# the CSS grid

COMP 126: Practical Web Design & Development for Everyone

# What and why?

- The CSS grid is a system of layout properties that allows us to stack and line up content in columns and rows
- Flexbox allows rows *or* columns; grid allows rows *and* columns; we often use flexbox to create smaller, one-dimensional components within grid layouts
- Like other grid systems like Bootstrap's, it's automatically viewport-responsive: makes proportions and relationships between elements easy to port between devices...
- ...but this one is built in to CSS3, so there's nothing you need to import or add to your project to make it work. Native CSS properties = better performance!

# DEVTOOLS & GUIDES

Download Firefox Quantum, which has the best devtools for CSS grid, by far

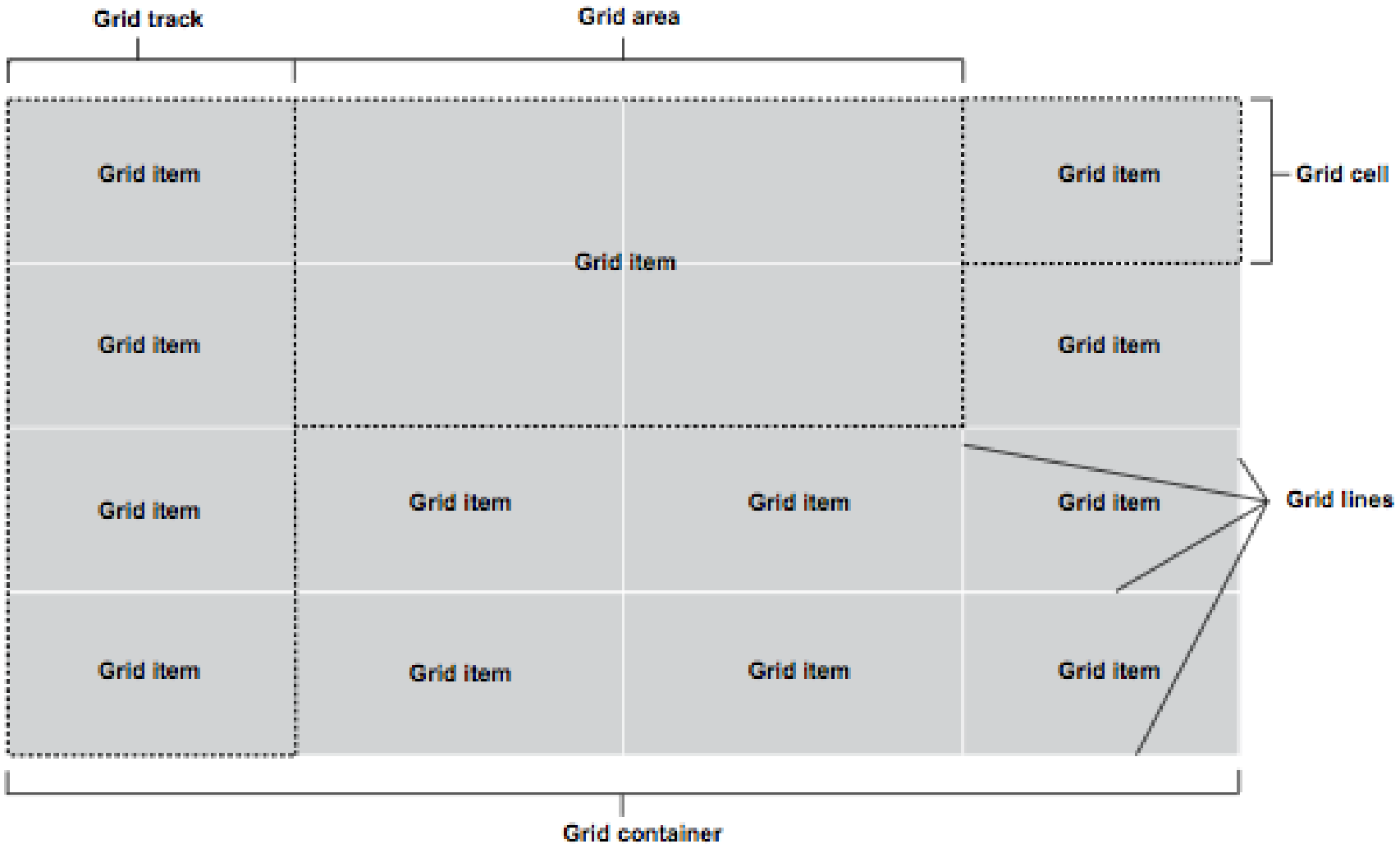
How to use the Grid Inspector in Mozilla Devtools

A nice guide to CSS grid by Mozilla

# CSS grid terms

- *grid container:* just like in flexbox: the containing unit
- *grid item*: just like in flexbox: anything inside the container
- *grid track*: column *or* row within the grid container
- *grid lines*: the lines bounding each side of a grid track: starts at far left-hand border and ends at far right-hand border
- *grid cell*: the space made up by two adjacent row and column grid lines
- *grid area*: rectangular area comprising one or more grid cells
- *grid-gap:* the gutter/space between rows/columns
- *row axis:* the x axis (always, *not* like in Flexbox)
- *column axis:* the y axis (ditto)

# The components of a grid layout



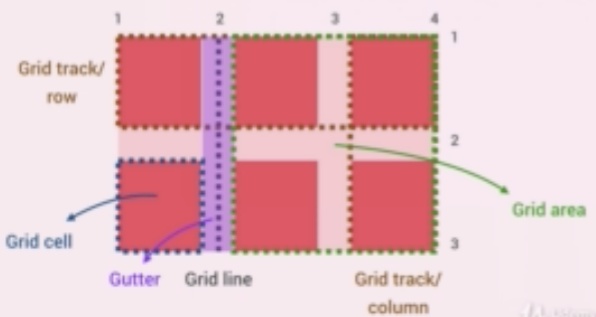
# CSS GRID PROPERTIES OVERVIEW

## CONTAINER

1. grid-template-rows
   grid-template-columns
   grid-template-areas
   → grid-template
2. grid-row-gap
   grid-column-gap
   → grid-gap
3. justify-items
   align-items
   justify-content
   align-content
4. grid-auto-rows
   grid-auto-columns
   grid-auto-flow

## ITEM

1. grid-row-start
   grid-row-end
   → grid-row
   grid-column-start
   grid-column-end
   → grid-column
   → grid-area
2. justify-self
   align-self
3. order




(image credit: Jonas Schmedtmann)

# declaring a grid container with columns and rows (in any measurement unit, or fractions [fr])

```
.grid-column-layout {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 100vh;
}
```

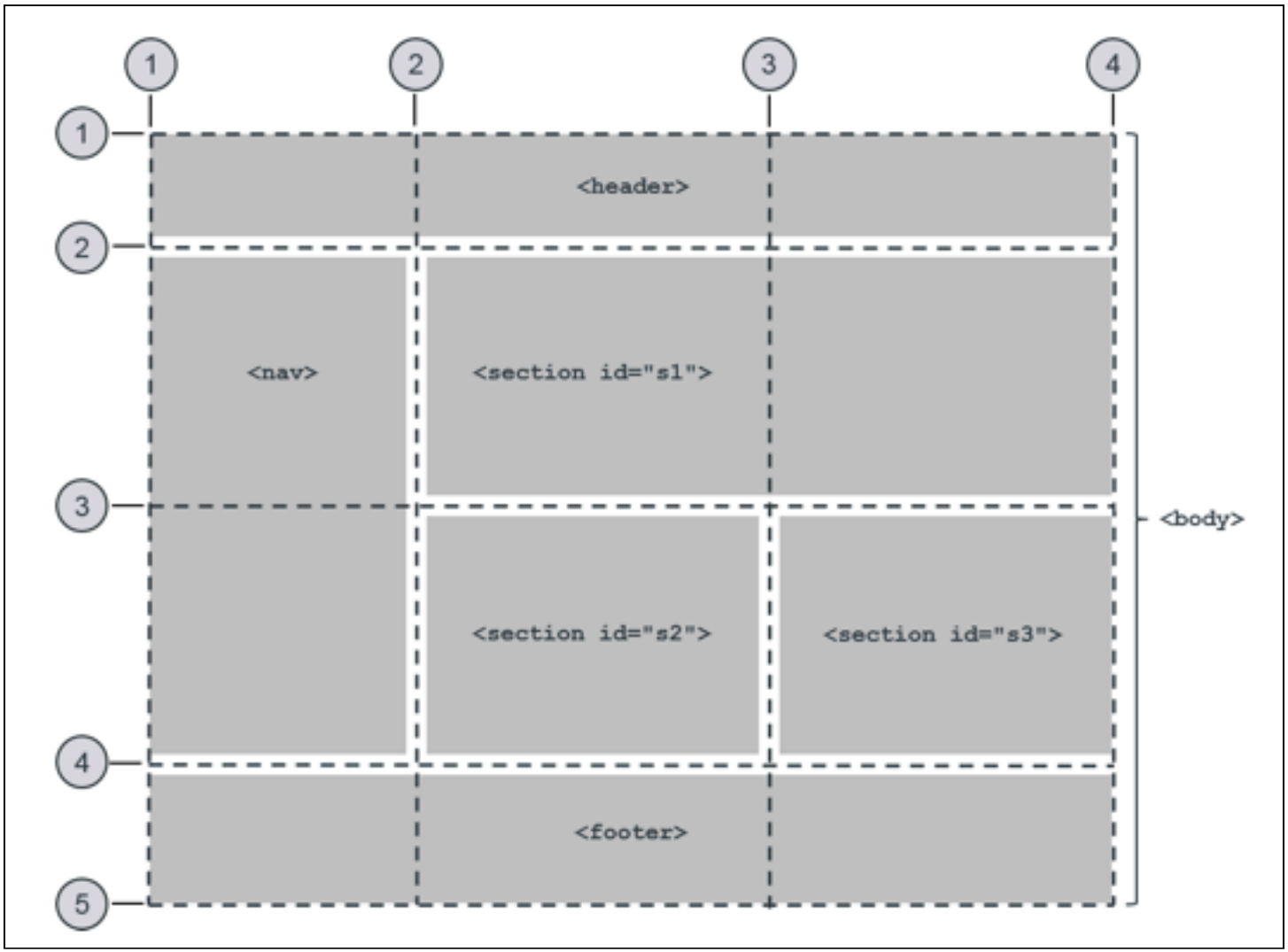
//codepen.io/tkjn/embed/GYXoGP/?height=265&theme-id=0&default-tab=css,result

add multiple rows to create content areas within your container

```
.grid-layout {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 50vh 50vh;
}
```

//codepen.io/tkjn/embed/LgJGbj/?height=265&theme-id=0&default-tab=css,result

# grid line numbers



# grid layout method 1: grid line numbers

```
.grid-column-layout {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 100vh;
}

.one  {
  grid-column: 1 / 2;
  grid-row: 1;
}
```

use grid line numbers to declare the start and end of each column or row

//codepen.io/tkjn/embed/YJOwvj/?height=265&theme-id=0&default-tab=css,result

# grid layout method 2: name your grid lines

//codepen.io/tkjn/embed/f32932c52ef242baf279a6c96605eb59/?height=265&theme-id=0&default-tab=css,result

```
<div class="aside"></div>
<div class="main"></div>
<div class="footer"></div>

.main {
  grid-area: content;
}
.footer {
  grid-area: footer;
}
.aside {
  grid-area: sidebar;
}
```

# grid layout method 3: named grid template areas

```
body {
      ...
      grid-template-areas: "sidebar  content"
                           "footer   footer";
  }

In this case, sidebar and content share the upper row/track,
while footer spans the bottom row/track
```

# grid layout method 3: named grid template areas

//codepen.io/tkjn/embed/XERLXb/?height=349&theme-id=0&default-tab=css,result&embed-version=2

# more grid areas

https://codepen.io/tkjn/embed/MWoMXBz?default-tab=html%2Cresult

# use media queries to change up your layout by redefining the grid at different viewport widths

```
@media screen and (max-width: 1024px)
  .grid-layout {
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-rows: 50vh 50vh;
  }
}
```

# mobile-first media queries with named template areas

//codepen.io/tkjn/embed/rdmELV/?height=265&theme-id=0&default-tab=css,result

# span()

You can tell an element to start at a particular grid line and span a specified number of columns/rows

https://codepen.io/tkjn/embed/ExxWMyM?height=265&theme-id=0&default-tab=css,result

# repeat()

- use the repeat() function on rows and/or columns to create repeating grid tracks
- generally used in situations with many items of equal or similar sizes
- repeat(numberOfTimes, trackDefinition)

# minmax()

use the minmax() function to set parameters for grid tracks

//codepen.io/tkjn/embed/ZPMQPV/?height=265&theme-id=0&default-tab=css,result

# auto-fit & auto-fill

- use auto-fit or auto-fill in place of a number in a repeat() to ask the browser to fill as many tracks as possible in the grid container
- create flexible columns as follows: grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
- if you only have a couple of items and you want them to stretch out to fill the track until enough items turn up to fill it, use auto-fit instead of auto-fill

# auto-fit vs. auto-fill

//codepen.io/tkjn/embed/JzbXgQ/?height=265&theme-id=0&default-tab=css,result

# minmax(), auto-fill, span

//codepen.io/tkjn/embed/EdzJob/?height=265&theme-id=0&default-tab=css,result

# grid-auto-flow

https://codepen.io/tkjn/embed/mdwZLrv?default-tab=html%2Cresult

# the implicit grid

If you define an explicit grid and don't explicitly place content inside it, the grid will auto-create *implicit* grid lines/tracks and place the content accordingly. You can size implicit tracks with grid-auto-rows & grid-auto-columns.

## Content is auto-placed in the following order:

1. Anything to which you've assigned placement properties (such as grid lines or a template area)
2. Anything that has spans applied to it
3. Each remaining item is placed in a grid cell in the order in which it appears in the HTML--*unless* you've set grid-auto-flow: dense; in which case it attempts to backfill any gaps in the grid with any item that fits