# Kapitel 15 - Teil 4

## *Flexbox und Gridlayout*

Inhalte in diesem Kapitel

Grundprinzip der Layouttechniken Flexbox und Gridlayout kennen lernen

Navigationsleisten mit Flexbox gestalten

Gridlayout für komplexe Layouts nutzen

### Flexbox für Layouts und Navigationsleisten

Flexbox Layout Module:

* + Elemente nebeneinander oder untereinander platzieren
  + Reihenfolge von Elementen verändern
  + Leerraum nach Belieben verteilen – soll er gleichmäßig zwischen die Elemente aufgeteilt werden oder sollen die Elemente ganz am Anfang oder am Ende angeordnet oder vielleicht zentriert werden
  + Ausmaße der Elemente flexibel bestimmen.

Geeignet für

* + Menüs in allen Variationen
  + Zur unkomplizierten Anordnung von Elementen nebeneinander, also für einfach Spaltenlayouts
  + Zur vertikalen und horizontalen Zentrierung

### Einstieg in Flexbox

Flexcontainer: umfassende Element, display: flex;

Flexitems: darin befindliche Elemente

Mit Flexbox nur direkte Kindelemente anordnen, nicht tiefer verschachtelte Elemente

flex-direction

* + Darstellung nebeneinander oder untereinander
  + flex-direction: row; -> default
  + flex-direction: column; -> von oben nach unten
  + flex-direction: column-reverse; -> von oben nach unten in umgekehrter Reihenfolge
  + flex-direction: row-reverse; -> von links nach rechts in umgekehrter Reihenfolge

### Flexbox-Konzepte

Hauptachse

* + Bezeichnet die Achse, auf der die Flexitems dargestellt werden
  + Veränderbar durch flex-direction

Querende Achse

### Elemente mit Flexbox ausrichten Leerraumverteilung auf der querenden Achse: align-items

flex-direction: row

alle Items gleich hohe Spalte

mit align-items beim Flexcontainer Ausrichtung auf der querenden Achse beeinflussen

* + align-items: flex-start; -> Elemente werden bei der Darstellung nebeneinander oben angeordnet
  + align-items: stretch; -> Elemente werden bei Darstellung nebeneinander gleich hoch (Standard)
  + align-items: center; -> Elemente werden mittig angeordnet
  + align-items: flex-end; -> Elemente werden bei der Darstellung nebeneinander unten angeordnet
  + align-items: baseline; -> Basislinie (Hauptlinie bei Schrift) wird zueinander angeordnet. Damit man eine Auswirkung sieht, muss die Schriftgröße unterschiedlich sein

um einzelnes Element anders auszurichten -> align-self beim Flexitem

Beispiele: /kapitel\_15/

* + flexbox-einstieg.html
  + flexbox-align-items.html
  + flexbox-align-items\_2.html
  + flexbox-align-items\_align\_self.html

### Leerraumverteilung auf der Hauptachse: justify-content

justify-content:

* + Regelung der Leerraumverteilung auf Hauptachse
  + justify-content: flex-start; -> Flexitems werden am Anfang des Flexcontainers angeordnet
  + justify-content: center; -> Flexitems werden in der Mitte des Flexcontainers angeordnet
  + justify-content: flex-end; ->Flexitems werden am Ende des Flexcontainers angeordnet
  + justify-content: space-around; Leerraum wird zwischen Items verteilt, am Anfang und am Ende ist halb so viel Leerraum wie zwischen Items
  + justify-content: space-evenly; ->Leeraum wird zwischen Items verteilt, Anfang und Ende und überall Leerraum gleich groß

Beispiel: /kapitel\_15/

* + flexbox-justify-content.html

### Element zentrieren

align-items und justify-content -> vertikal und horizontal zentrieren

body Mindesthöhe 100vh

Beispiel: /kapitel\_15/

* + flexbox-zentrieren.html

### Navigation mit Flexbox realisieren – Horizontale Navigation definieren

Beispiel: /kapitel\_15/

* + flexbox-navi.html

.navi ul -> maximale Breite und mit margin: auto zentriert, Innenabstand wird entfernt und Element wird zum Flexcontainer. Leerraum zwischen Elementen wird festgelegt

### Zweigeteilte Navigation – Elemente mit margin anordnen

Beispiel: /kapitel\_15/

* + flexbox-navi\_2.html

justify-content-Angabe fällt weg

letzter Menüpunkt erhält Klasse und margin-left: auto;

margin bei Flexbox: kann verwendet werden, um einzelne Elemente auf der Hauptachse auszurichten

### Geänderte Reihenfolge dank order

Anweisung wird bei Flexitems benötigt, um Reihenfolge zu verändern

Standartwert: order: 0;

order: -1; -> Element an den Anfang schieben

order: 1; -> ans Ende schieben

wenn zwei Elemente gleichen Wert haben, zählt die im Quellcode festgelegte Reihenfolge

Beispiel: /kapitel\_15/

* + flexbox-navi\_reihenfolge.html

### Flexible Ausmaße

wenn Navigationspunkte den verfügbaren Platz unter sich aufteilen sollen -> Anweisung: flex -> wird bei Flexitems angegeben

Beispiel: /kapitel\_15/

* + flexbox-navi\_flex.html

Abkürzung für:

* + flex-basis: Der Basiswert von Elementen, also die optimalen Ausmaße
  + flex-shrink: inwieweit Elemente schrumpfen dürfen
  + flex-grow: inwieweit Elemente größer werden dürfen

Ausgeschrieben: flex: 1;

flex-grow: 1;  
flex-shrink: 1;  
flex-basis: 0%

* + Durch flex-basis: 0%; -> alle Elemente gleich breit

### Mehrzeilig/mehrspaltig mit flex-wrap

Wenn nicht genügend Platz zur Verfügung steht -> Elemente auf mehrere Zeilen aufteilen

flex-wrap: wrap;

Beispiel: /kapitel\_15/

* + flexbox-navi\_flex\_wrap.html

### Autoprefixer: Flexbox für so viele Browser wie möglich

Flexbox-Code für ältere Browser -> <http://autoprefixer.github.io>

Beispiel: /kapitel\_15/code\_nach\_autoprefixer

# Rasterlayouts leicht gemacht mit Gridlayout

Flexbox ist im Wesentlichen eindimensional  
Skizze: <https://www.w3.org/TR/css-grid-1>

Gridlayout ist ein CSS-Layoutmodul (wie Flexbox)

Funktioniert seit März 2017 in allen wichtigen Browsern

Für ältere Browser-> Autoprefixe

Beispiel: /kapitel\_15/code\_nach\_autoprefixer

### Erstes Raster

Mit Gridlayout = richtige Rasterlayouts

Layout beim umfassenden Element aktivieren mit display: grid;

Horizontale: Definition des Rasters über grid-template-columns

* + Breite der Spalten, Anzahl an Werten gibt die Anzahl an Spalten vor
  + grid-template-columns: 1fr 3fr; -> zwei Spalten, die erste ist 1fr die andere 3fr groß
  + fr ist eine Einheit nur für Gridlayout = ein Teil des verfügbaren Platzes

Vertikale: Definition der Zeilen über grid-template-rows

* + grid-template-rows: auto auto auto auto; definiert vier Zeilen, die so hoch werden, wie der Inhalt es erfordert
  + da diese Zeielen keine besonderen Höhenangaben haben und Zeilen automatisch erzeugt werden, wird dieser Code nicht angegeben.

Viele gleiche Angaben können mit repeat() verkürzt werden

* + grid-template-columns: 1fr 1fr 1fr;   
    -> grid-template-columns: repeat(3, 1fr);

Layout ändern:

* + Kopfbereich über zwei Spalten
    - soll an der vertikalen Linie (grid-column) 1 beginnen u bis zur Linie 3 reichen
    - soll an der horizontalen Linie (grid-row) 1 beginnen und bis zur Linie 2 reichen

grid-column: 1/ 3;  
grid-row: 1/2;

* + Beispiel: /kapitel\_15/gridlayout\_einstieg.html

### Gridlayout mit benannten Bereichen

grid-template-areas

Anweisung bei Umschließenden Grid Element

grid-template-areas:  
“kopf kopf“  
“navi inhalt“  
“seite inhalt“  
“fuss fuss“;

* + In Anführungszeichen immer eine Zeile des Rasters, Wörter in Anführungszeichen steht für Rasterzelle
  + “kopf kopf“ -> Element erstreckt sich über zwei Rasterzellen

Elemente den Rasterbereichen zuweisen mit grid-area bei Gridzellen

.kopf { grid-area: kopf; }  
.navi { grid-area: navi; }

* + Beispiel: /kapitel\_15/
    - gridlayout\_benannte-bereiche.html
    - gridlayout\_benannte-bereiche\_modifiziert.html

Anmerkungen:

* + Benennung hat sich an Klassennamen orientiert = optional, kann man machen, wie man will
  + Gridinspektor von Firefox benutzen!

### Voll flexibles Raster

Raster mit Gridlayout erstellen, bei denen der Browser selbst die Anzahl an Spalten und Zeilen ermittelt

Definition:

* + grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));
  + repeat() -> sorgt für Wiederholung dessen was in der Klammer steht
    - erster Wert ist Anzahl an Wiederholungen (Zahl)  
      im Beispiel: auto-fit -> Browser ermittelt selbst die benötigte Anzahl
    - Ausmaß der Spalten mit minmax() festgelegt ->   
      mindestens 120px groß, maximal aber so groß, wie Platz zur Verfügung steht -> Browser ermittelt wie viele Spalten Platz haben und verteilt restlichen Platz unter den einzelnen Spalten = Bereich immer vollständig ausgefüllt

Anzahl der Zeilen nicht festlegen -> Browser soll ergänzen

* + grid-auto-rows: minmax(120px, auto);

kann zu Lücken kommen -> deshalb:

* + grid-auto-flow: dense;
  + kann sein, dass manche Elemente, die im Quellcode später stehen, früher angezeigt werden

ganzer Code des umgebenen Gridelements:

display: grid;  
grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));  
grid-auto-rows: minmax(120px, auto);  
grid-auto-flow: dense;

wenn einzelne Elemente über mehrere Spalten oder Zeilen erstecken dann jene Gridelemente definieren, Browser macht den Rest

.item3 {   
 grid-column: span 3;  
 grid-row: span 2;  
}  
.item5 {   
 grid-row: span 2;  
}  
.item10 {   
 grid-column: span 3;  
}

* + Beispiel: /kapitel\_15/gridlayout\_flexibles\_raster.html
  + BSP mit Bildern: <https://labs.jensimmons.com/2016/examples/image-gallery-grid-1.html>

### Ausrichten mit Gridlayout und Abständen zwischen Rasterzellen

Beispiel: /kapitel\_15/gridlayout\_ausrichtungen\_items.html

Gridlayout kann verschachtelt werden = Griditem kann selbst wieder ein Gridcontainer sein

Zwischenräume im Raster über

* + grid-gap = beide Richtungen
  + grid-row-gap = Abstand zwischen Zeilen
  + grid-column-gap = Abstand zwischen Spalten