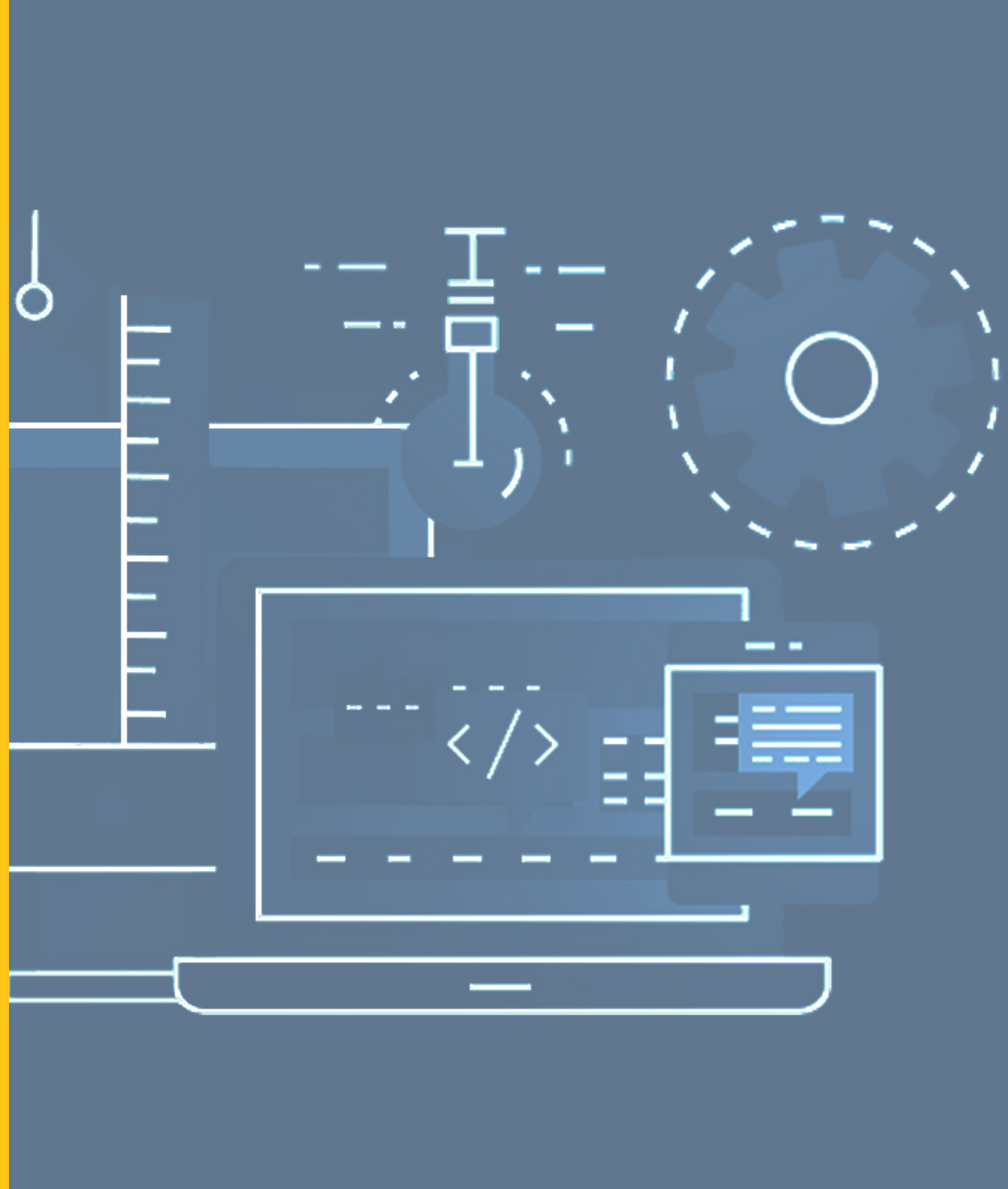
SHELLY GRAHAM, 08/10/2020

WEB DEV 4 SUMMER 2020

Week 6: CSS Grid



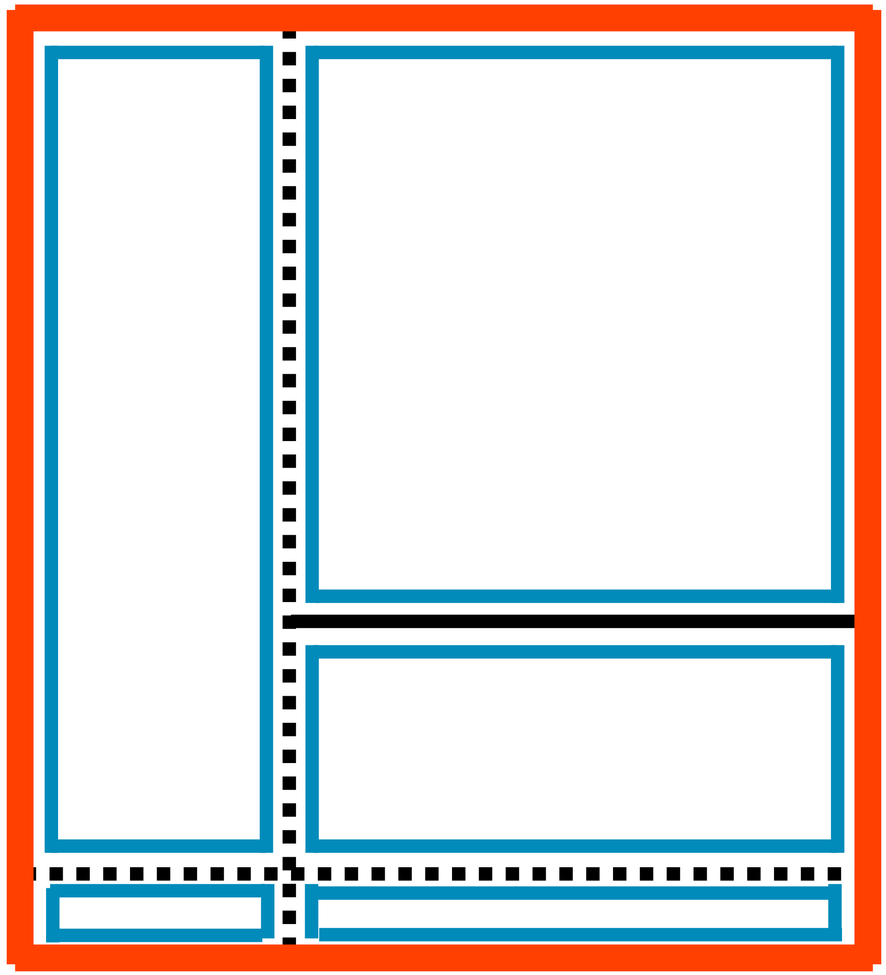
WEEK 6: CSS GRID

WHAT IS A GRID?

- A network of lines that cross each other to form a series of squares or rectangles
- No, but really?!


1.
.site-header
.site-main
.widget-area
.site-footer
MOBILE
2.
.site-header
.site-main
.widget-area
.site-footer
TABLET
3.
.site-header
.site-main
.widget-area
.site-footer
DESKTOP

IT ALL STARTS WITH...



Grid Container

Grid Items

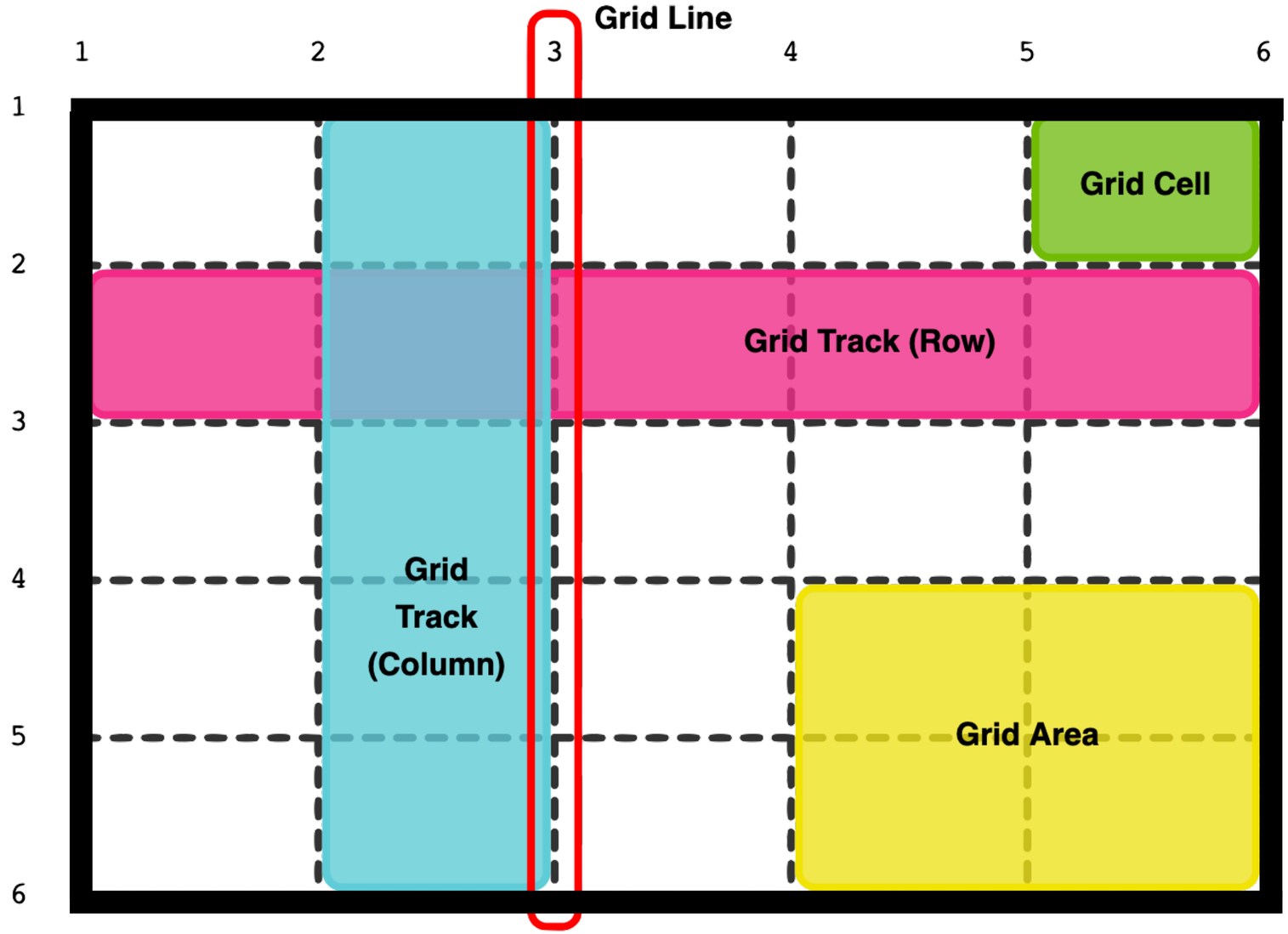
STRUCTURE OF CSS GRID

- Consists of rows and columns
- Rows and columns are defined by lines
- Each block is a grid cell, like in Excel
- You can merge cells, add cells, add rows, columns - also just like in Excel!
- Grid items can also be grid containers

```
<div class="container">
    <div class="container__item item1"></div>
    <div class="container__item item2"></div>
    <div class="container__item item3"></div>
    <div class="container__item item4"></div>
    <div class="container__item item5"></div>
</div>
```

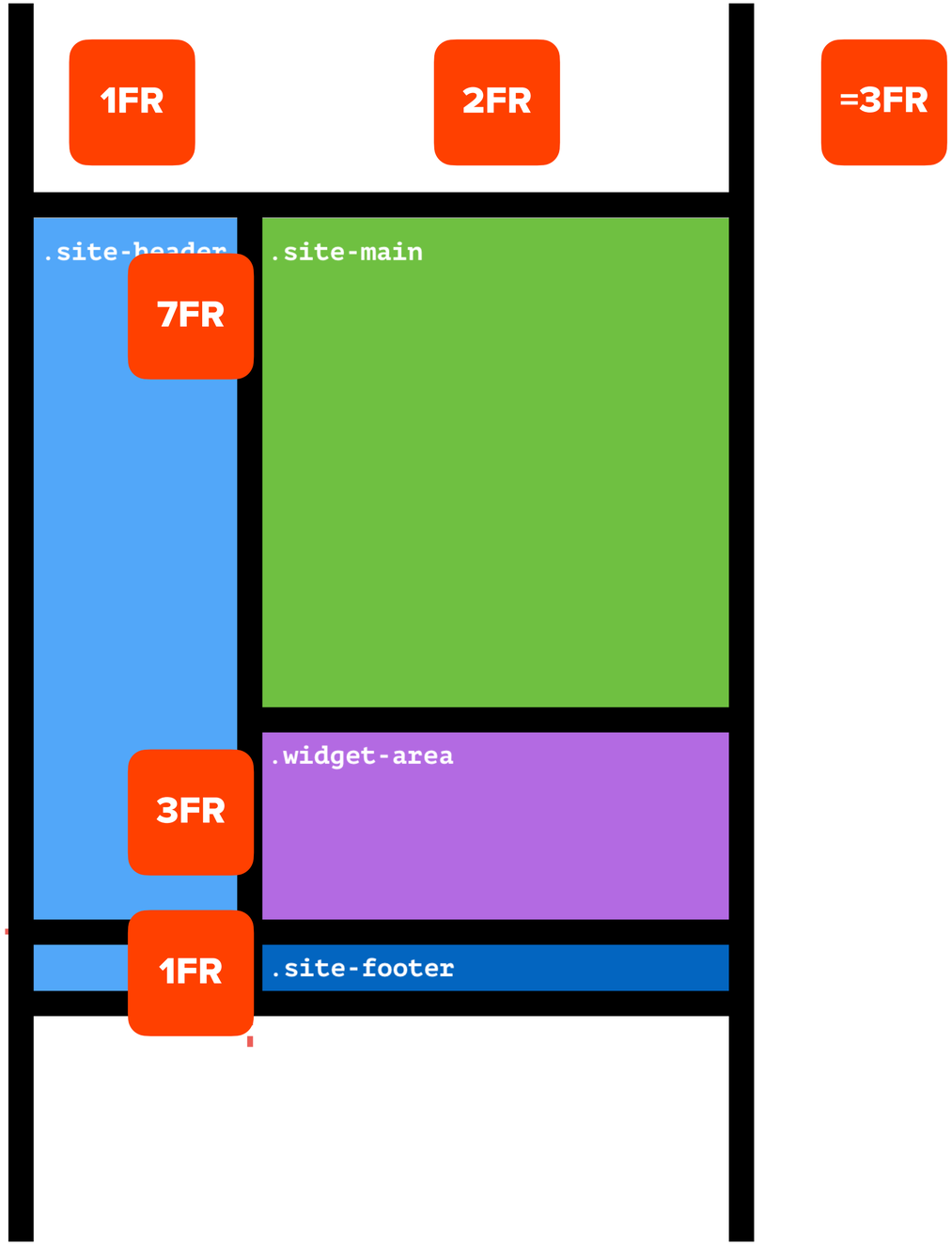
STRUCTURE OF CSS GRID

- Grid Cell: One block of content
- Grid Line: Divides grid into individual grid cells
  - Explicit grid line: manually set structure of grid
  - Implicit grid line: lines to accommodate content that has not been explicitly defined
- Track: One entire row or column
- Area: Square or rectangle of multiple cells



SPECIAL UNIT: FR

- fr = fraction aka free space
- Uses the remaining space of the grid
- Often best unit to use, even over percentages



SPECIAL USES:

- repeat(x,y): takes two values: repeat(amount of rows/columns, unit)
- span: instead of specifying end row/column, set how many cells you want the item to span!
  only works with positive values
  span 2
- min-content
- max-content
- grid-auto-fill: grid does the math for you and essentially makes your grid responsive, can replace media-queries if used with minmax(150, 1fr)!
- grid-auto-fit: always ends grid after last item
- grid-row/column: 1/-1 —> easy use to emulate width: 100% on grid!

THE PARENT - TEMPLATE

STEP 1: DISPLAY: GRID;

- Defines element as grid container
- Values:
  - grid: generates a block-level grid
  - inline-grid: generates an inline-level grid

```css
.container {
  display: grid | inline-grid;
}
```

THE PARENT - TEMPLATE

STEP 2: DEFINE ROWS AND COLUMNS IN PARENT CONTAINER

- Define # and width of rows:
  - grid-template-rows: [value]
- Define # and width of columns:
  - grid-template-columns: [value]

- Value can be:
  - Any unit (px, %, vw, rems...)
  - Any name you choose

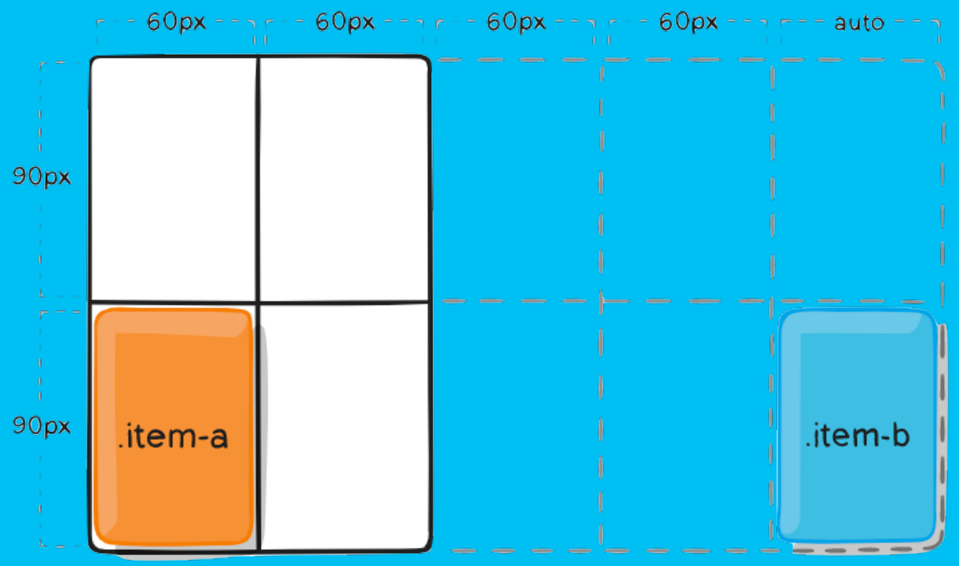
```css
.container {
  grid-template-columns: 40px 50px auto 50px 40px;
  grid-template-rows: 25% 100px auto;
}
```

THE PARENT - TEMPLATE

STEP 2.1: GRID-AUTO-COLUMNS GRID-AUTO-ROWS

- Specifies the size of any auto-generated grid tracks = *implicit tracks*
- Implicit tracks get created when there are more grid items than cells in the grid or when a grid item is placed outside of the explicit grid

```css
.container {
  grid-auto-columns: 60px;
}
```



THE CHILDREN

STEP 3: DEFINE ROWS AND COLUMNS FOR CHILDREN

- Determines a grid item’s location within the grid

- grid-row-start
- grid-row-end
- grid-column-start
- grid-column-end

- If no "grid-row/column end” is declared, the item will span 1 track by default

```css
.item-a {
  grid-column-start: 2;
  grid-column-end: five;
  grid-row-start: row1-start;
  grid-row-end: 3;
}
```

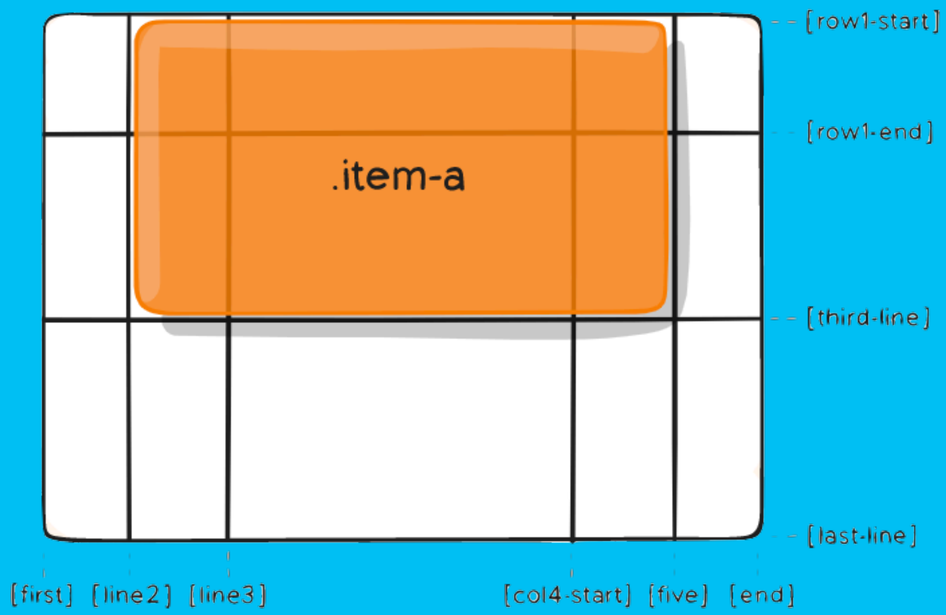
THE CHILDREN

STEP 3: DEFINE ROWS AND COLUMNS FOR CHILDREN

- Values:
  - Any unit (px, %, vw, rems...)
  - span [any number]
  - span [any name you choose]

- If no "grid-row/column end" is declared, the item will span 1 track by default

```css
.item-a {
  grid-column-start: 2;
  grid-column-end: five;
  grid-row-start: row1-start;
  grid-row-end: 3;
}
```



THE CHILDREN

STEP 3: DEFINE ROWS AND COLUMNS FOR CHILDREN

- SHORTCUT:
  - grid-row
  - grid-column

```css
.item-c {
  grid-column: 3 / span 2;
  grid-row: third-line / 4;
}
```

THE CHILDREN

STEP 4: GRID-AREA

- **SHORTCUT** for grid-row and grid-column
- Order: row-start/column-start/row-end/column-end
- Values:
  - Any unit (px, %, vw, rems...)
  - Any name you choose, without quotation marks

```css
.item-d {
  grid-area: 1 / col4-start / last-line / 6;
}
```

THE CHILDREN

STEP 4: GRID-AREA

- **SHORTCUT:**
  grid-row-start +
  grid-column-start +
  grid-row-end +
  grid-column-end

```css
.item-d {
  grid-area: 1 / col4-start / last-line / 6;
}
```

THE PARENT - TEMPLATE

STEP 5: GRID-TEMPLATE-AREAS

- Defines a grid template by referencing the names of the grid areas
- Easiest to work in lines
- Each line uses quotation marks
- If you want three rows, write grid-area 3 times, if you want four, write it out 4 times etc...

```css
.item-a {
  grid-area: header;
}
.item-b {
  grid-area: main;
}
.item-c {
  grid-area: sidebar;
}
.item-d {
  grid-area: footer;
}

.container {
  display: grid;
  grid-template-columns: 50px 50px 50px 50px;
  grid-template-rows: auto;
  grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer";
}
```

THE PARENT - TEMPLATE

STEP 6: GRID-TEMPLATE

- SHORTHAND:
  - grid-template: grid-areas / grid-template-rows / grid-template-columns
  - For setting in one declaration:
    grid-template-rows
    grid-template-columns
    grid-template-areas

```
.container {
  display: grid;
  grid-template-columns: 50px 50px 50px 50px;
  grid-template-rows: auto;
  grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer";
}
```

grid-template:

No commas at end of lines!

“header header header header”
“main main . sidebar”
“footer footer footer footer” /
20% 1fr 15% 150px

No quotations for columns

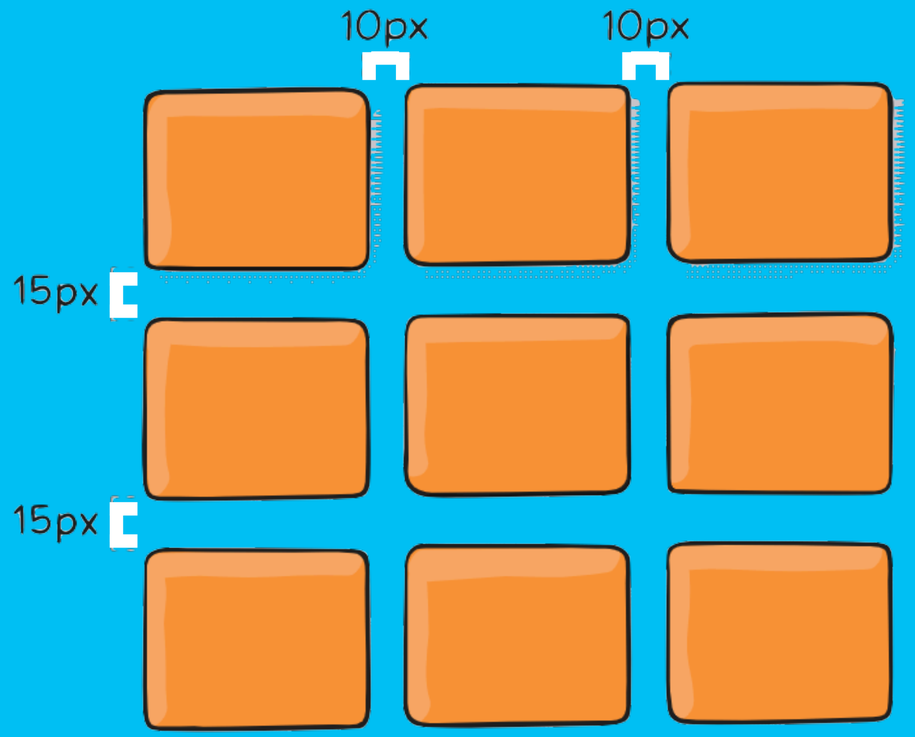
THE PARENT - TEMPLATE

STEP7: GAPS

- row-gap
- column-gap
- grid-row-gap
- grid-column-gap

- Value:
  - Any unit (px, %, vw, rems...)

```css
.container {
  grid-template-columns: 100px 50px 100px;
  grid-template-rows: 80px auto 80px;
  column-gap: 10px;
  row-gap: 15px;
}
```



THE PARENT - TEMPLATE

STEP7: GAPS

- SHORTHAND:
  - gap: grid-row-gap grid-column-gap

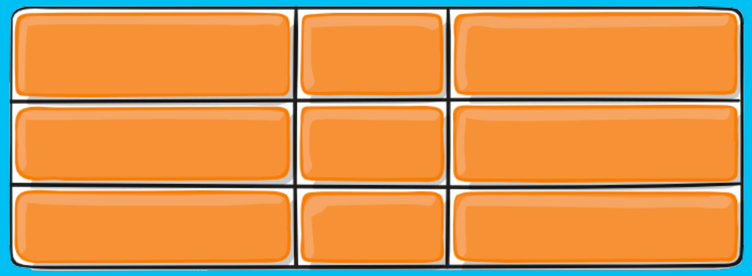
```css
.container {
  /* standard */
  gap: <grid-row-gap> <grid-column-gap>;
```

THE PARENT - TEMPLATE

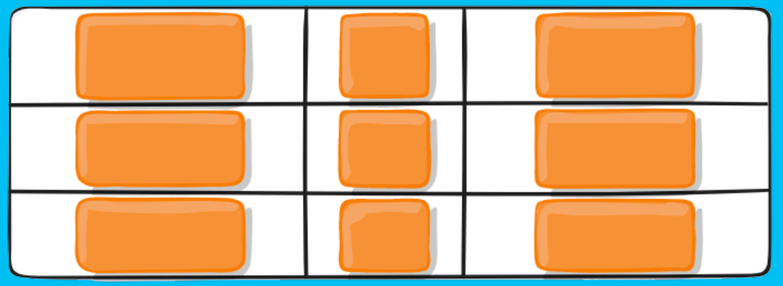
STEP 8: JUSTIFY-ITEMS

- Aligns grid items along the row axis = horizontal positioning
- Values:
  - start
  - end
  - center
  - stretch (default)

```css
.container {
  justify-items: stretch;
}
```



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

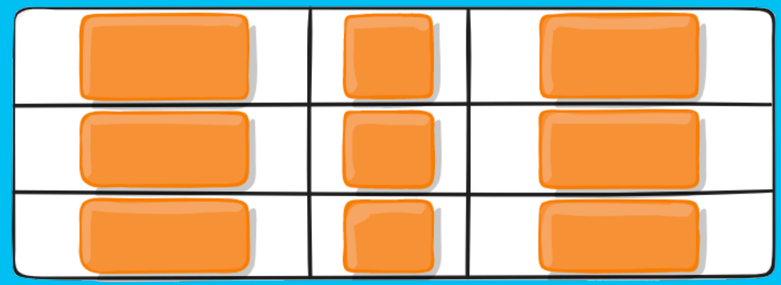


THE CHILDREN

STEP 8.1: JUSTIFY-SELF

- Aligns grid item inside the cell along the row axis = horizontal
- Values:
  - start
  - end
  - center
  - stretch (default)

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



Grid Container

```css
.item-a {
  justify-self: end;
}
```

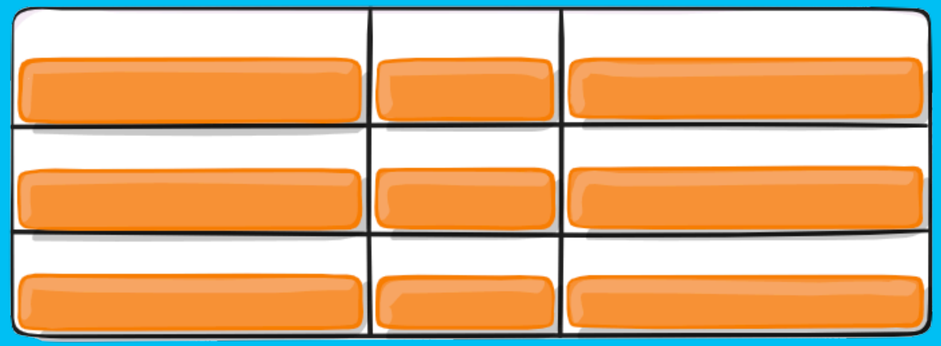
Grid Item

THE PARENT - TEMPLATE

STEP 9: ALIGN-ITEMS

- Aligns grid items along the column axis = vertical positioning
- Values:
  - start
  - end
  - center
  - stretch (default)

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

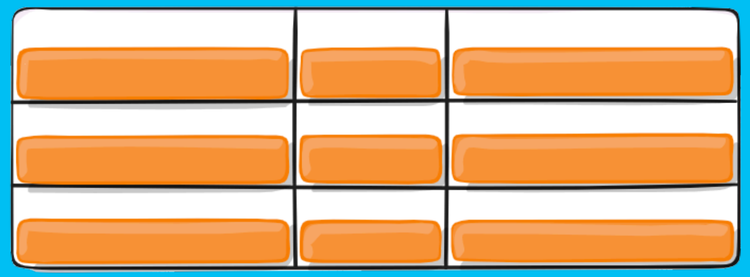


THE CHILDREN

STEP 9.1: ALIGN-SELF

- Aligns grid item inside the cell along the column axis = vertical
- Values:
  - start
  - end
  - center
  - stretch (default)

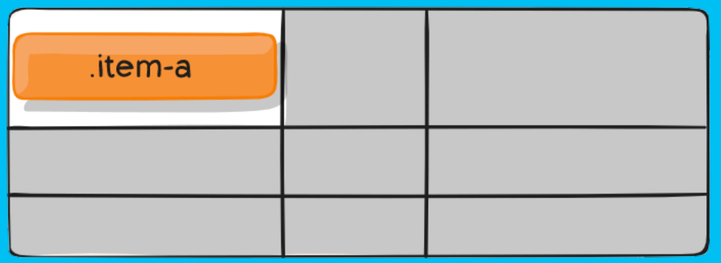
```css
.container {
  align-items: end;
}
```



Grid Container

```css
.item-a {
  align-self: center;
}
```

Grid Item

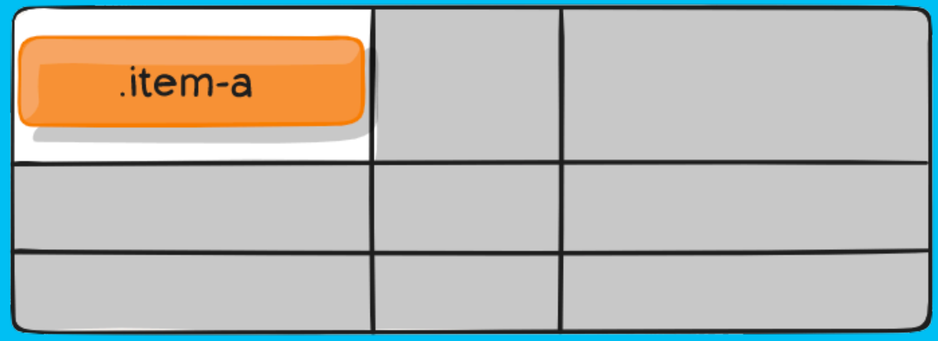


THE PARENT - TEMPLATE

STEP 10: PLACE-ITEMS

- **SHORTHAND:**
- Sets in one declaration:
  justify-items
  align-items
- Values:
  - justify-items
  - align-items
- Position grid in center:
  place-items: center center;

```css
.item-a {
  place-items: center stretch;
}
```

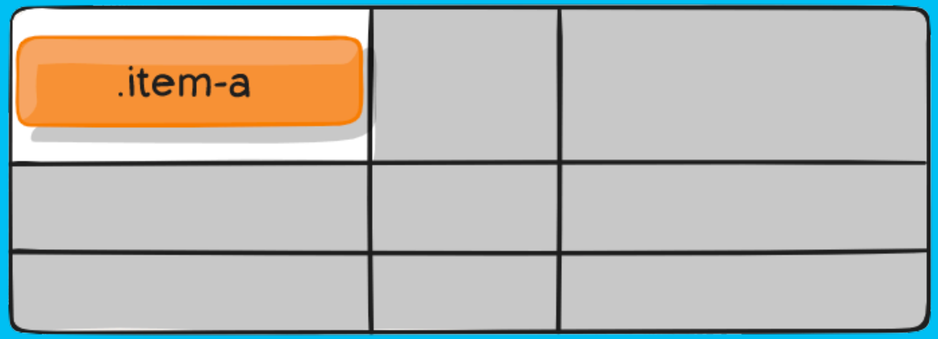


THE CHILDREN

STEP 10.1: PLACE-SELF

- **SHORTHAND:**
- Sets in one declaration:
  align-self
  justify-self
- Values:
  - align-self
  - justify-self
  - auto
- Position cell in center:
  place-self: center center;

```css
.item-a {
  place-self: center stretch;
}
```

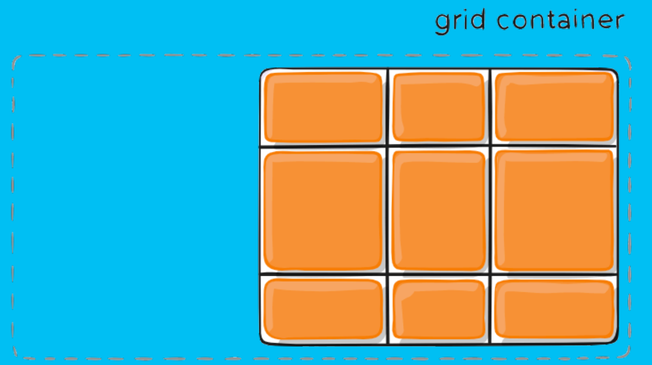


THE PARENT - TEMPLATE

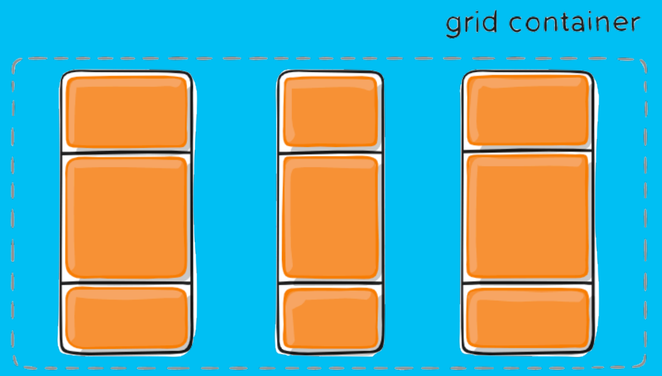
STEP 11: JUSTIFY-CONTENT

- Aligns grid within grid container along row axis = horizontal positioning
- Values:
  - start
  - end
  - center
  - stretch (default)
  - space-around
  - space-between
  - space-evenly

```css
.container {
  justify-content: end;
}
```




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

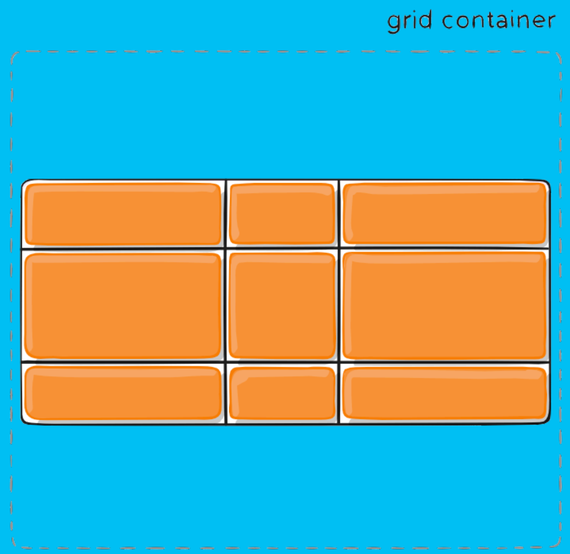


THE PARENT - TEMPLATE

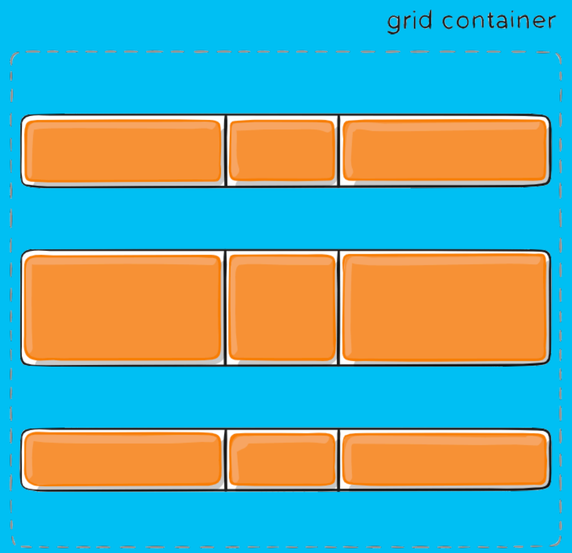
STEP 11: ALIGN-CONTENT

- Aligns grid within grid container along column axis = vertical positioning
- Values:
  - start
  - end
  - center
  - stretch (default)
  - space-around
  - space-between
  - space-evenly

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




```css
.container {
  align-content: space-evenly;
}
```



THE PARENT - TEMPLATE

STEP 12: PLACE-CONTENT

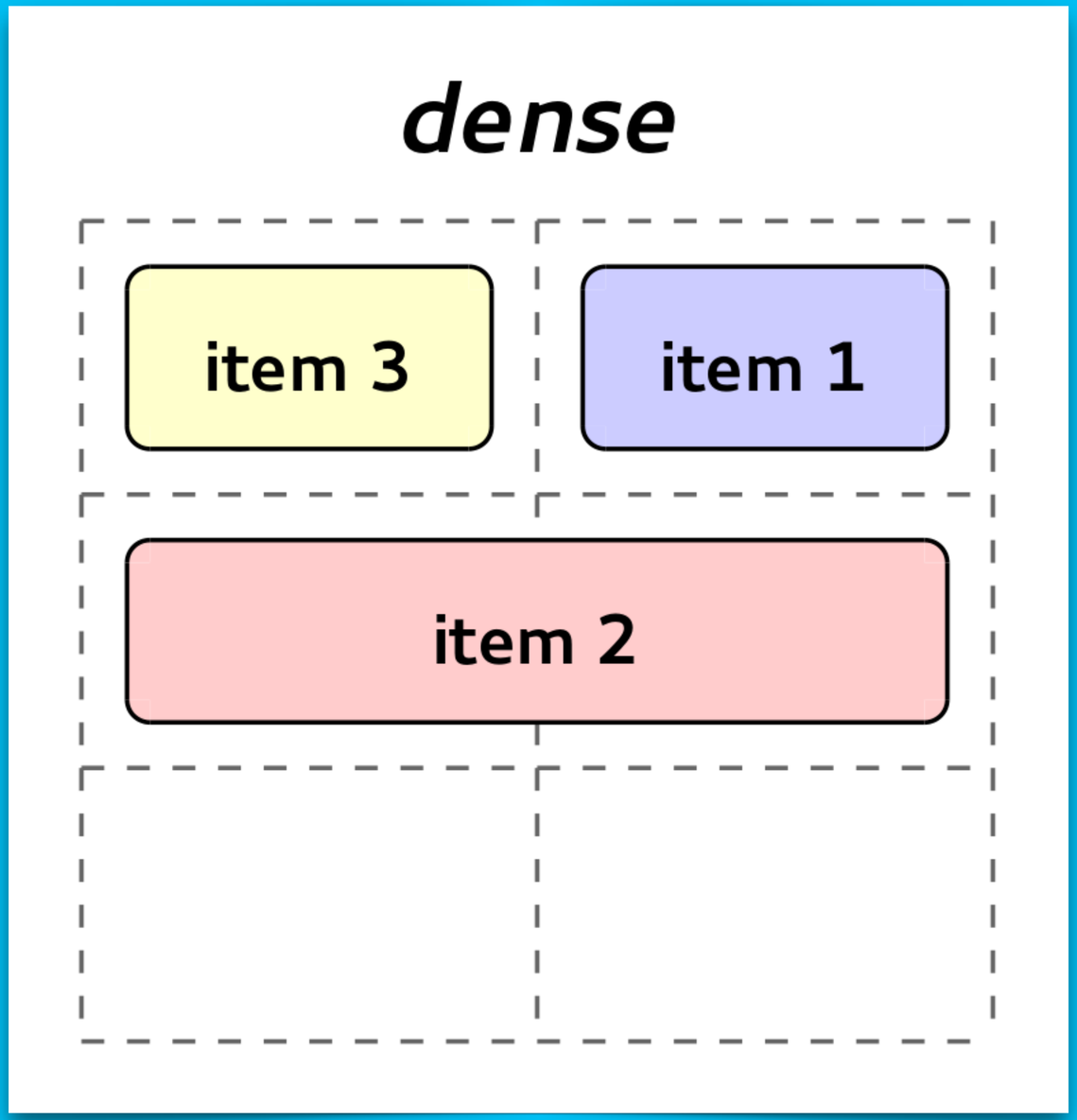
- SHORTHAND:
- Sets in one declaration:
  align-content
  justify-content
- Values:
  - align-self
  - justify-self
- Position content in center:
  place-content: center center;

```css
.item-a {
  place-content: center stretch;
}
```

THE PARENT - TEMPLATE

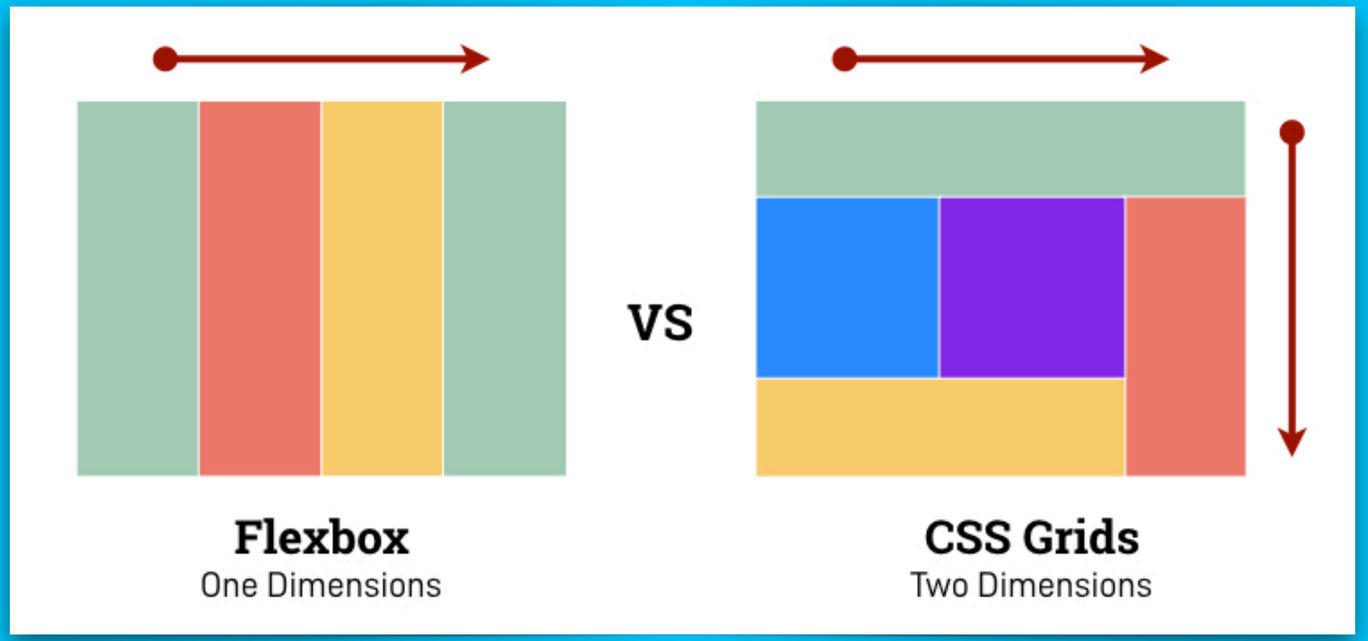
STEP 13: GRID-AUTO-FLOW

- Like flex-direction
- Values:
  - row (default)
  - column
  - dense —>
    tells the auto-placement algorithm to attempt to fill in holes earlier in the grid if smaller items come up later



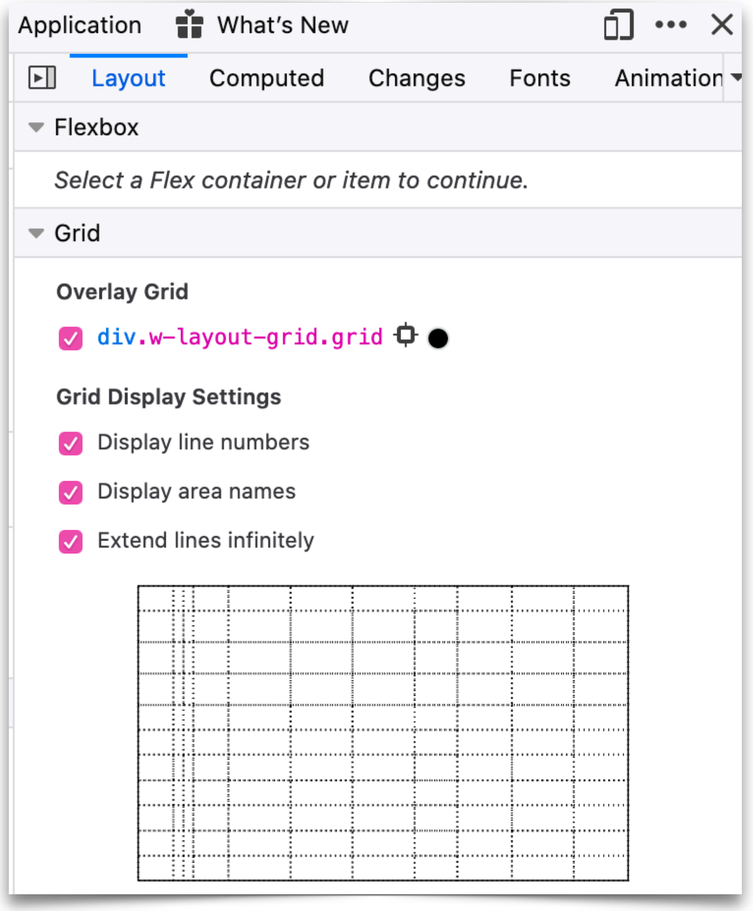
CSS GRID VS. CSS FLEXBOX

- They both have their uses, one is not better than the other
- Grid is newer, so always check browser support
- float, display: inline-block, vertical-align and column-* properties have no effect on a grid item



GOOD TO KNOW THINGS:

- When coding with CSS Grid, use Firefox Mozilla browser instead of Chrome for extra layout tab!
- Empty space: simply use a dot (.)
- End value can be lower than start value
- You can use negative values to start counting from the right site (because there is no row/column-reverse like in Flexbox
- Create mosaic-grid layouts by overlapping items and using z-index to determine hierarchy
- order: Use carefully, it will mess up your accessibility (focus order)



QUESTIONS?

FIN