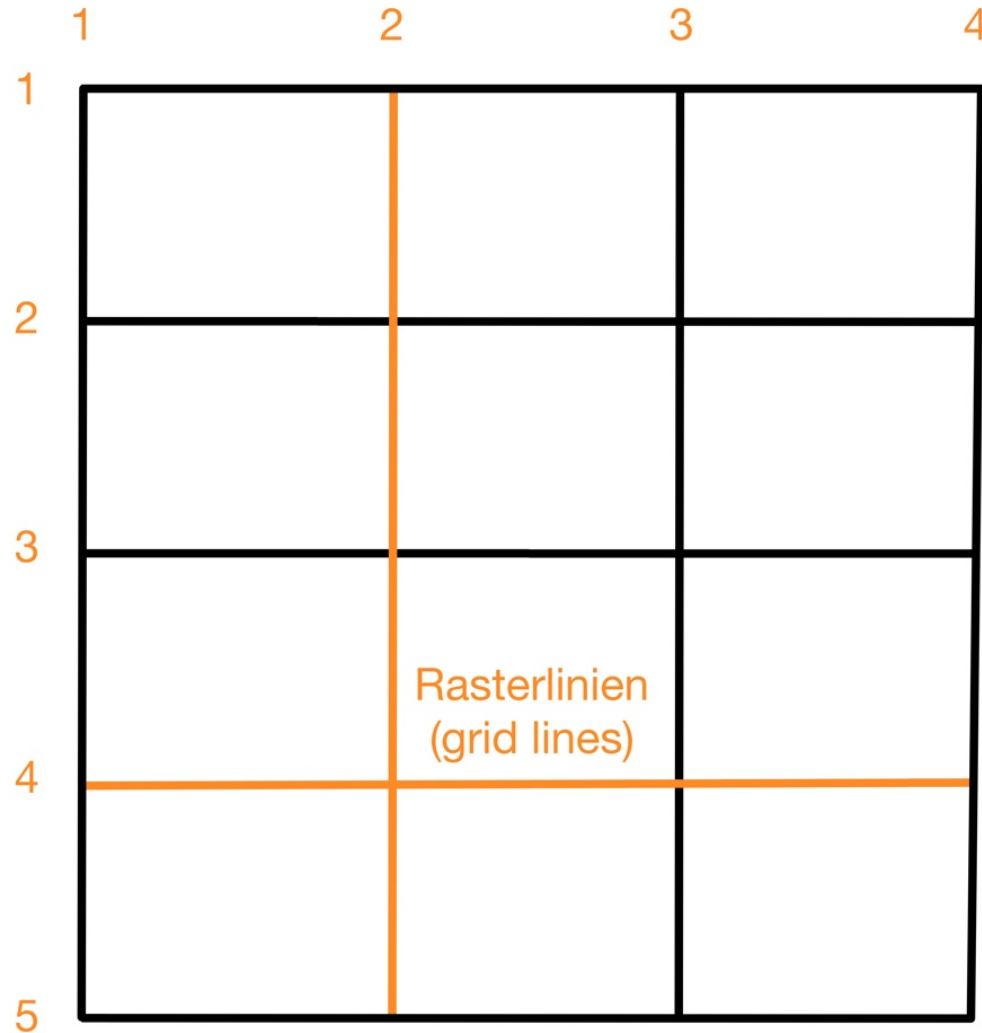
GRID

- **Grid** ist ein [CSS-Modul](#) zur Anordnung und Positionierung von Elementen in Form eines **Rasters**
- Grundprinzip:
 - Wie Flexbox benötigt Grid ein Elternelement, den **Grid-Container**
 - Die im Grid-Container enthaltenen Kindelemente, die **Grid-Items**, werden vom Container im Raster platziert
 - Im Gegensatz zu Flexbox ist Grid ein **zweidimensionales** Modell - das Raster besteht aus Zeilen *und* Spalten
- ! Grid ist ein komplexes und mächtiges Modul - daher im Folgenden ein Kurzüberblick.

AUFBAU DES RASTERS



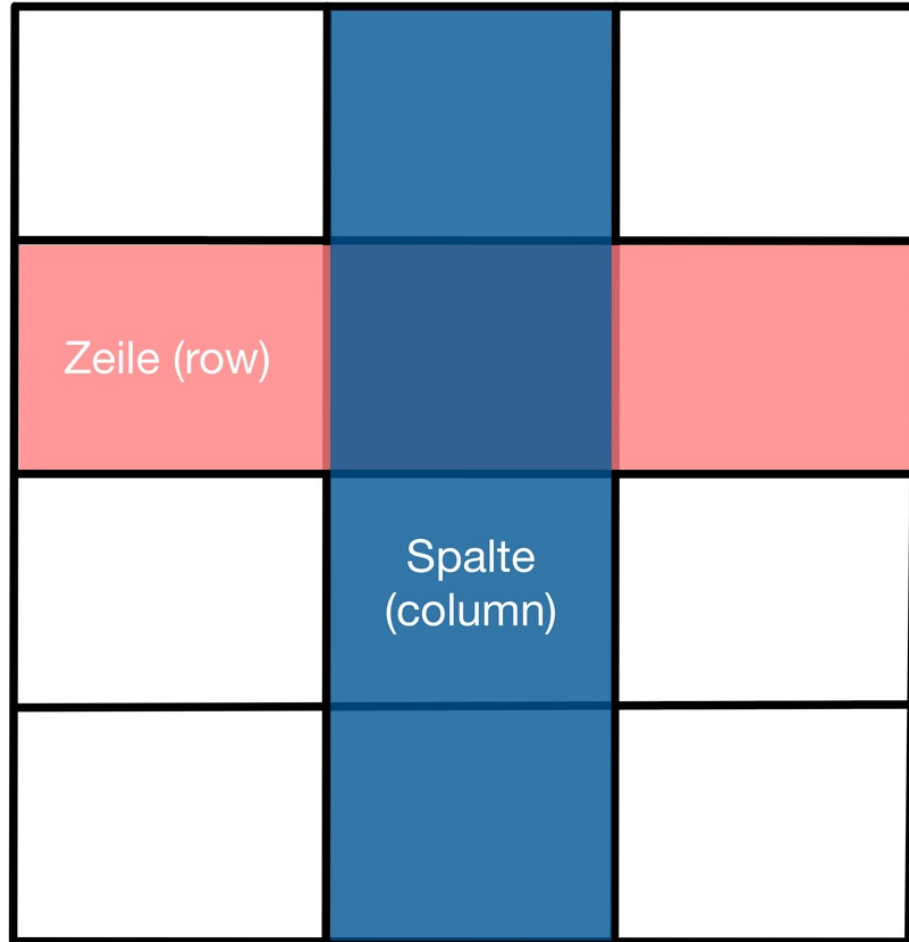
AUFBAU DES RASTERS



AUFBAU DES RASTERS

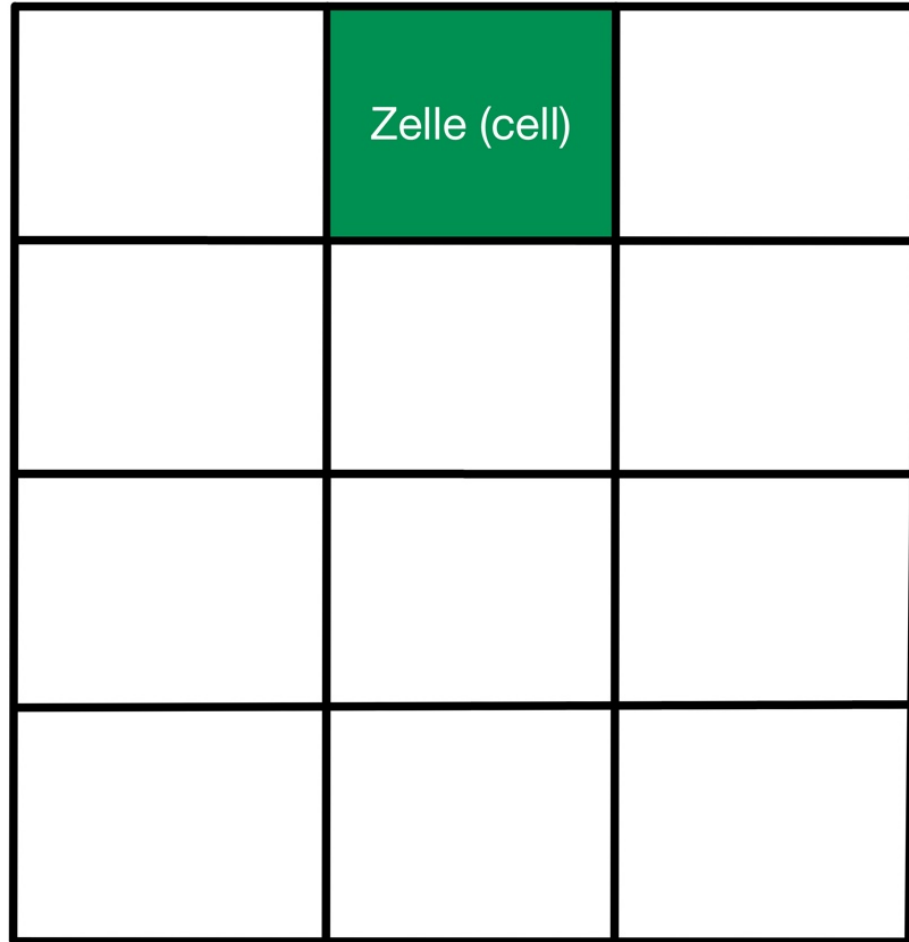
Zeile (row)		

AUFBAU DES RASTERS



Oberbegriff für Zeilen und Spalten: *grid tracks*

AUFBAU DES RASTERS



AUFBAU DES RASTERS



GRID-CONTAINER

Grid-Container definieren

Ein Element wird zum Grid-Container durch
Setzen der Eigenschaft `display: grid`

- ! Ohne weitere Einstellungen wird jedes Kindelement standardmäßig in eine eigene Zeile platziert.

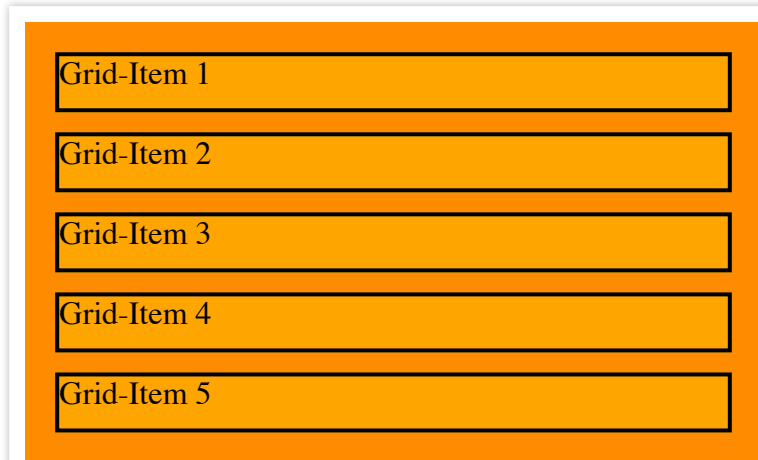
GRID-CONTAINER: BEISPIEL

style.css

```
#container {  
  display: grid;  
  background-color: darkorange;  
  padding: 10px;  
  margin: 5px;  
  height: 200px;  
}  
  
#container > div {  
  border: 2px solid;  
  background-color: orange;  
  margin: 5px;  
}
```

seite.html

```
<!DOCTYPE html>  
<html>  
  [...]  
  
  <body>  
    <div id="container">  
      <div>Grid-Item 1</div>  
      <div>Grid-Item 2</div>  
      <div>Grid-Item 3</div>  
      <div>Grid-Item 4</div>  
      <div>Grid-Item 5</div>  
    </div>  
  </body>  
</html>
```



RASTERSTRUKTUR FESTLEGEN

Durch die Angabe von Vorlagen (*templates*) kann die Struktur des Rasters festgelegt werden

Eigenschaften **grid-template-rows** / **grid-template-columns**

- Definieren die Zeilen/Spalten des Rasters als durch Leerzeichen getrennte Liste
- Jeder Eintrag in der Liste definiert eine Zeile/Spalte
- Der Wert jedes Eintrags repräsentiert die Höhe/Breite der Zeile/Spalte (*track size*)
- Mögliche Werte sind u.A.:
 - Numerische Werte
 - Schlüsselwort `auto` (Höhe/Breite passen sich gemäß Inhalt und verfügbarem Platz an)

RASTERSTRUKTUR: BEISPIEL

style.css

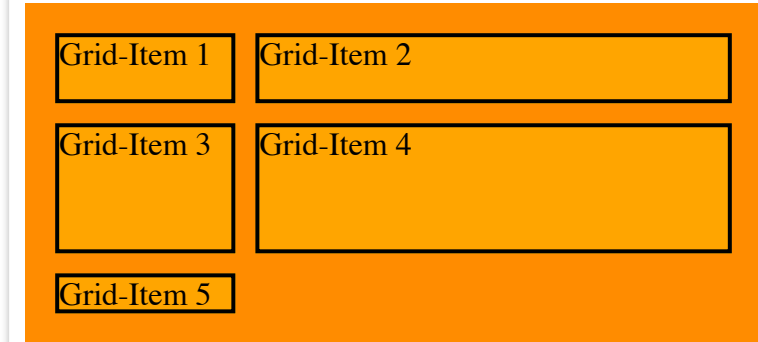
```
#container {
  display: grid;
  /* Zwei Spalten mit folgenden Breiten definieren:
     Erste Spalte: 100px, Zweite Spalte: Automatische Breite */
  grid-template-columns: 100px auto;
  /* Drei Zeilen mit folgenden Höhen definieren:
     Erste Zeile: Automatische Höhe, Zweite Zeile: 50%,
     Dritte Zeile: 30px */
  grid-template-rows: auto 50% 30px;
  background-color: darkorange;
  padding: 10px;
  margin: 5px;
  height: 200px;
}

#container > div {
  border: 2px solid;
  background-color: orange;
  margin: 5px;
}
```

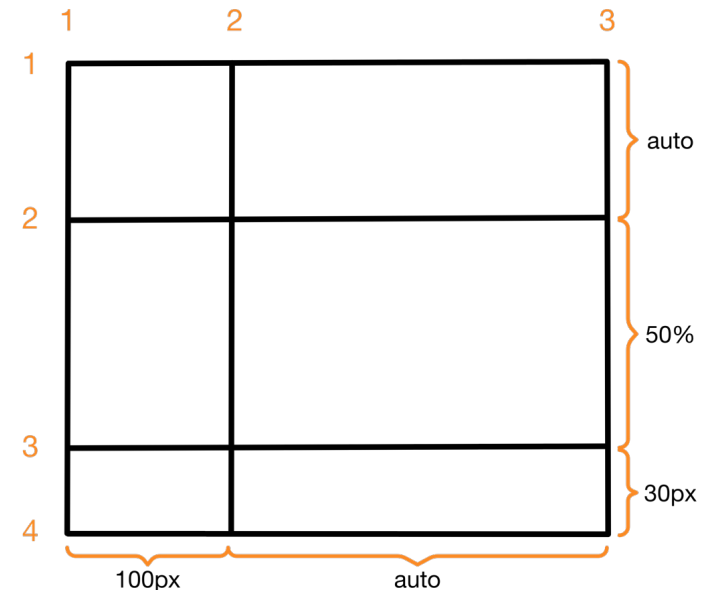
seite.html

```
<!DOCTYPE html>
<html>
  [...]

  <body>
    <div id="container">
      <div>Grid-Item 1</div>
      <div>Grid-Item 2</div>
      <div>Grid-Item 3</div>
      <div>Grid-Item 4</div>
      <div>Grid-Item 5</div>
    </div>
  </body>
</html>
```



Entstehendes Raster:



BENANNTE BEREICHE

Eigenschaft **grid-template-areas**

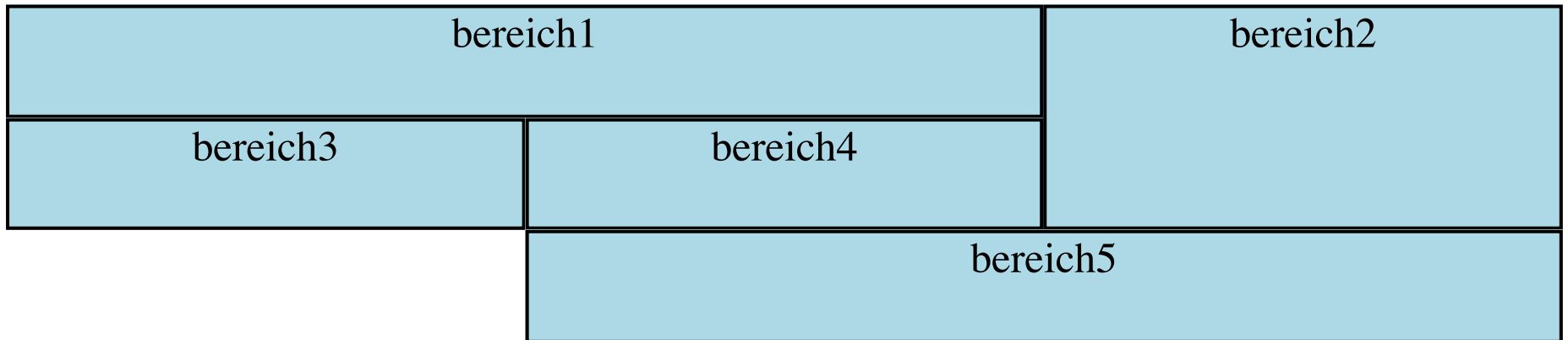
- Erlaubt die Definition benannter Bereiche (*areas*) im Grid-Container
- Die Syntax für den Wert dieser Eigenschaft visualisiert die Rasterstruktur:
 - Der Wert besteht aus einer Liste von Zeichenketten, wobei jede Zeichenkette eine Zeile des Rasters beschreibt
 - Jede Zeichenkette ist wiederum eine durch Leerzeichen getrennte Liste, wobei jeder Eintrag in der Liste einer Zelle des Rasters entspricht
 - Eine einzelne Zelle bekommt dabei entweder einen Bereichsnamen zugewiesen oder wird durch Notation eines Punktes (.) als leere Zelle deklariert
 - Eine Wiederholung von Bereichsnamen bedeutet dabei, dass der Bereich sich über mehrere Zellen erstreckt

BENANNTE BEREICHE: BEISPIEL

Definition des Rasters:

```
#container {  
  grid-template-areas: "bereich1 Bereich1 Bereich2"  
                      "bereich3 Bereich4 Bereich2"  
                      ".      Bereich5 Bereich5";  
}
```

Resultierendes Raster mit Bereichen:



BENANNTE BEREICHE: ELEMENTE PLATZIEREN

Eigenschaft `grid-area`

- Mittels `grid-area` wird ein Grid-Item einem Bereich zugewiesen
- Wert ist ein Bereichsname, der in der Definition des Rasters mit `grid-template-areas` festgelegt wurde

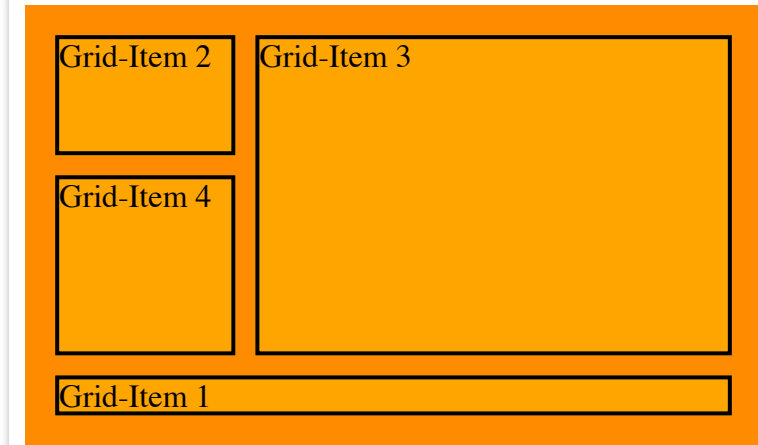
BENANNTE BEREICHE: CODE-BEISPIEL

style.css

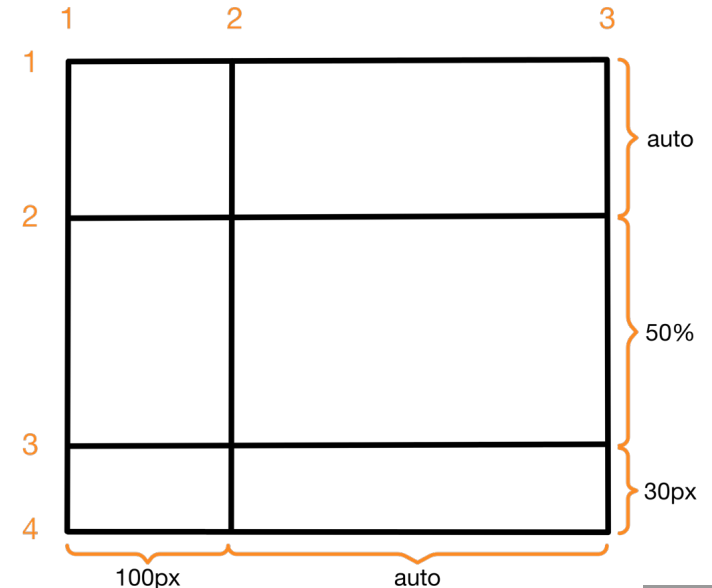
```
#container {  
  display: grid;  
  grid-template-columns: 100px auto;  
  grid-template-rows: auto 50% 30px;  
  grid-template-areas: "area2 area3"  
                      "area4 area3"  
                      "area1 area1";  
  background-color: darkorange;  
  padding: 10px;  
  margin: 5px;  
  height: 200px;  
}  
  
#container > div {  
  border: 2px solid;  
  background-color: orange;  
  margin: 5px;  
}  
  
#item1 { grid-area: area1; }  
#item2 { grid-area: area2; }  
#item3 { grid-area: area3; }  
#item4 { grid-area: area4; }
```

seite.html

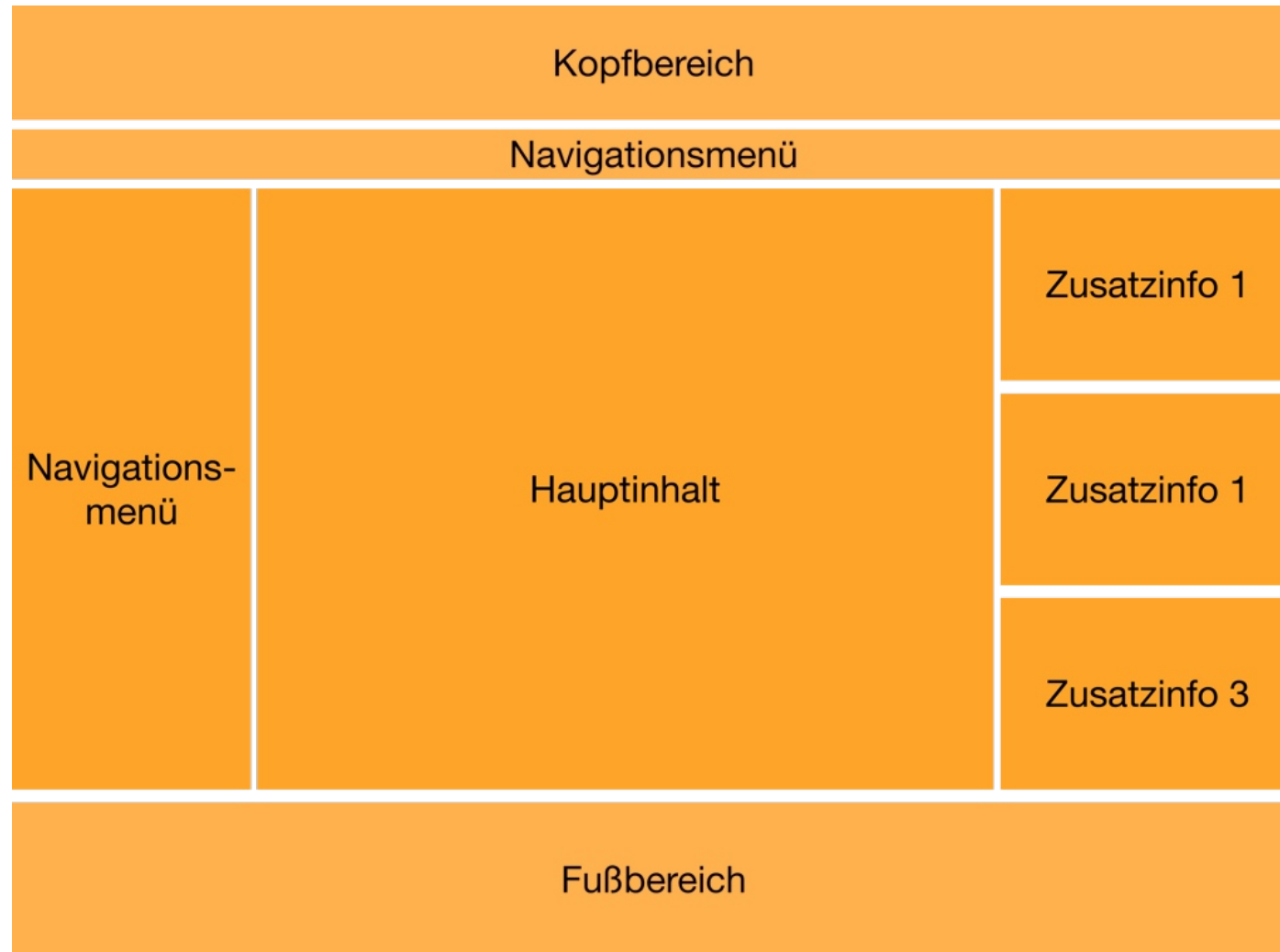
```
<!DOCTYPE html>  
<html>  
  [...]  
  <body>  
    <div id="container">  
      <div id="item1">Grid-Item 1</div>  
      <div id="item2">Grid-Item 2</div>  
      <div id="item3">Grid-Item 3</div>  
      <div id="item4">Grid-Item 4</div>  
    </div>  
  </body>  
</html>
```



Entstehendes Raster:



BEISPIEL-LAYOUT MIT GRID



	Kopfbereich	
	Navigationsmenü	
		Zusatzinfo 1
Navigations- menü	Hauptinhalt	Zusatzinfo 1
		Zusatzinfo 3
	Fußbereich	

BEISPIEL-LAYOUT MIT GRID

style.css

```
header, nav, main, aside, footer {
  border: 1px solid white;
  background-color: orange;
  padding: 1em;
  text-align: center;
}

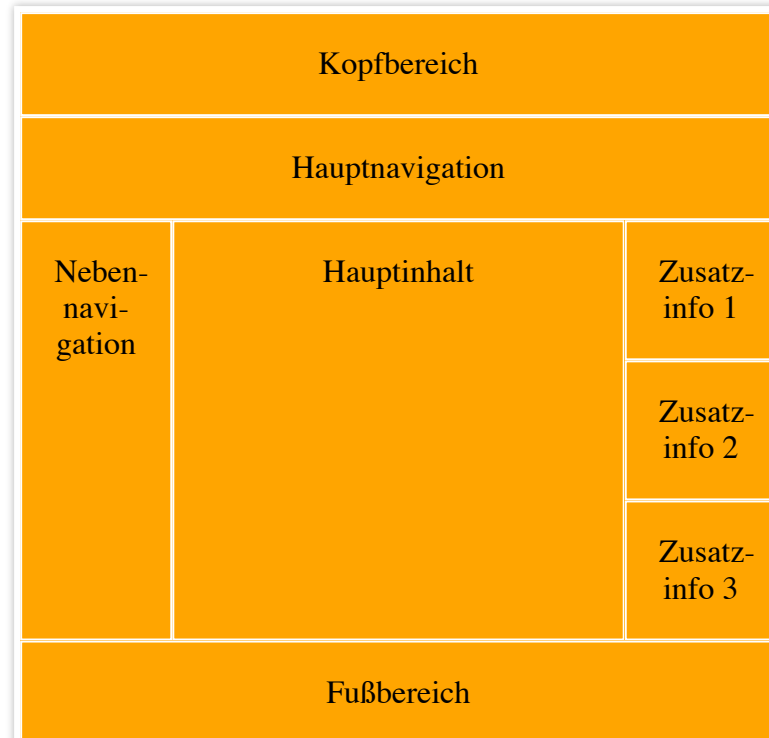
#container {
  display: grid;
  grid-template-areas: "header header header"
    "mainNav mainNav mainNav"
    "nebenNav haupt aside1"
    "nebenNav haupt aside2"
    "nebenNav haupt aside3"
    "footer footer footer";
  grid-template-columns: 20% 60% 20%;
}

header { grid-area: header; }
nav { grid-area: mainNav; }
main { grid-area: haupt; }
#nebenNav { grid-area: nebenNav; }
#zusatz1 { grid-area: aside1; }
#zusatz2 { grid-area: aside2; }
#zusatz3 { grid-area: aside3; }
footer { grid-area: footer; }
```

seite.html

```
<!DOCTYPE html>
<html>
  [...]

  <body>
    <div id="container">
      <header>Kopfbereich</header>
      <nav>Hauptnavigation</nav>
      <nav id="nebenNav">Nebennavigation</nav>
      <main>Hauptinhalt</main>
      <aside id="zusatz1">Zusatzinfo 1</aside>
      <aside id="zusatz2">Zusatzinfo 2</aside>
      <aside id="zusatz3">Zusatzinfo 3</aside>
      <footer>Fußbereich</footer>
    </div>
  </body>
</html>
```



GRID: VIELE WEITERE MÖGLICHKEITEN...

Grid bietet viele weitere Funktionen, um komplexe Layouts zu konstruieren, z.B.:

- Automatische Erweiterung des definierten Rasters (z.B. wenn es mehr Grid-Items als definierte Zellen gibt)
- Mechanismen zur Konfiguration der Anordnung von Grid-Items (ähnlich zu Flexbox)
- Benannte Rasterlinien
- Ab Level 2: Unterraster (*subgrids*)

MEHR ZU FLEXBOX:

[CSS Flexible Box Layout Module Level 1](#) 

Animationen: [Flexbox Playground](#) 

MEHR ZU GRID:

[CSS Grid Layout Module Level 1](#) 