16B

CSS LAYOUT WITH GRID

OVERVIEW

- Grid terminology
- Grid display type
- Creating the grid template
- Naming grid areas
- Placing grid items
- Implicit grid behavior
- Grid spacing and alignment

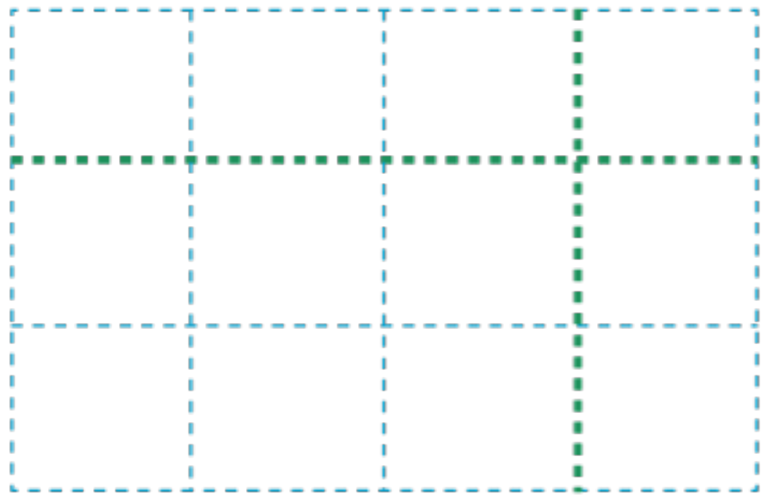
How CSS Grids Work

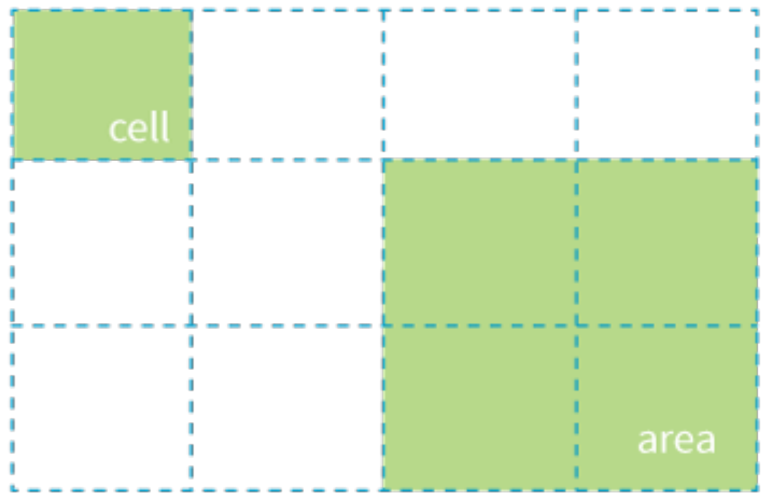
1. Set an element's **`display`** to **`grid`** to establish a **grid container**. Its children become **grid items**.

2. Set up the columns and rows for the grid (explicitly or with rules for how they are created on the fly).

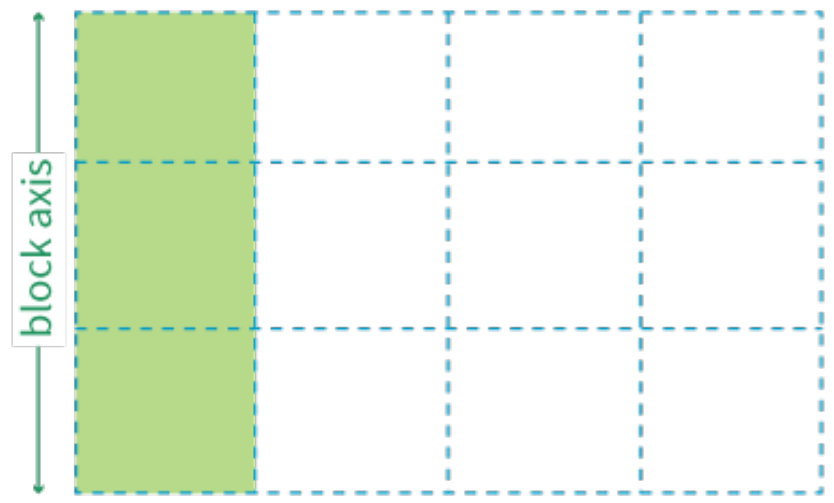
3. Assign each grid item to an area on the grid (or let them flow in automatically in sequential order).

Grid Terminology

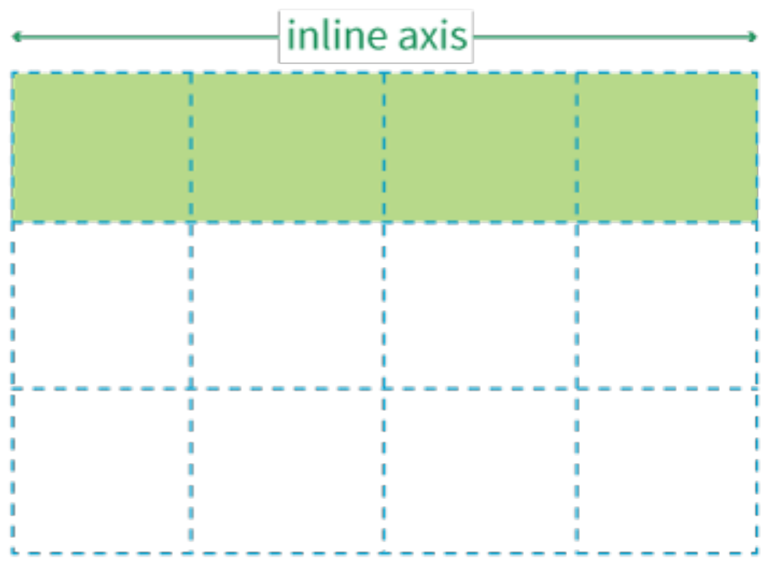

grid lines



grid cell and grid area



grid track (column)



grid track (row)

Creating a Grid Container

To make an element a **grid container**, set its **`display`** property to **grid**.

All of its children automatically become **grid items**.

The markup

```
<div id="layout">
  <div id="one">One</div>
  <div id="two">Two</div>
  <div id="three">Three</div>
  <div id="four">Four</div>
  <div id="five">Five</div>
</div>
```

The styles

```
#layout {
  display: grid;
}
```

Defining Row and Column Tracks

`grid-template-rows`
`grid-template-columns`

Values: `none`, *list of track sizes and optional line names*

- The value of **`grid-tempate-rows`** is a list of the *heights* of each row track in the grid.

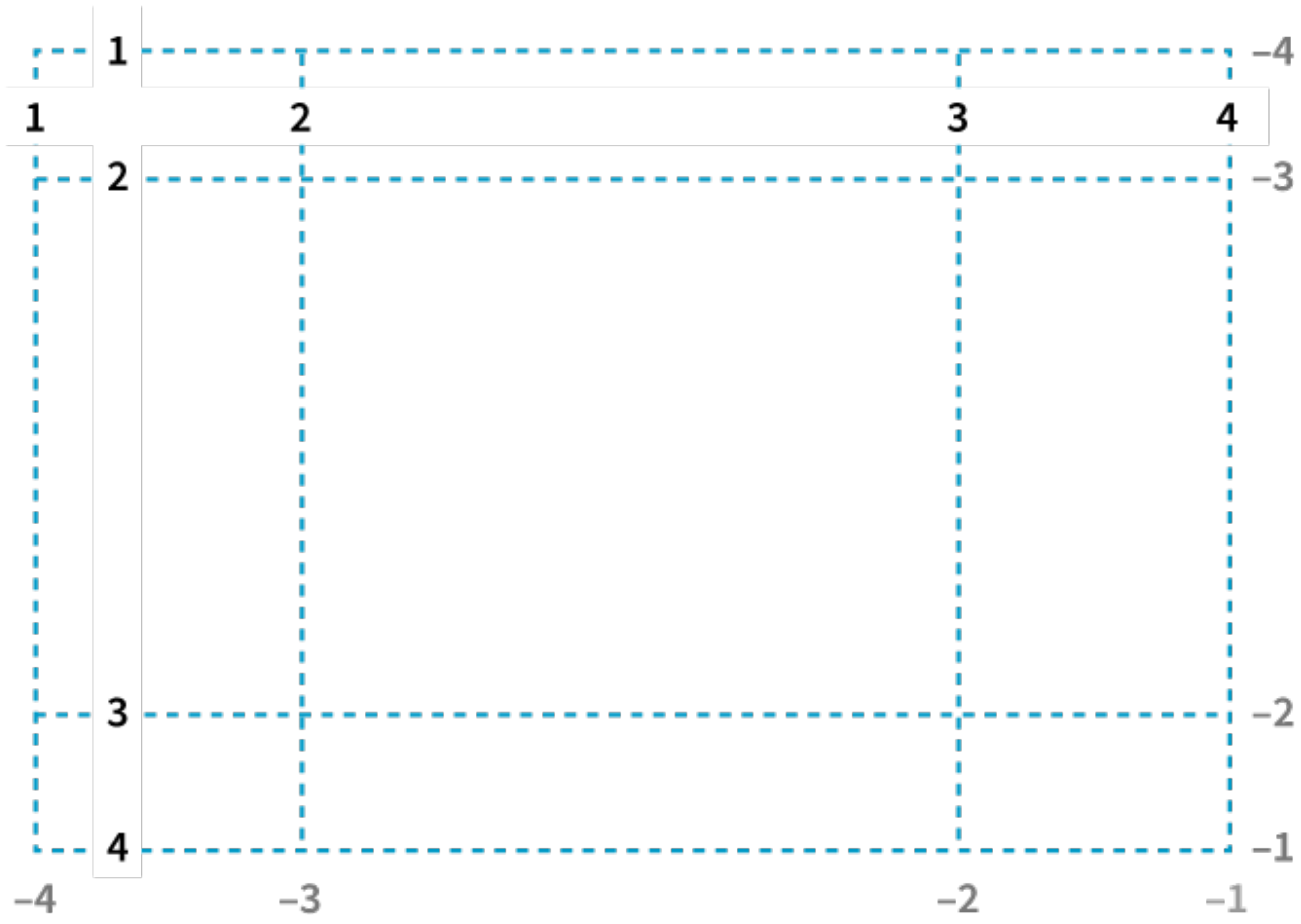
- The value of **`grid-template-columns`** is a list of the *widths* of each column track in the grid.

```
#layout {
  display: grid;
  grid-template-rows: 100px 400px 100px;
  grid-template-columns: 200px 500px 200px;
}
```

- The number of sizes provided determines the number of rows/columns in the grid. This grid in the example above has 3 rows and 3 columns.

Grid Line Numbers

Browsers assign a number to every grid line automatically, starting with **1** from the beginning of each row and column track and also starting with **–1** from the end.



Grid Line Names

You can also assign names to lines to make them more intuitive to reference later.

Grid line names are added in square brackets in the position they appear relative to the tracks.

To give a line more than one name, include all the names in brackets, separated by spaces.

```
#layout {
  display: grid;
  grid-template-rows: [header-start] 100px [header-end content-start]
400px [content-end footer-start] 100px;
  grid-template-columns: [ads] 200px [main] 500px [links] 200px;
}
```

Track Size Values

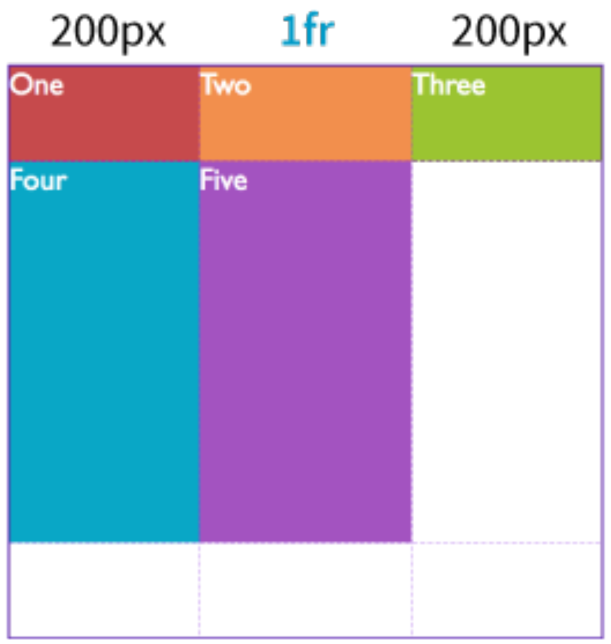
The CSS Grid spec provides a *lot* of ways to specify the width and height of a track. Some of these ways allow tracks to adapt to available space and/or to the content they contain:

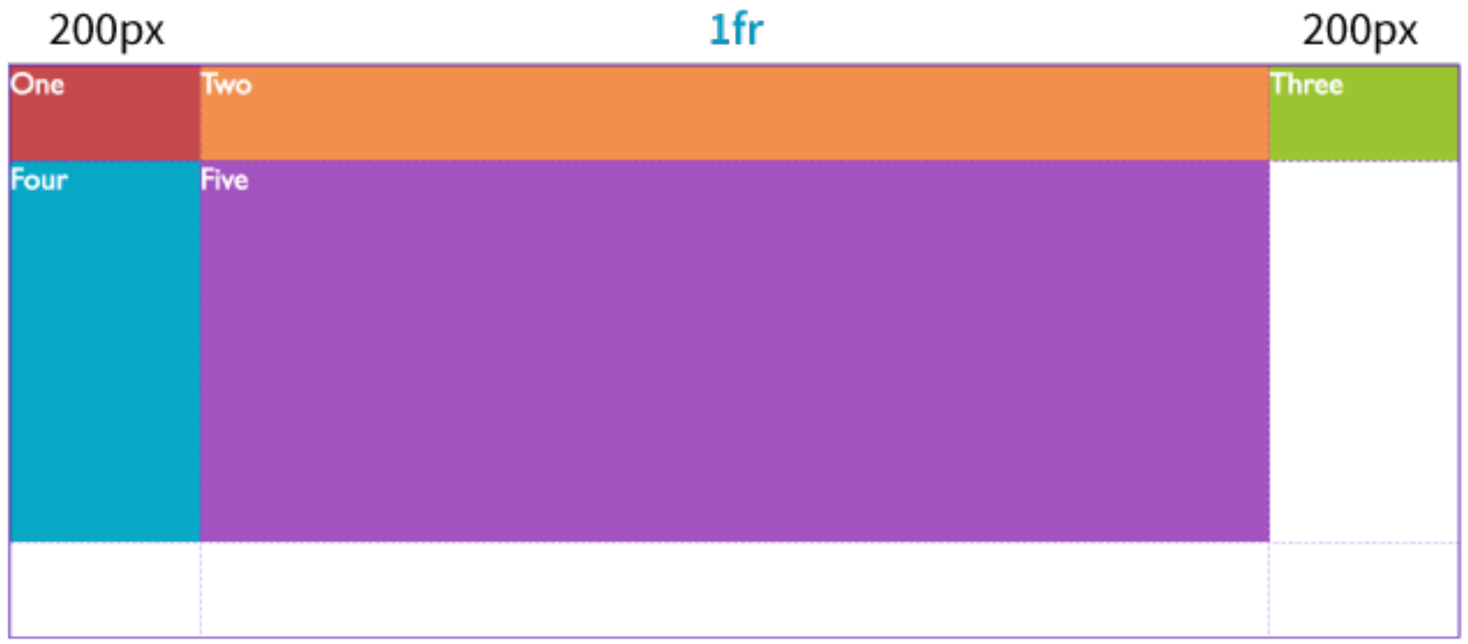
- Lengths (such as `px` or `em`)
- Percentage values (%)
- Fractional units (`fr`)
- `minmax()`
- `min-content`, `max-content`
- `auto`
- `fit-content()`

Fractional Units (fr)

The Grid-specific fractional unit (**`fr`**) expands and contracts based on available space:

```
#layout {
  display: grid;
  grid-template-rows: 100px 400px 100px;
  grid-template-columns: 200px 1fr 200px;
}
```





Size Range with minmax()

- The `minmax()` function constricts the size range for the track by setting a minimum and maximum dimension.
- It's used in place of a specific track size.
- This rule sets the middle column to at least 15em but never more than 45em:

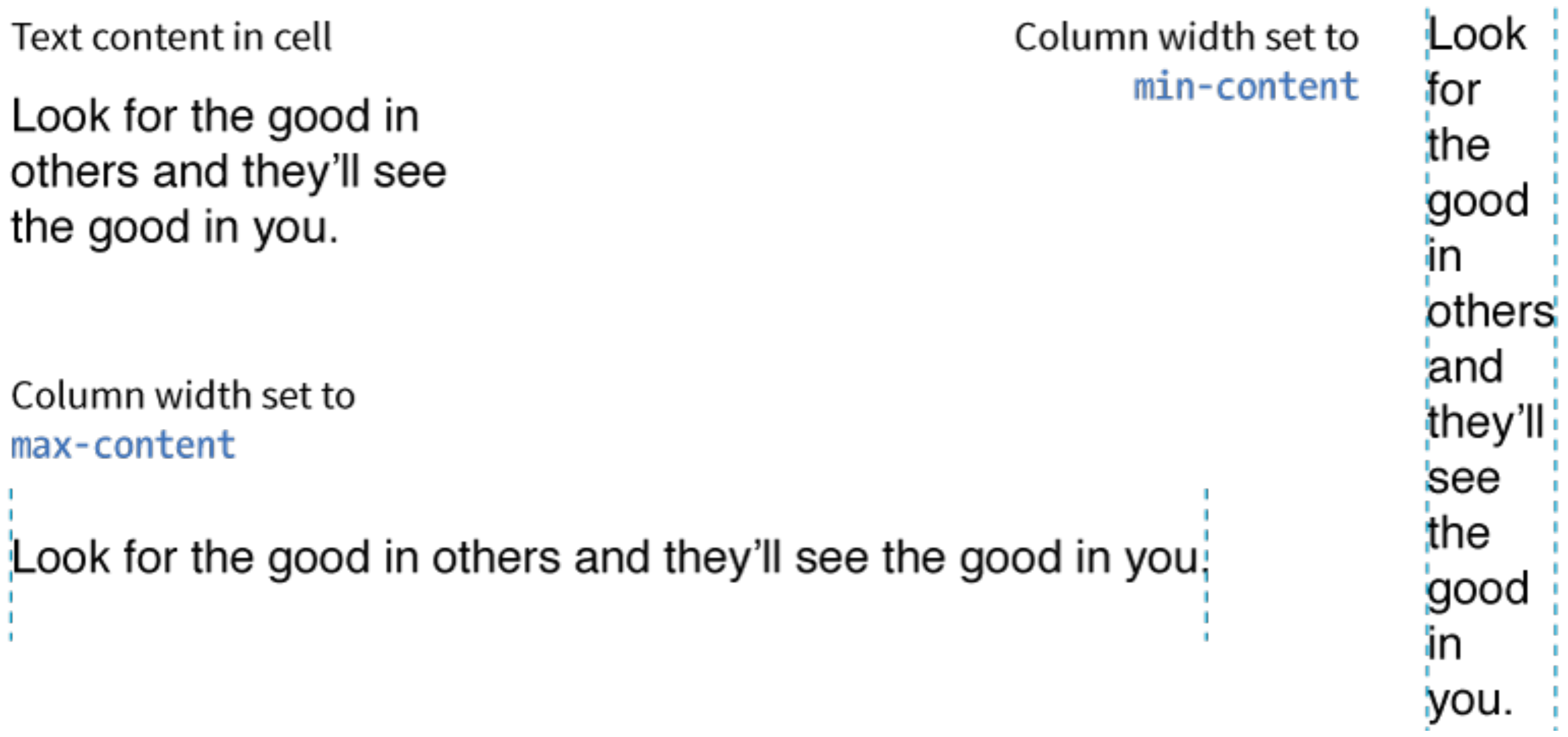
```
grid-template-columns: 200px minmax(15em, 45em) 200px;
```

min-content and max-content

`min-content` is the smallest that a track can be.

`max-content` allots the maximum amount of space needed.

`auto` lets the browser take care of it. Start with **`auto`** for content-based sizing.



Text content in cell
Look for the good in others and they'll see the good in you.
Column width set to max-content
Look for the good in others and they'll see the good in you.
Column width set to min-content
Look for the good in others and they'll see the good in you.

Repeating Track Sizes

The shortcut **`repeat()`** function lets you repeat patterns in track sizes:

`repeat(#, `*`track pattern`*`)`

The first number is the number of repetitions. The track sizes after the comma provide the pattern:

BEFORE:

```
grid-template-columns: 200px 20px 1fr 20px 1fr 20px 1fr 20px
1fr 20px 1fr 20px 1fr 200px;
```

AFTER:

```
grid-template-columns: 200px repeat(5, 20px 1fr) 200px;
```

(Here **`repeat()`** is used in a longer sequence of track sizes. It repeats the track sizes `20px 1fr` 5 times.)

Repeating Track Sizes (cont'd.)

You can let the browser figure out how many times a repeated pattern will fit with **`auto-fill`** and **`auto-fit`** values instead of a number:

```
grid-template-rows: repeat(auto-fill, 15em);
```

`auto-fill` creates as many tracks as will fit in the available space, even if there's not enough content to fill all the tracks.

`auto-fit` creates as many tracks as will fit, dropping empty tracks from the layout.

NOTE: If there's leftover space in the container, it's distributed according to the provided vertical and horizontal alignment values.

Giving Names to Grid Areas

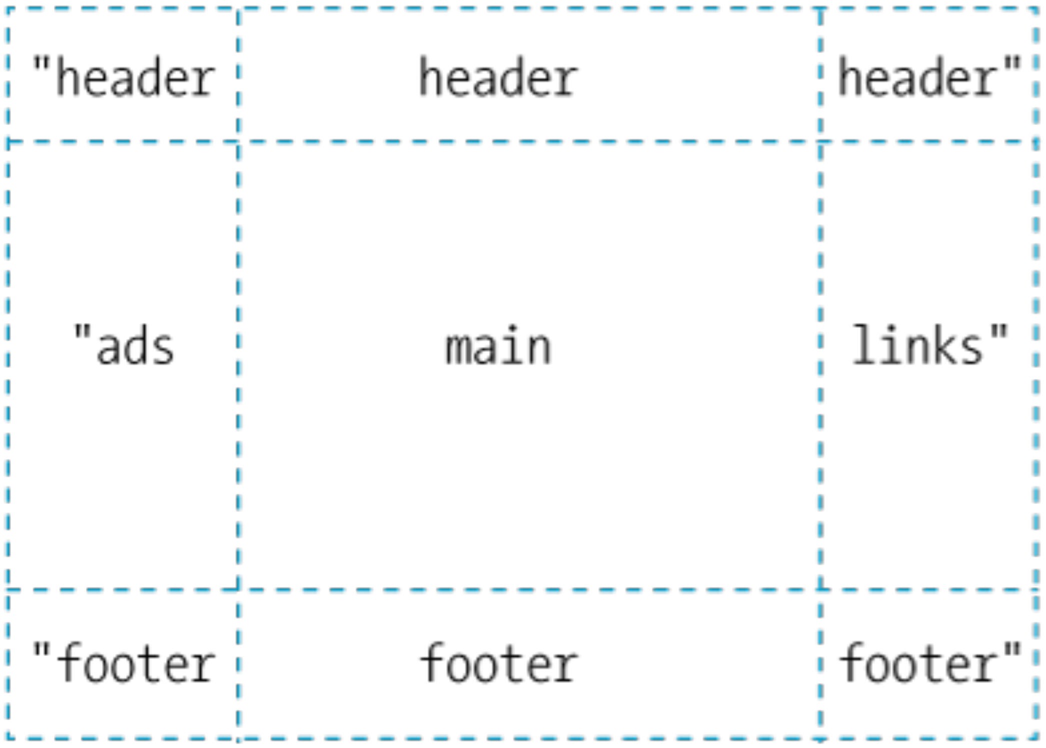
`grid-template-areas`

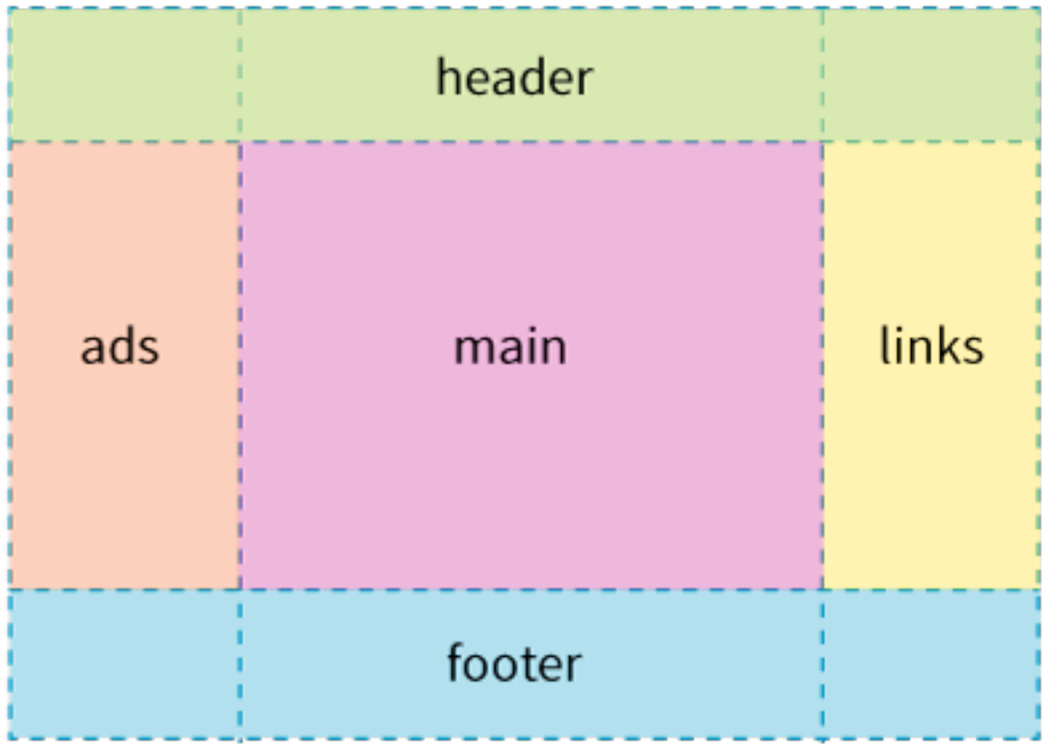
Values: `none`, *series of area names by row*

- **`grid-template-areas`** lets you assign names to areas in the grid to make it easier to place items in that area later.
- The value is a list of names for every cell in the grid, listed by row.
- When neighboring cells share a name, they form a grid area with that name.

Giving Names to Grid Areas (cont'd)

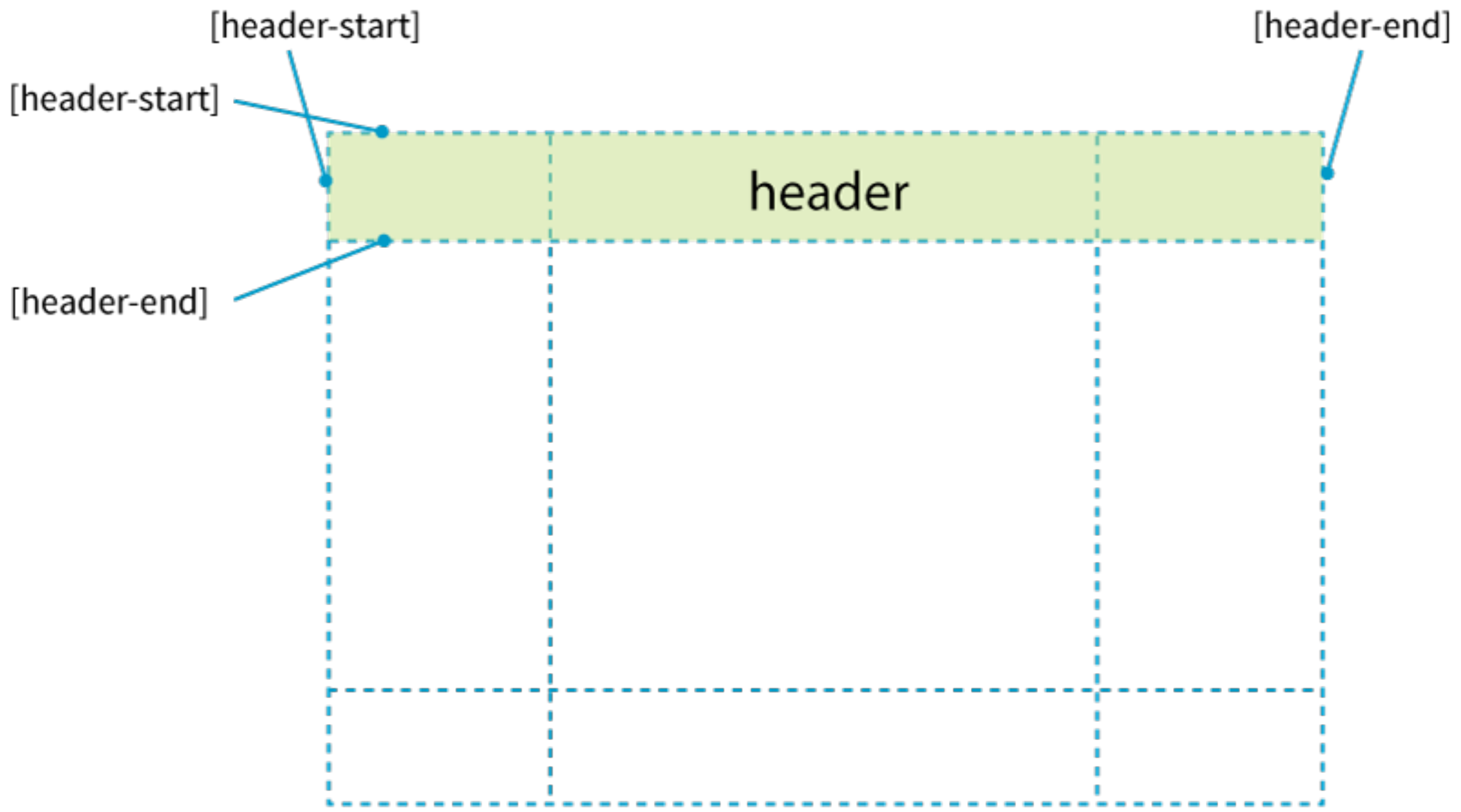
```
#layout {
  display: grid;
  grid-template-rows: [header-start] 100px [content-start] 400px
[footer-start] 100px;
  grid-template-columns: [ads] 200px [main] 1fr [links] 200px;
  grid-template-areas:
    "header  header  header"
    "ads     main    links"
    "footer  footer  footer"
}
```





Giving Names to Grid Areas (cont'd)

Assigning names to lines with -start and -end suffixes creates an area name **implicitly**.



Similarly, when you specify an area name with `grid-template-areas`, line names with -start and -end suffixes are implicitly generated.

The grid Shorthand Property

`grid`

Values: `none`, *row info/column info*

The **`grid`** shorthand sets values for `grid-template-rows`, `grid-template-columns`, and `grid-template-areas`.

NOTE: The `grid` shorthand is available, but the word on the street is that it's more difficult to use than separate template properties.

The grid Shorthand Property (cont'd)

Put the row-related values before the slash (/) and column-related values after:

```
grid: rows / columns
```

Example:

```
#layout {
  display: grid;
  grid: 100px 400px 100px / 200px 1fr 200px;
}
```

The grid Shorthand Property (cont’d)

You can include line names and area names as well, in this order:

```
[start line name] "area names" <track size> [end line name]
```

Example:

```
#layout {
  display: grid;
  grid:
    [header-start]  "header   header   header" 100px
    [content-start] "ads      main     links"  400px
    [footer-start]  "footer   footer   footer" 100px
    / [ads] 200px [main] 1fr [links] 200px;
}
```

The names and height for each row are stacked here for clarity. Note that the column track information is still after the slash (/).

Placing Items Using Grid Lines

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

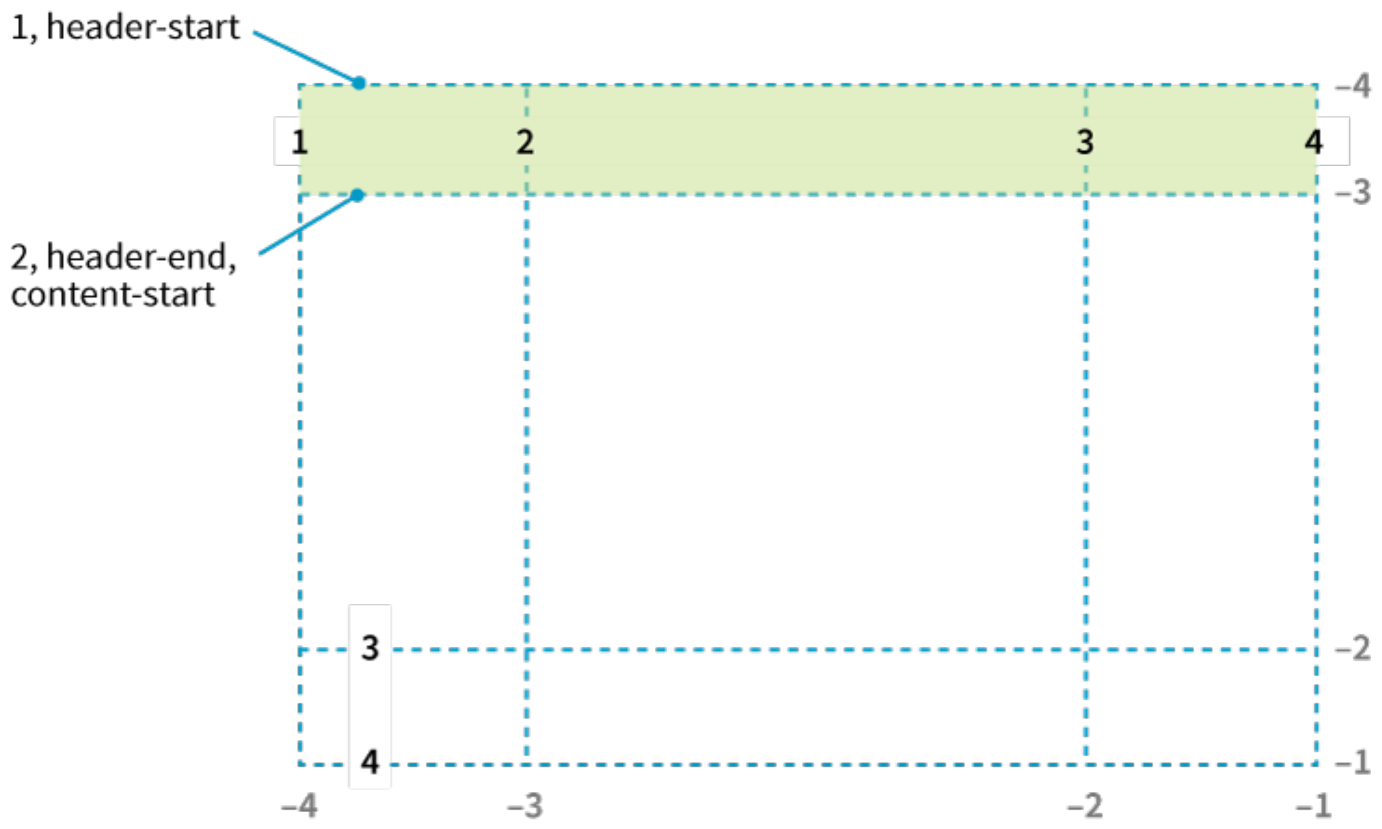
Values: `auto`, *"line name"*, `span` *number*, `span` *"line name"*, *number "line name"*

- These properties position grid items on the grid by referencing the grid lines where they begin and end.
- The property is applied to the **grid item** element.

Placing Items on the Grid (cont'd)

By line number:

```
#one {
  grid-row-start: 1;
  grid-row-end: 2;
  grid-column-start: 1;
  grid-column-end: 4;
}
```



Using a span:

```
#one {
  ...
  grid-column-start: 1;
  grid-column-end: span 3;
}
```

Starting from the last grid line and spanning backward:

```
#one {
  ...
  grid-column-start: span 3;
  grid-column-end: -1;
}
```

By line name:

```
#one {
  grid-row-start: header-start;
  grid-row-end: header-end;
  …
}
```

Placing Items on the Grid (cont'd)

`grid-row`
`grid-column`

Values: *"start line" / "end line"*

These shorthand properties combine the `*-start` and `*-end` properties into a single declaration. Values are separated by a slash (/):

```
#one {
  grid-row: 1 / 2;
  grid-column: 1 / span 3;
}
```

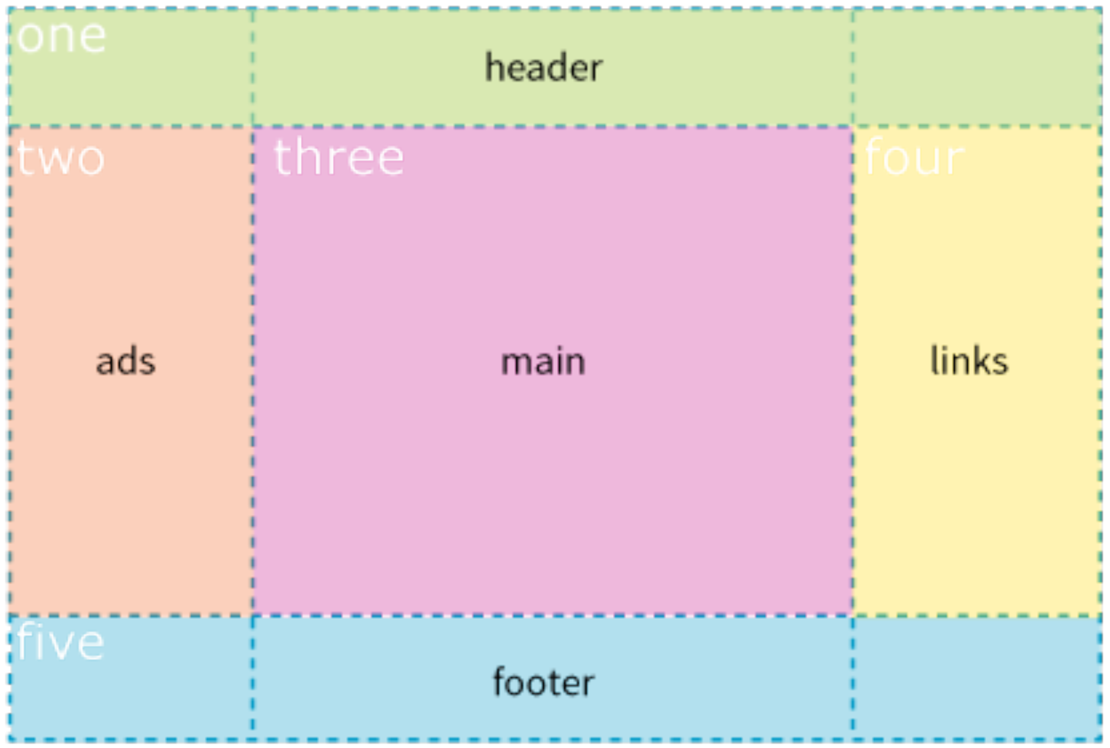
Placing Items on the Grid Using Areas

`grid-area`

Values: *Area name, 1 to 4 line identifiers*

Positions an item in an area created with `grid-template-areas`:

```
#one { grid-area: header; }
#two { grid-area: ads; }
#three { grid-area: main; }
#four { grid-area: links; }
#five { grid-area: footer; }
```



Implicit Grid Behavior

The Grid Layout system does some things for you automatically (**implicit behavior**):

- Generating "-start" and "-end" line names when you name an area (and vice versa)
- Flowing items into grid cells sequentially if you don't explicitly place them
- Adding rows and columns on the fly as needed to fit items

Automatically Generated Tracks

`grid-auto-rows`
`grid-auto-columns`

Values: *List of track sizes*

Provide one or more track sizes for automated tracks. If you provide more than one value, it acts as a repeating pattern.

Example:

Column widths are set explicitly with a template, but columns will be generated automatically with a height of 200 pixels:

```
grid-template-columns: repeat(3, 1fr);
grid-auto-rows: 200px;
```

Flow Direction and Density

`grid-auto-flow`

Values: `row` *or* `column`, `dense` *(optional)*

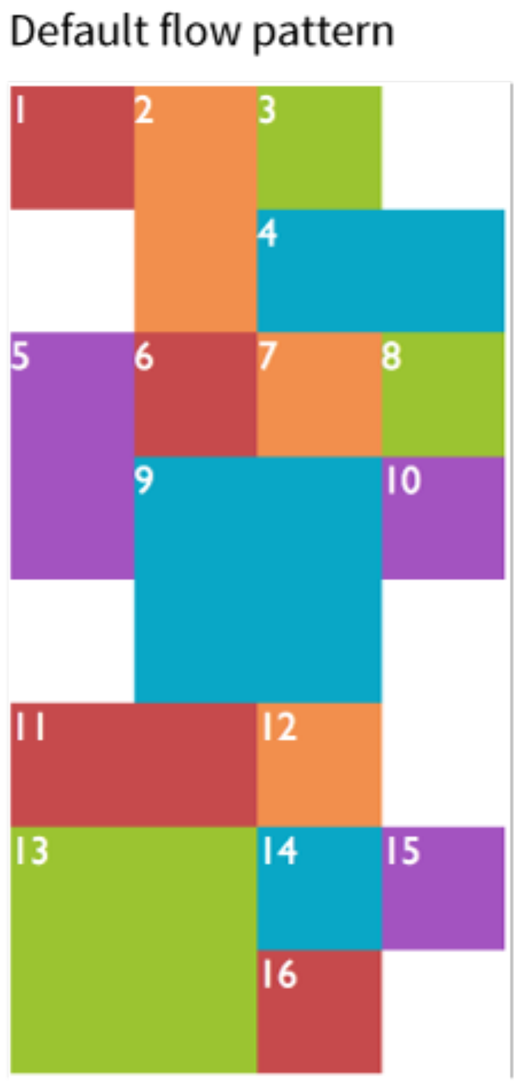
Specifies whether you'd like items to flow in by **`row`** or **`column`**. The default is the writing direction of the document.

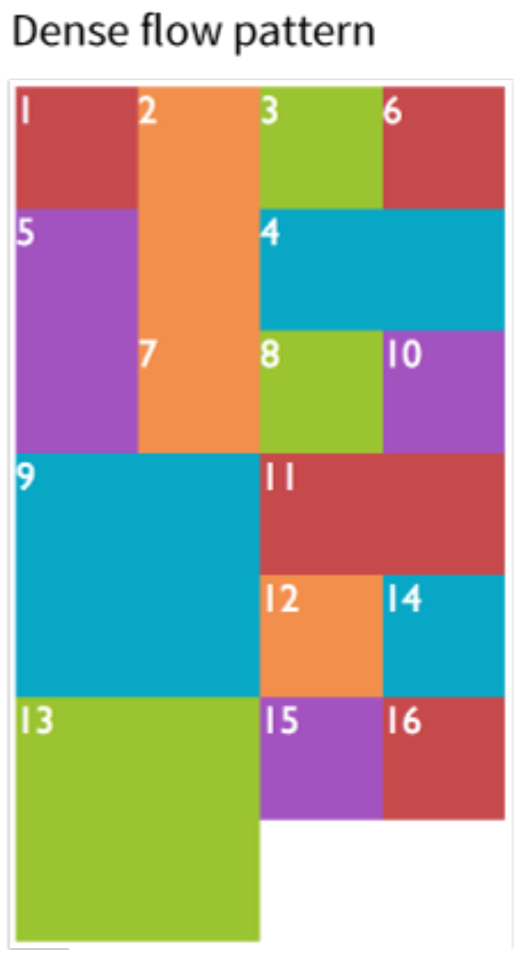
Example:

```
#listings {
  display: grid;
  grid-auto-flow: column dense;
}
```

Flow Direction and Density (cont’d)

The **`dense`** keyword instructs the browser to fill in the grid as densely as possible, allowing items to appear out of order.





The Grid Property (Revisited)

Use the **`auto-flow`** keyword in the shorthand **`grid`** property to indicate that the rows or columns should be generated automatically.

Example:
Columns are established explicitly, but the rows generate automatically. *(Remember, row information goes before the slash.)*

```
grid: auto-flow 200px / repeat(3, 1fr);
```

Because `auto-flow` is included with row information, `grid-auto-flow` is set to `row`.

Spacing Between Tracks

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

Values: *Length (must not be negative)*

`grid-gap`

Values: *`grid-row-gap grid-column-gap`*

Adds space between the row and/or columns tracks of the grid

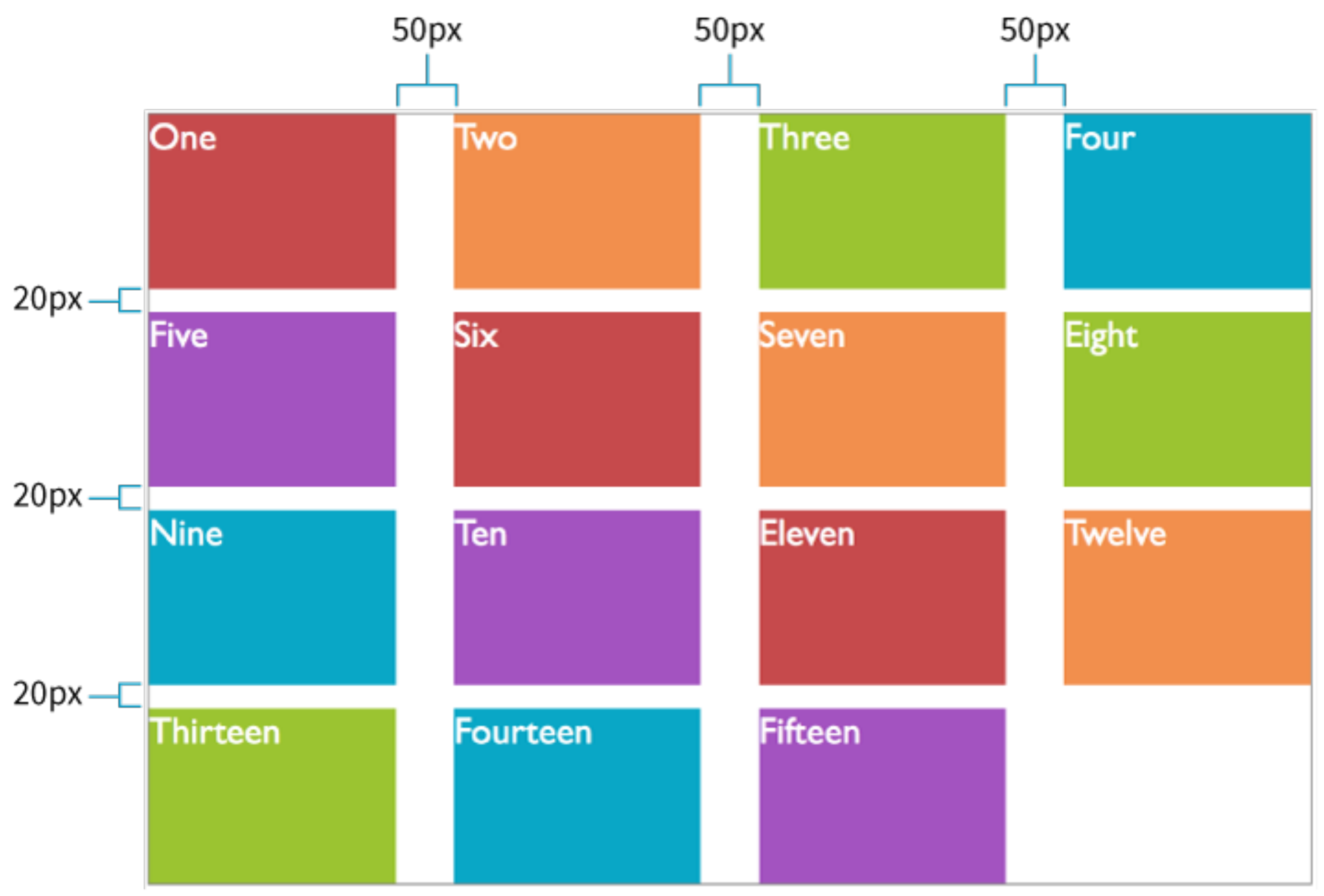
NOTE: These property names will be changing to **`row-gap`**, **`column-gap`**, and **`gap`**, but the new names are not yet supported.

Space Between Tracks (cont'd)

If you want equal space between all tracks in a grid, use a gap instead of creating additional spacing tracks:

```
grid-gap: 20px 50px;
```

(Adds 20px space between rows and 50px between columns)



Item Alignment

`justify-self`
`align-self`

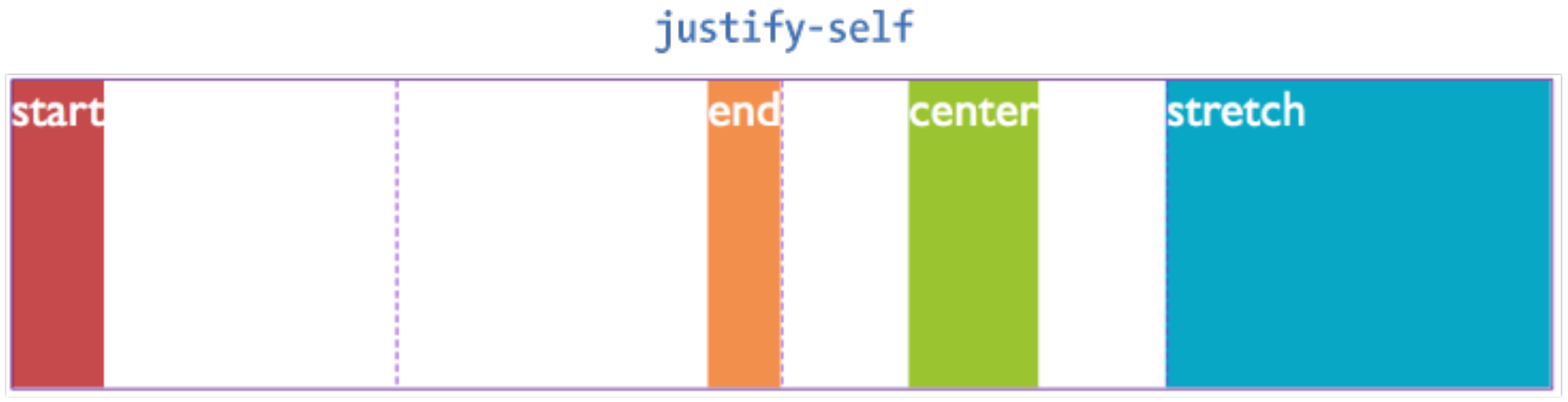
Values: `start`, `end`, `center`, `left`, `right`, `self-start`, `self-end`, `stretch`, `normal`, `auto`

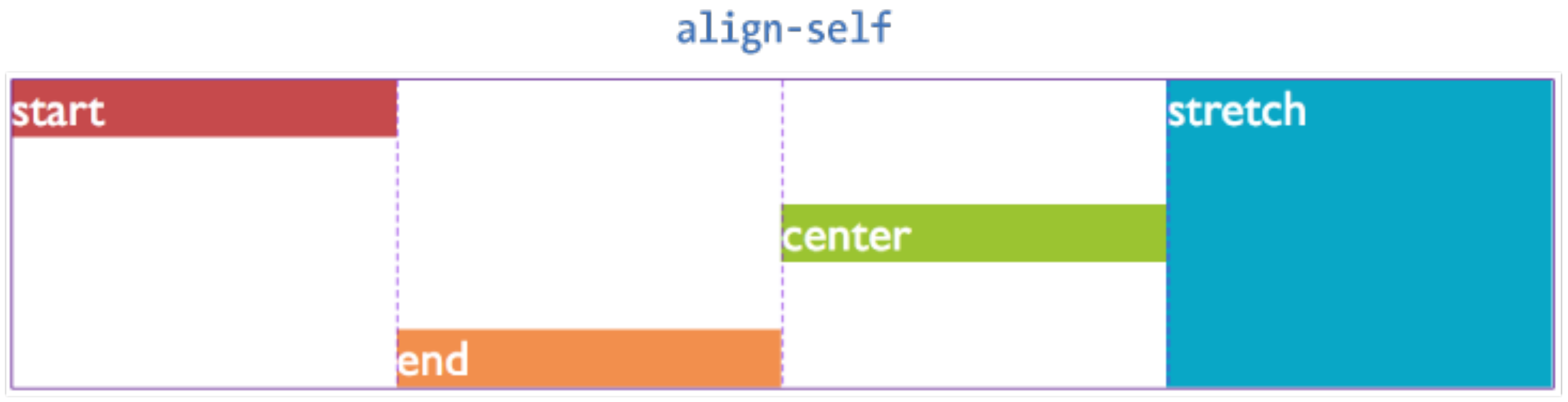
When an item element doesn’t fill its grid cell, you can specify how it should be aligned within the cell.

`justify-self` aligns on the **inline** axis (horizontal for L-to-R languages).

`align-self` aligns on the **block** (vertical) axis.

Item Alignment (cont'd)





NOTE: These properties are applied to the individual **grid item** element.

Aligning All the Items

`justify-items`
`align-items`

Values: `start`, `end`, `center`, `left`, `right`, `self-start`, `self-end`, `stretch`, `normal`, `auto`

These properties align items in their cells all at once. They are applied to the **grid container**.

`justify-items` aligns on the **inline** axis.

`align-items` aligns on the **block** (vertical) axis.

Track Alignment

`justify-content`
`align-content`

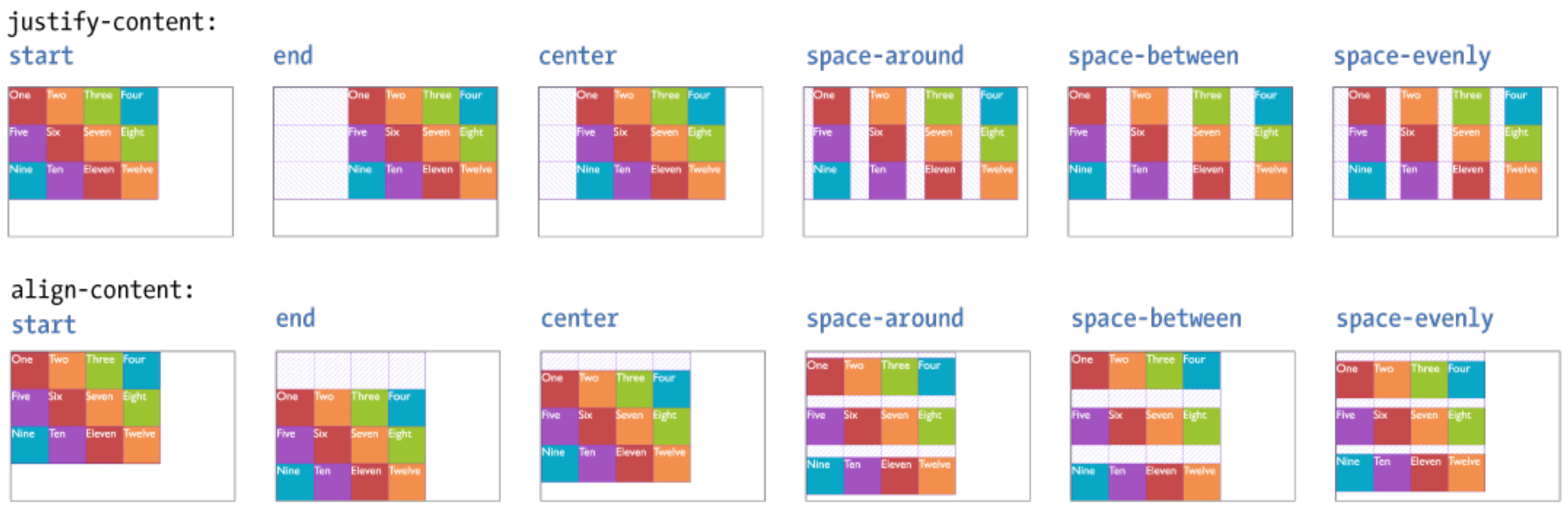
Values: `start`, `end`, `center`, `left`, `right`, `stretch`, `space-around`, `space-between`, `space-evenly`

When the **grid tracks do not fill the entire container**, you can specify how tracks align.

`justify-content` aligns on the **inline** axis (horizontal for L-to-R languages).

`align-content` aligns on the **block** (vertical) axis.

Track Alignment (cont'd)



NOTE: These properties are applied to the **grid container**.

Grid Property Review

Grid container properties

```
display: grid | inline-grid
grid
   grid-template
      grid-template-rows
      grid-template-columns
      grid-template-areas
  grid-auto-rows
  grid-auto-columns
  grid-auto-flow
grid-gap
  grid-row-gap
  grid-column-gap
justify-items
align-items
justify-content
align-content
```

Grid item properties

```
grid-column
   grid-column-start
   grid-column-end
grid-row
   grid-row-start
   grid-row-end
grid-area
justify-self
align-self
```

`order` (not part of Grid Module)

`z-index` (not part of Grid Module)