# CSS: Flexible box layout (flexbox, grid)

Bc. Samuel Tuka

## Outline of today's lecture

History of CSS layout techniques

Flexbox - what is it, its history, all the available properties with examples

Grid - what is it, its history, all the available properties with examples

Comparison between these two layout models

## History of layout techniques

Before flexbox and grid there was not tool created specifically for layout

Table

Floats

Positioning - static, relative, fixed, absolute, sticky

Block and inline-block elements

Web layout history: How we got to grid and flex

## Flexbox and its history

One-dimensional layout model

Idea to create flexbox - 2008 - CSS working group

First working draft published - 2009 - several problems - layout algorithm was slow and

there were divergent details in implementation - Webkit and Firefox

Original specification was based on XUL - technology used by Firefox to produce UI designs

Revision by Tab Atkins, significant revision of the syntax - 2011

Tweener syntax published - late 2011

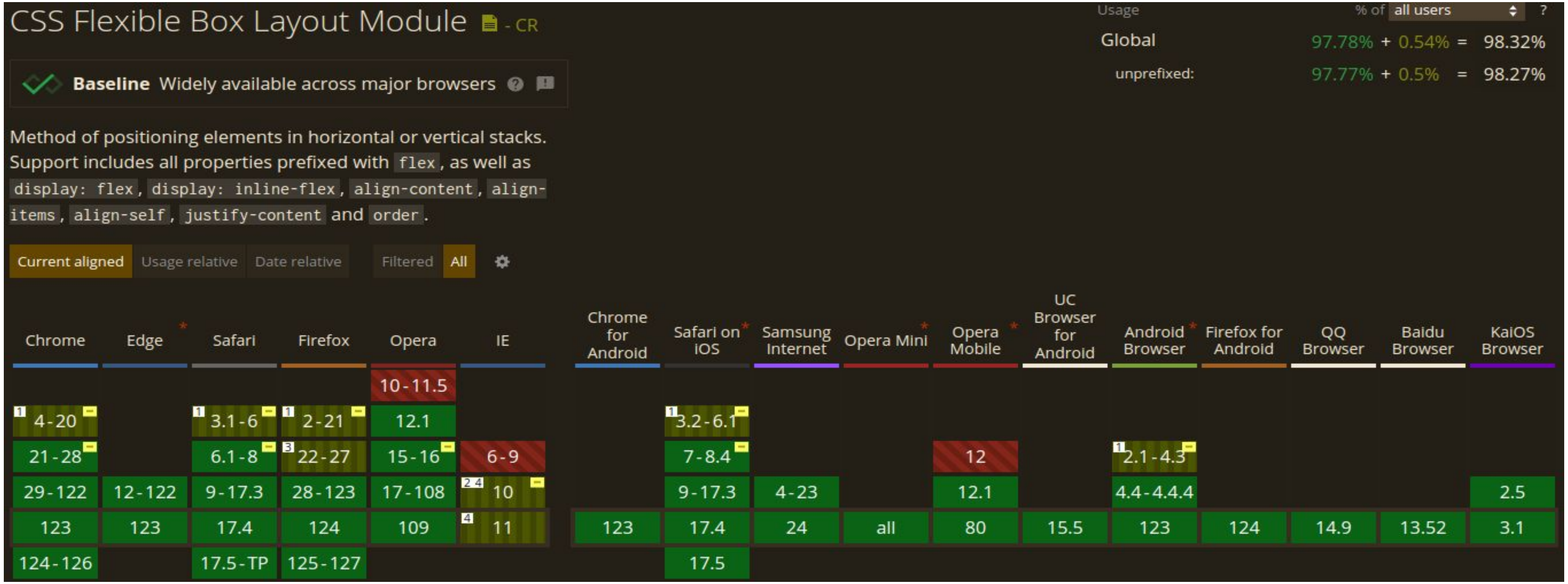
W3C candidate recommendation - 2012

W3C working draft published - 2013

New syntax - 2015

# Browser support flexbox

[CSS Flexible Box Layout Module | Can I use... Support tables for HTML5, CSS3, etc](#)



CSS Flexible Box Layout Module - CR

Usage % of all users

Global 97.78% + 0.54% = 98.32%

unprefixed: 97.77% + 0.5% = 98.27%

**Baseline** Widely available across major browsers

Method of positioning elements in horizontal or vertical stacks. Support includes all properties prefixed with `flex`, as well as `display: flex`, `display: inline-flex`, `align-content`, `align-items`, `align-self`, `justify-content` and `order`.

Current aligned | Usage relative | Date relative | Filtered | All

| Chrome | Edge | Safari | Firefox | Opera | IE | Chrome for Android | Safari on iOS | Samsung Internet | Opera Mini | Opera Mobile | UC Browser for Android | Android Browser | Firefox for Android | QQ Browser | Baidu Browser | KaiOS Browser |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 10-11.5 | | | | | | | | | | | | |
| [1] 4-20 | | [1] 3.1-6 | [1] 2-21 | 12.1 | | | [1] 3.2-6.1 | | | | | | | | | |
| 21-28 | | 6.1-8 | [3] 22-27 | 15-16 | 6-9 | | 7-8.4 | | | 12 | | [1] 2.1-4.3 | | | | |
| 29-122 | 12-122 | 9-17.3 | 28-123 | 17-108 | [2] [4] 10 | | 9-17.3 | 4-23 | | 12.1 | | 4.4-4.4.4 | | | | 2.5 |
| 123 | 123 | 17.4 | 124 | 109 | [4] 11 | 123 | 17.4 | 24 | all | 80 | 15.5 | 123 | 124 | 14.9 | 13.52 | 3.1 |
| 124-126 | | 17.5-TP | 125-127 | | | | 17.5 | | | | | | | | | |

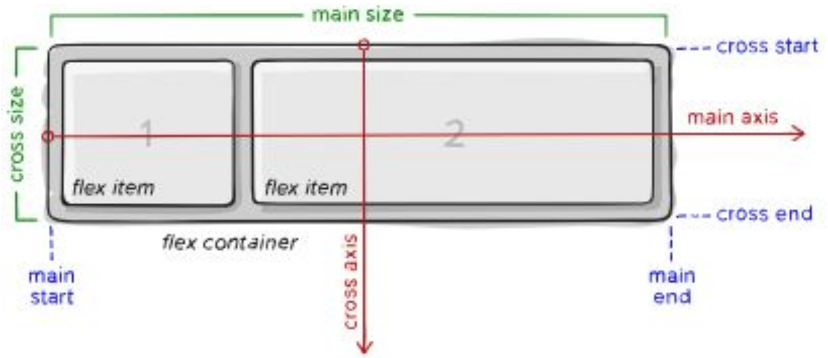
## Basic flex terms

Flex container

Flex item

Main axis

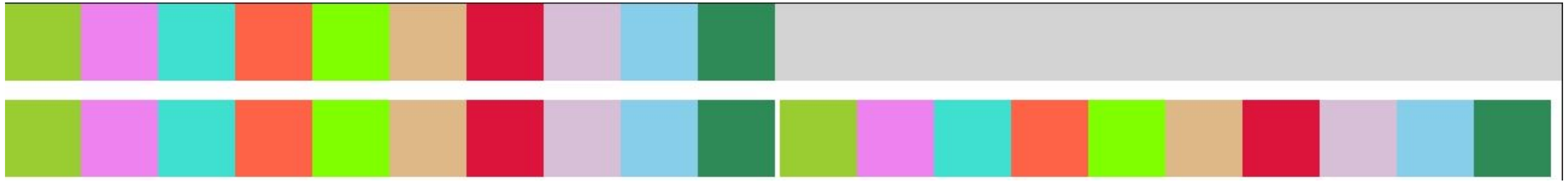
Cross axis



## display property

flex

inline-flex



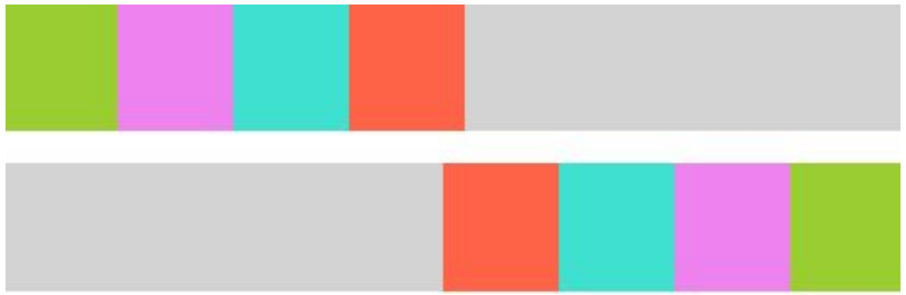
## flex-direction property

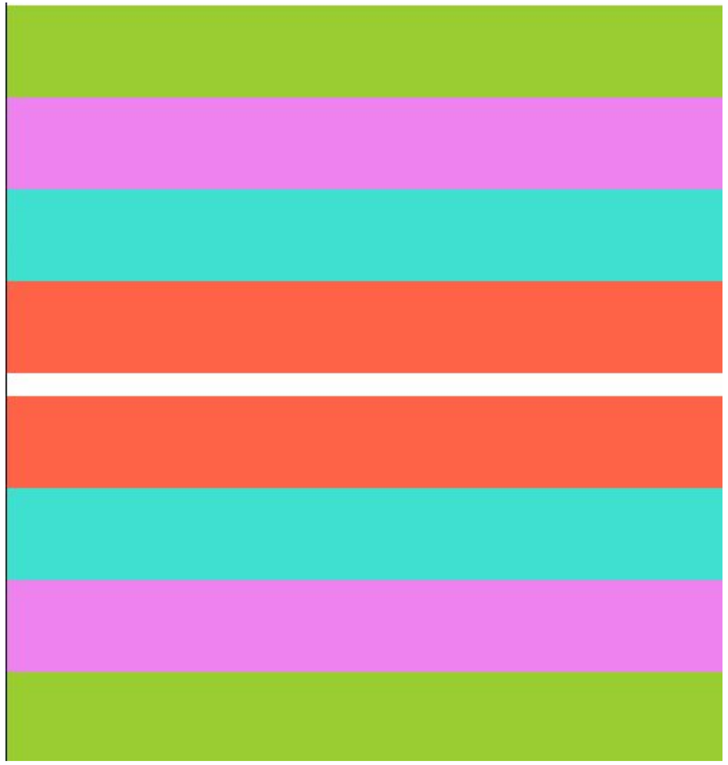
row (default)

column

row-reverse

column-reverse



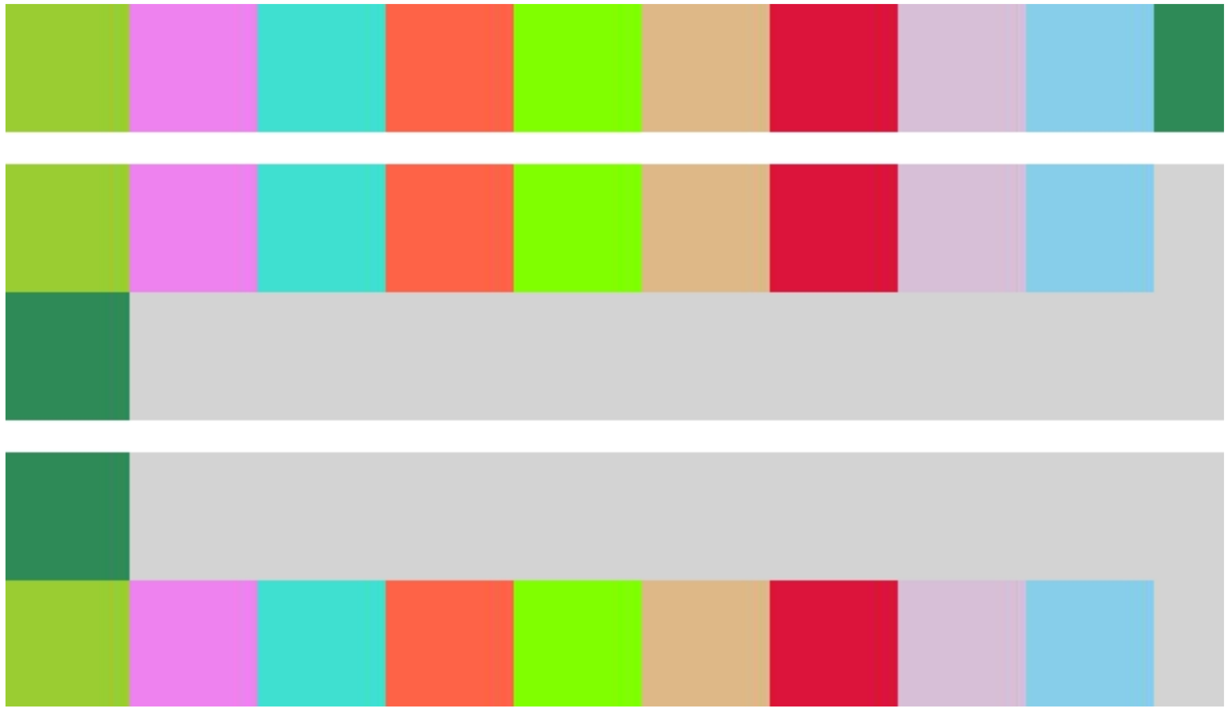


# flex-wrap property

no-wrap (default)

wrap

wrap-reverse



## flex-flow property

Shorthand property for flex-direction and flex-wrap properties

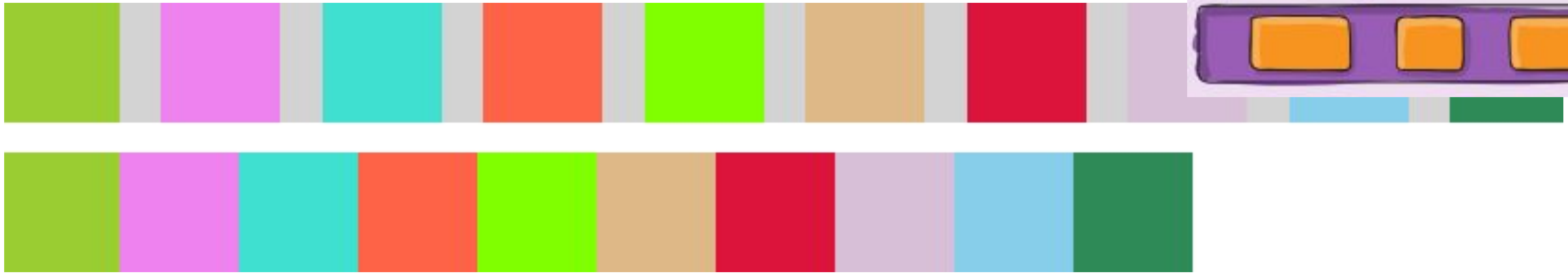
Default value is row nowrap

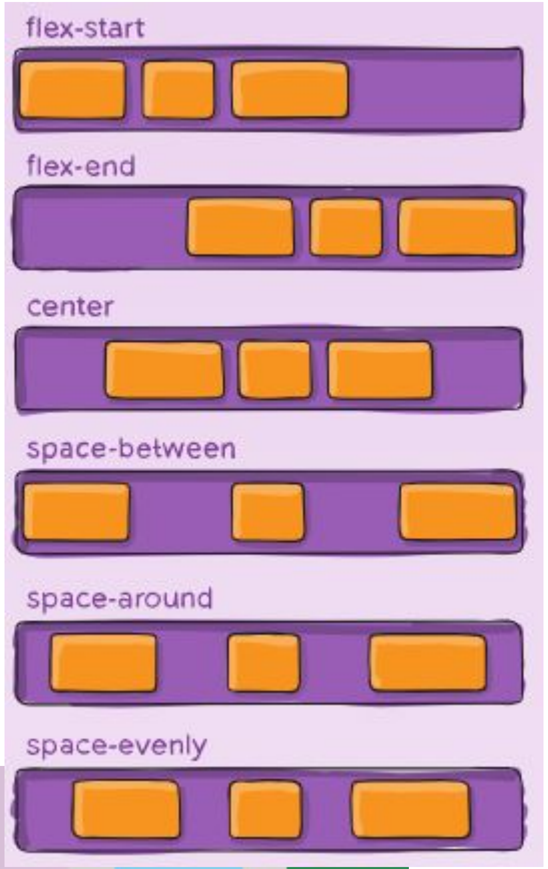
## justify-content property

Alignment along the main axis

Default value is flex-start

Another options are: start, end, left, right

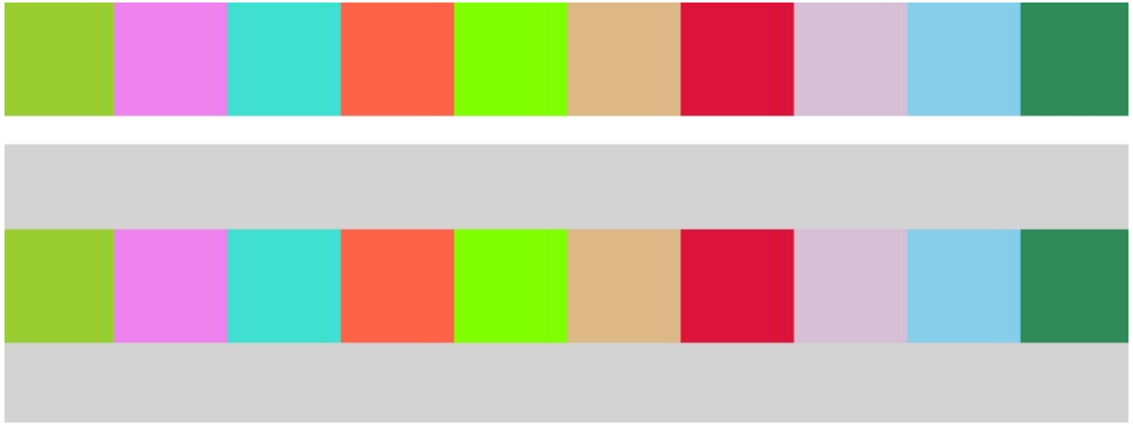


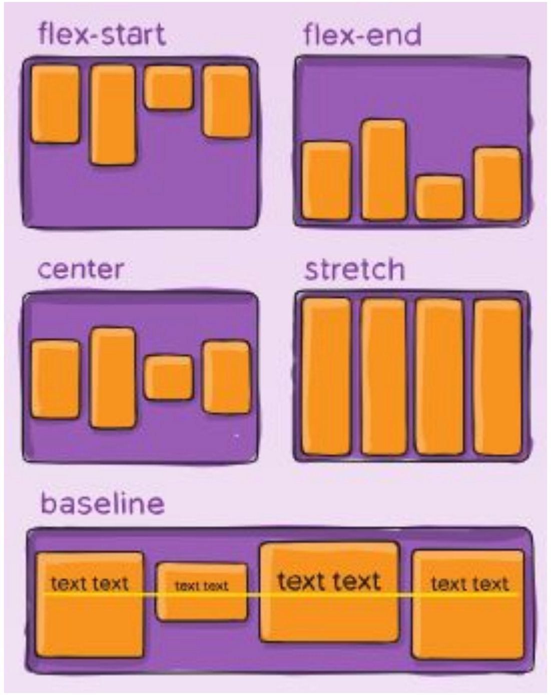


## align-items property

Default value is stretch

Other options are baseline, first baseline, last baseline, start, end, self-start, self-end





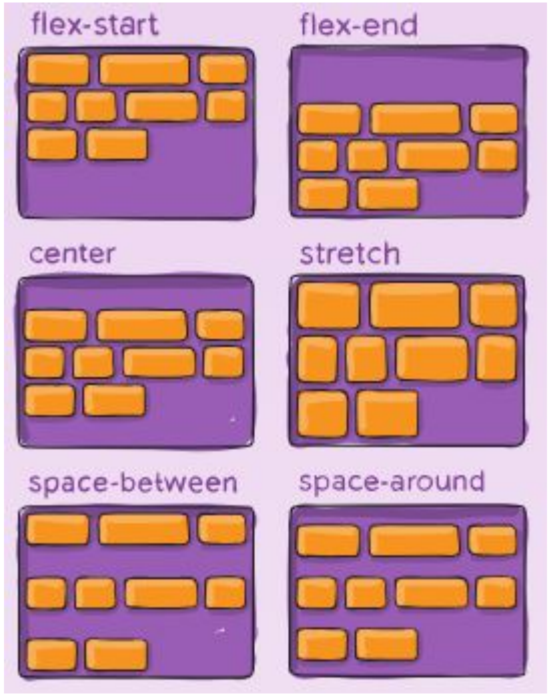
## align-content property

Used to align flex container lines when there is extra space in the cross axis.

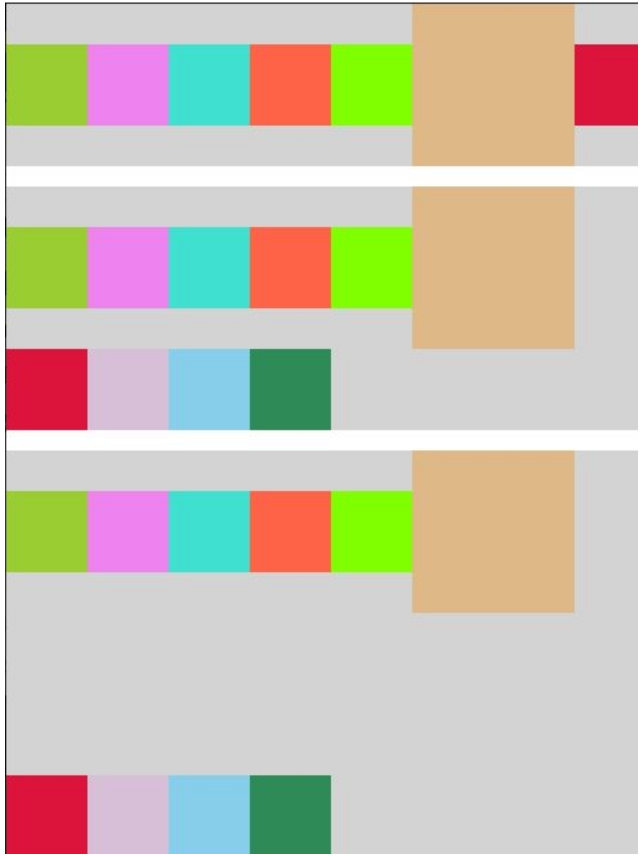
This will take effect only with flex-wrap property set to wrap or wrap-reverse.

Other options are space evenly, start, end, baseline, first baseline, last baseline

Default value is normal as if no value was set



## align-content example

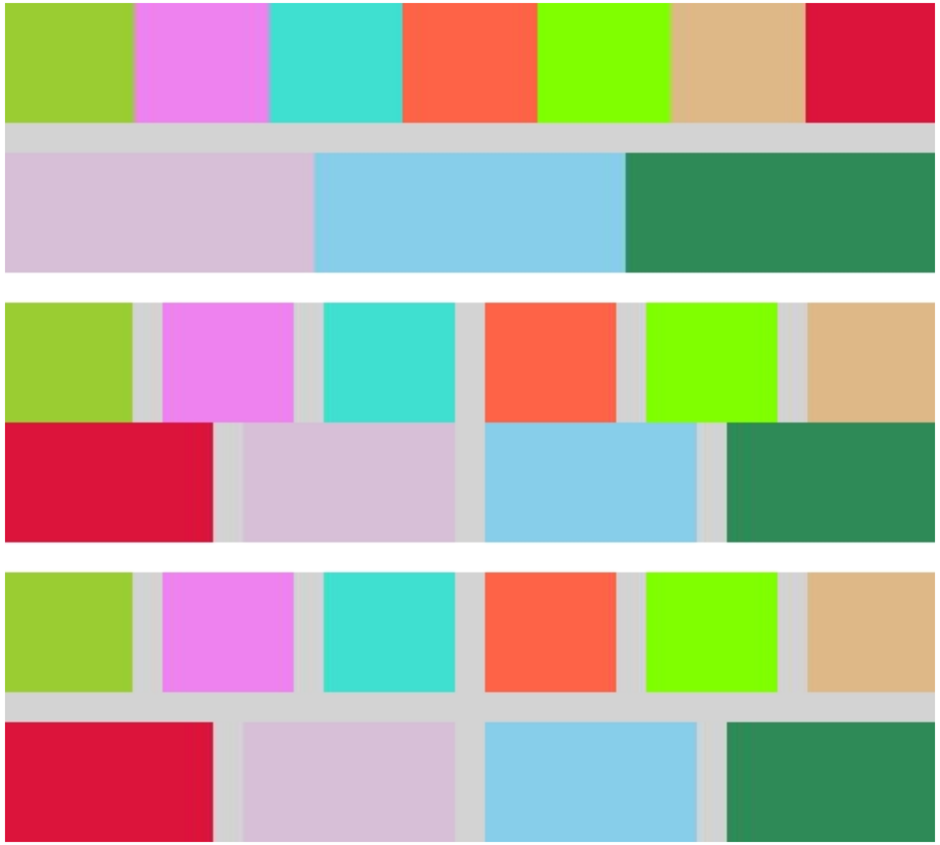


## gap, row-gap, column-gap properties

Control of the space between flex items

Space is applied only between items

not on the edges



## order property

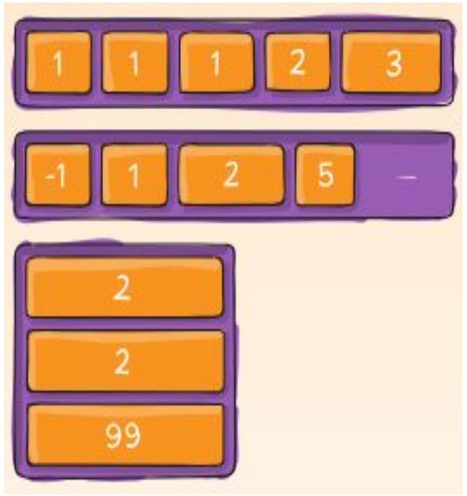
By default items are laid out in the source order

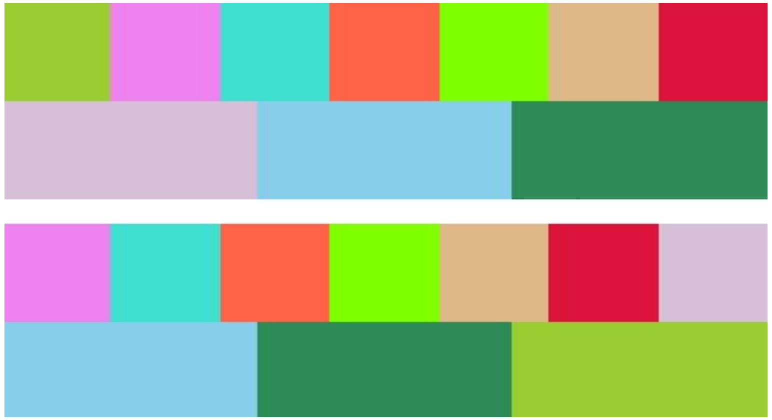
We can change it manually

Items with the same order revert to source order

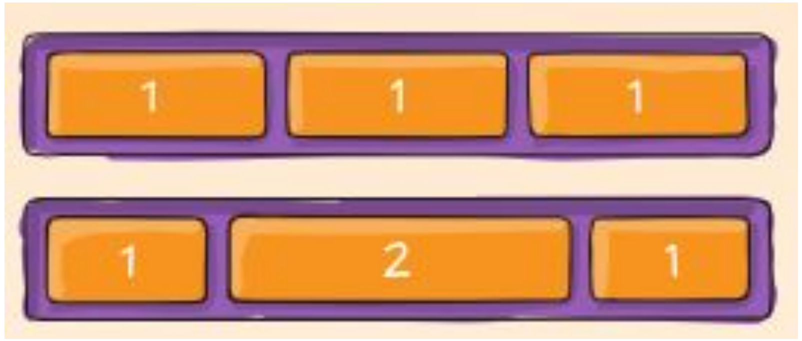
Default value is 0

There can be problem with accessibility





## flex-grow property



Ability for a flex item to grow

Accepts a unitless value - value servers as a proportion

Amount of the available space the item should take up

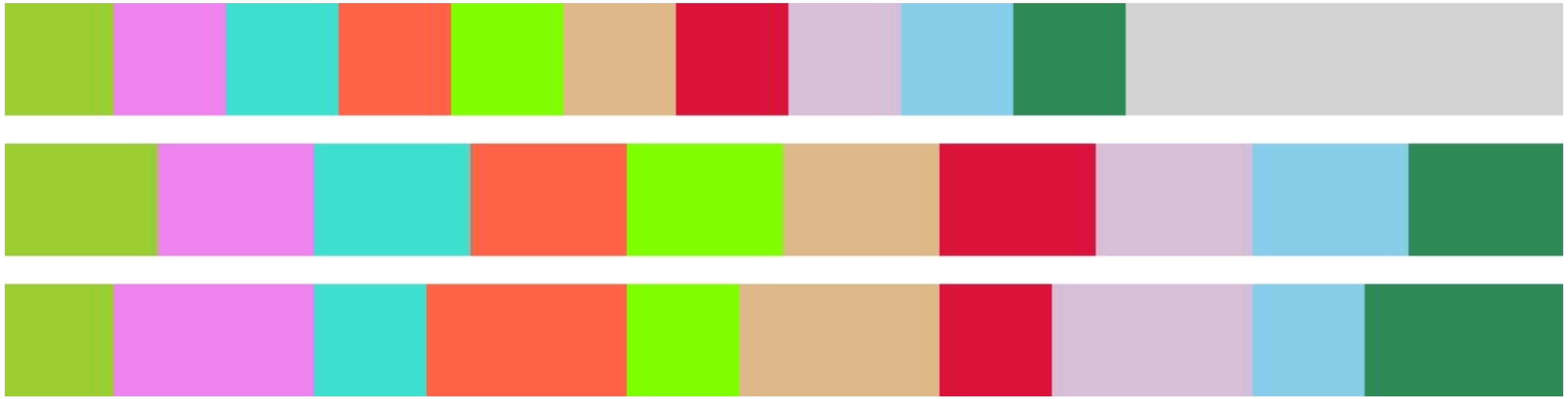
All items have same value -> free space is distributed equally

One item - flex-grow: 1, second item flex-grow: 2 -> second item will (or will try) take up twice as much of the free space then the first item

Negative numbers are invalid

Default value is 0

## flex-grow example



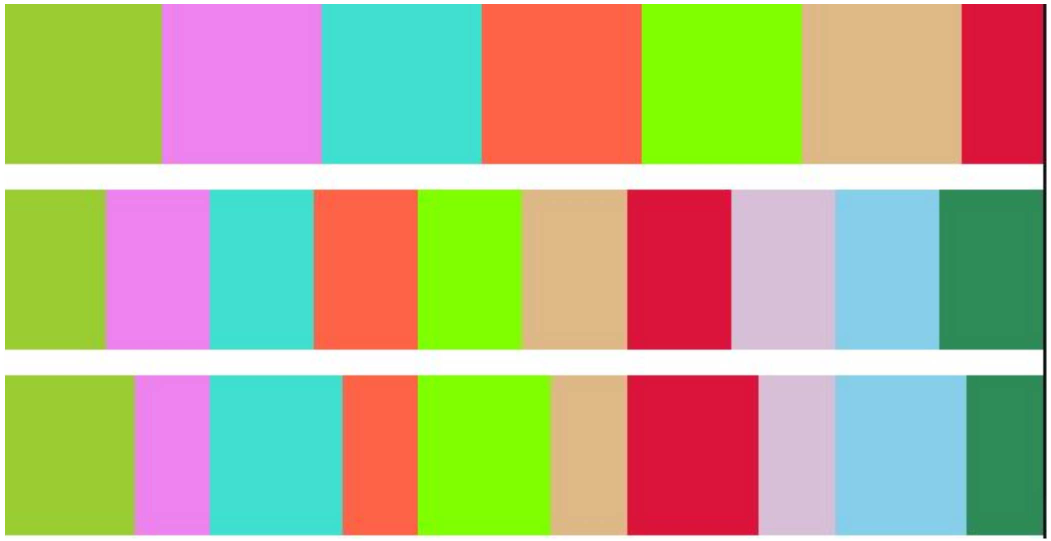
# flex-shrink property

Ability for a flex item to shrink if necessary

Default value is 1

One item - flex-shrink: 1, second item - flex-shrink: 3, the second item will shrink 3 times faster than the first one

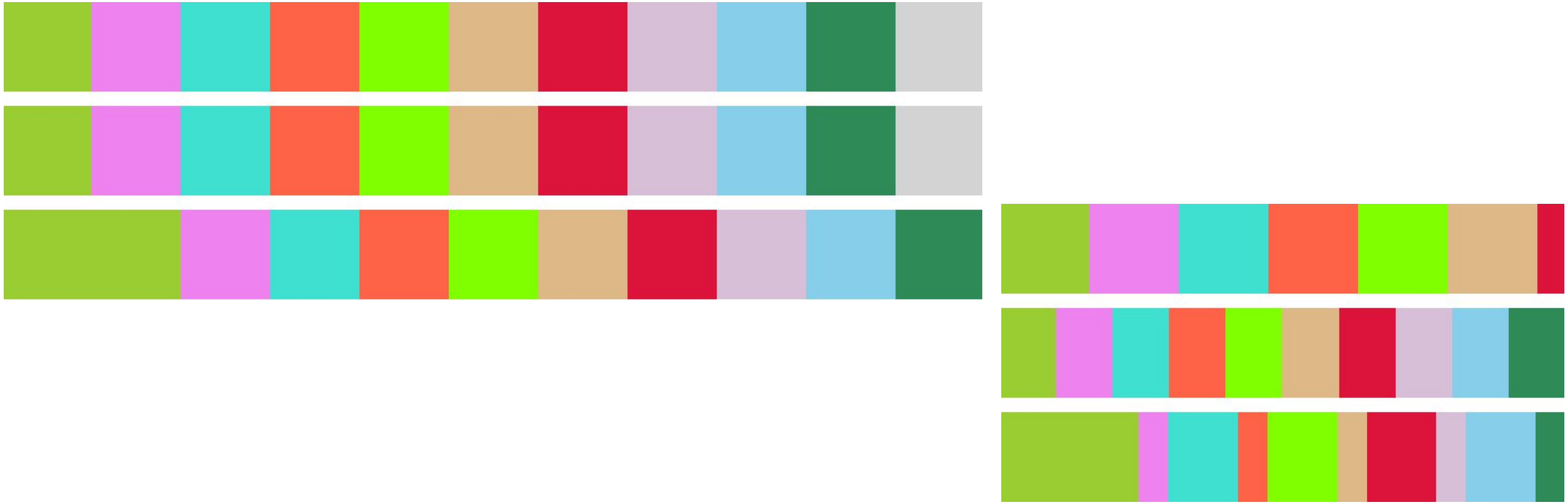
Negative numbers are invalid



## flex-basis property

Defines default size of an element before the remaining space is distributed

We can use almost every length like %, rem, em etc. or a keyword.



## flex property

Shorthand property for flex-grow, flex-shrink and flex-basis

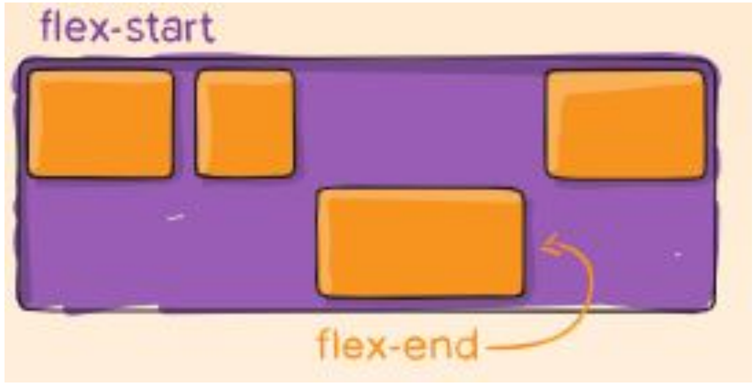
Flex-shrink and flex-basis are optional

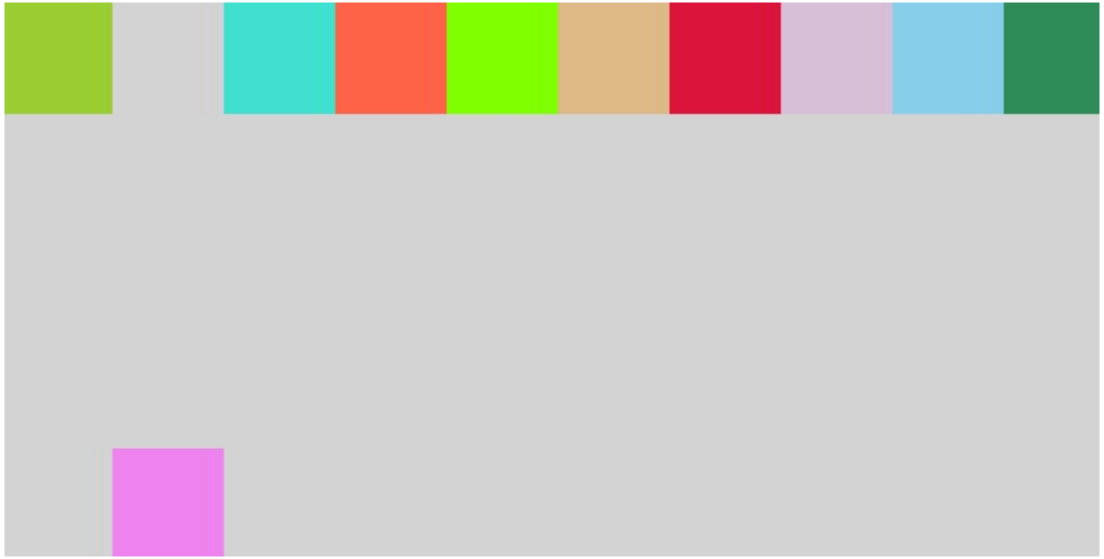
Default value is 0 1 auto

It is recommended to use this property instead of setting individual properties

## align-self property

Allows to override the default alignment





## When should we use flexbox?

Align items

Small layout design to implement with a few rows or a few columns

Content-first design

## Grid and its history

Two-dimensional grid-based layout

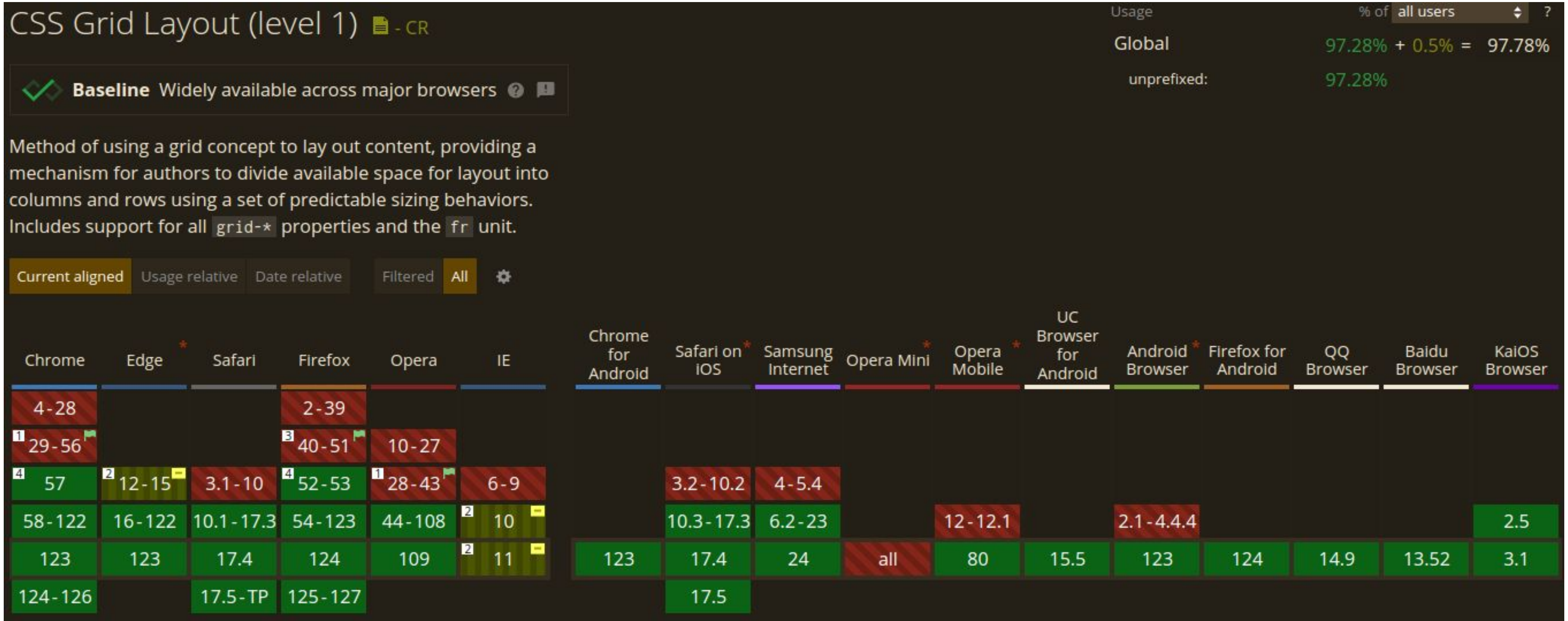
First comprehensive draft of a grid layout for CSS was created by Phil Cupp at MIcrosoft in 2011

Several interactions in the CSS Working Group let to restructured syntax - led primarily by Elika Etemad and Tab Atkins.

2017 - Chrome, Firefox, Safari and Edge all started to support CSS grid without vendor prefixes

# Browser support grid

https://caniuse.com/css-grid



CSS Grid Layout (level 1) - CR

Usage % of all users ?

Global 97.28% + 0.5% = 97.78%

unprefixed: 97.28%

**Baseline** Widely available across major browsers

Method of using a grid concept to lay out content, providing a mechanism for authors to divide available space for layout into columns and rows using a set of predictable sizing behaviors. Includes support for all `grid-*` properties and the `fr` unit.

Current aligned | Usage relative | Date relative | Filtered | All

| Chrome | Edge * | Safari | Firefox | Opera | IE | Chrome for Android | Safari on iOS * | Samsung Internet | Opera Mini * | Opera Mobile * | UC Browser for Android | Android Browser * | Firefox for Android | QQ Browser | Baidu Browser | KaiOS Browser |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4-28 | | | 2-39 | | | | | | | | | | | | | |
| [1] 29-56 | | | [3] 40-51 | 10-27 | | | | | | | | | | | | |
| [4] 57 | [2] 12-15 | 3.1-10 | [4] 52-53 | [1] 28-43 | 6-9 | | 3.2-10.2 | 4-5.4 | | | | | | | | |
| 58-122 | 16-122 | 10.1-17.3 | 54-123 | 44-108 | [2] 10 | | 10.3-17.3 | 6.2-23 | | 12-12.1 | | 2.1-4.4.4 | | | | 2.5 |
| 123 | 123 | 17.4 | 124 | 109 | [2] 11 | 123 | 17.4 | 24 | all | 80 | 15.5 | 123 | 124 | 14.9 | 13.52 | 3.1 |
| 124-126 | | 17.5-TP | 125-127 | | | | 17.5 | | | | | | | | | |

## Basic grid terms

Grid container

Grid item

Row

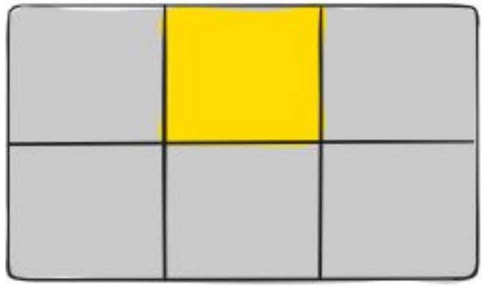
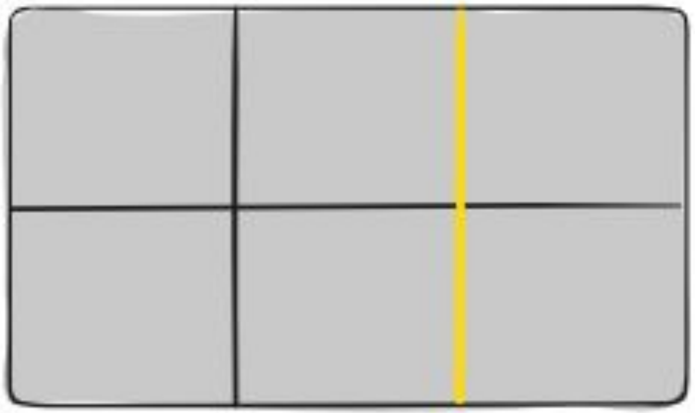
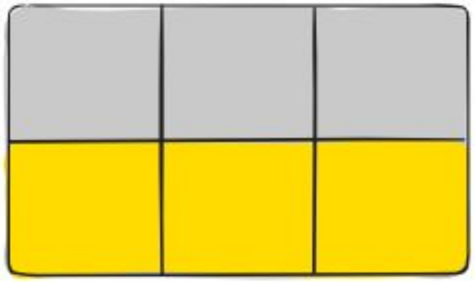
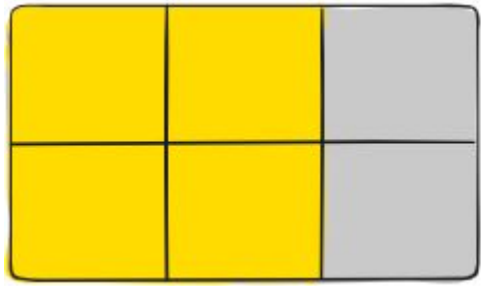
Column

Grid line

Grid cell

Grid track

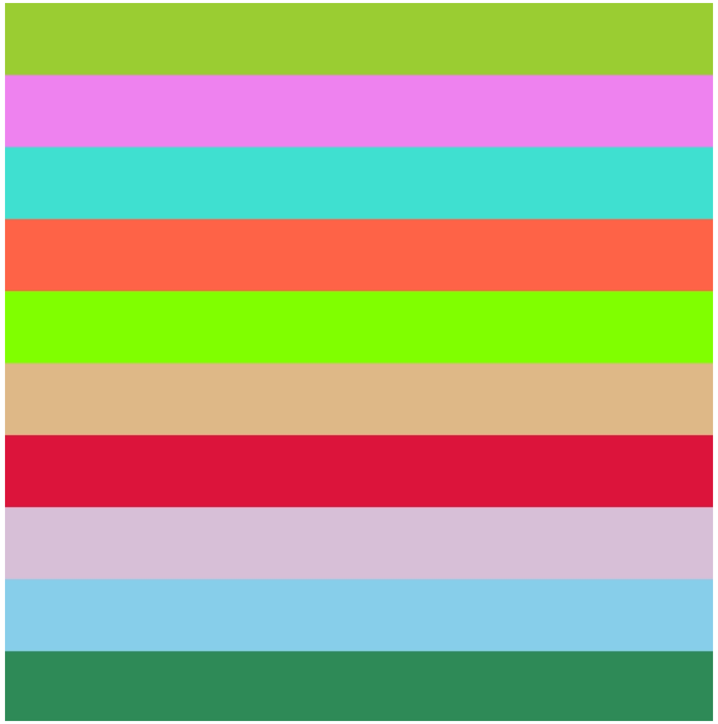
Grid area

## display property

Defines the element as a grid container and establishes a new grid formatting context for its contents

grid - block level grid



## grid-template-columns && grid-template-rows

Defines the columns and rows of the grid with a space-separated list of values

These values represents the track size (size of row/column)

Values can be of any length - %, em, rem etc.

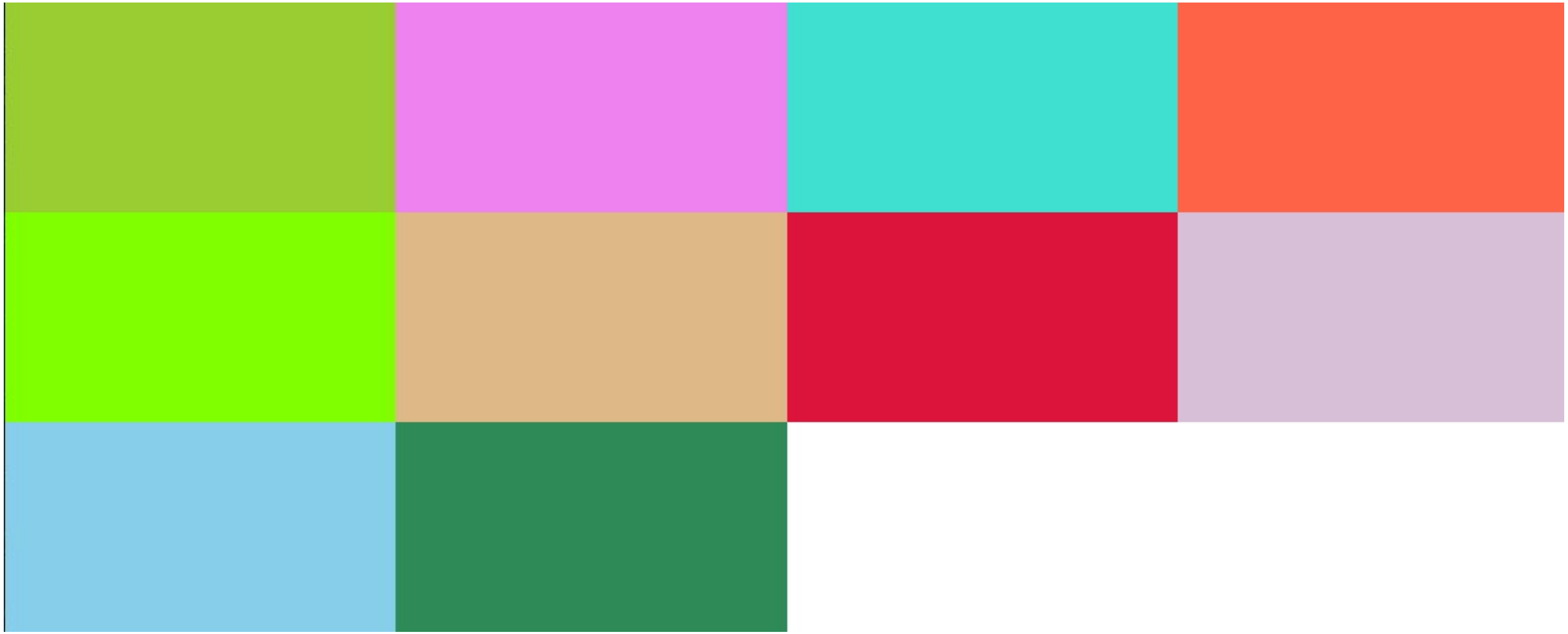
fr unit is very often used when using grid - it is fraction of the free space in the grid

Line names can be specified when declaring columns and row sizes

Different functions can be used like minmax(), repeat(),

Keywords can be used as well - auto, min-content, max-content

## grid-template-columns/rows example



# grid-template-columns && grid-template-rows

When we declare columns and rows, the lines are automatically created and assigned numbers (there are also negative numbers so we can reference the line from the last row to the beginning).

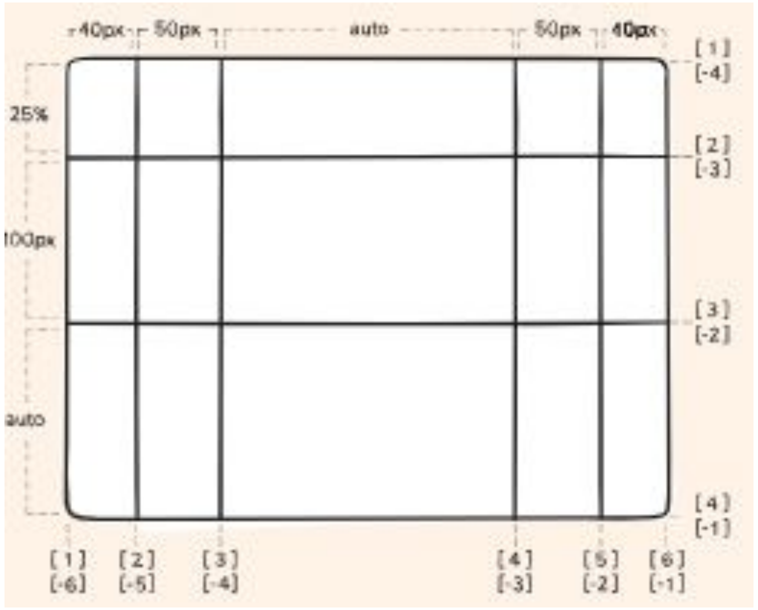
Lines can be explicitly named

Bracket syntax is used for naming lines

Line can have more than one name

If multiple lines share the same name, they

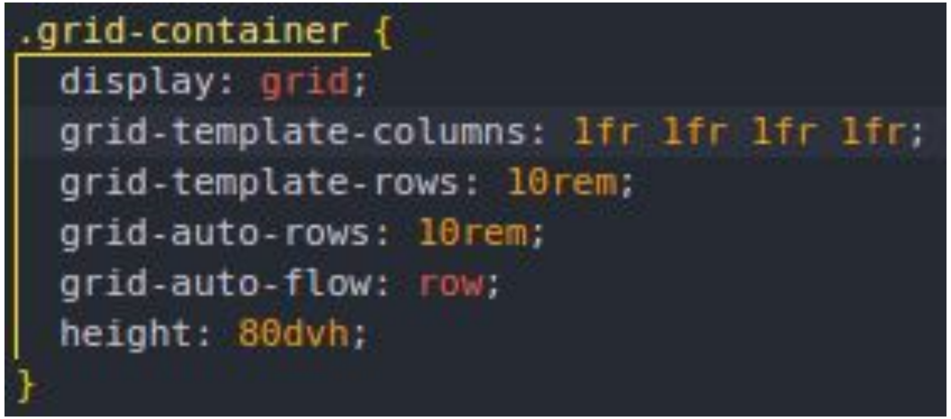
can be referenced by their line name and count



## fr - fraction unit

Fraction of the free space of the grid container

The free space is calculated after any non-flexible items



```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 10rem;
  grid-auto-rows: 10rem;
  grid-auto-flow: row;
  height: 80dvh;
}
```

## grid-template-areas property

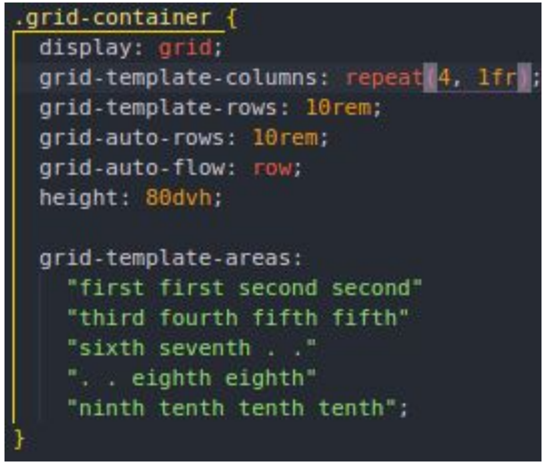
Defines a grid template by referencing the names of the grid areas which are specified with the grid-area property

Repeating the name causes the content to span those cells

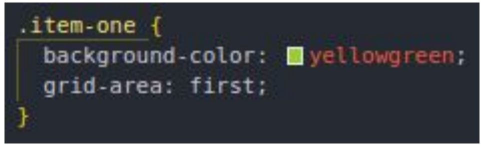
Period signifies empty cell

none - no grid areas are defined

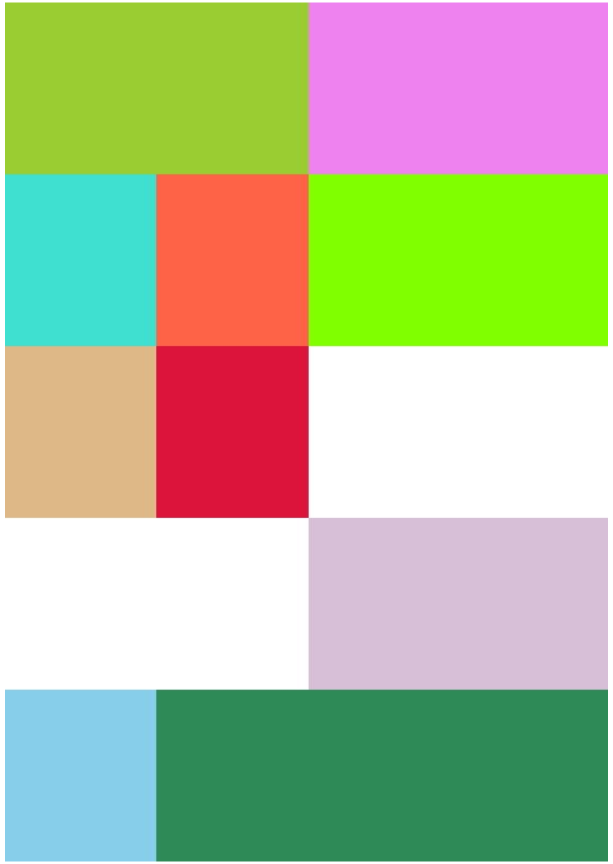
# grid-template-areas example



```
.grid-container {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-template-rows: 10rem;
  grid-auto-rows: 10rem;
  grid-auto-flow: row;
  height: 80dvh;

  grid-template-areas:
    "first first second second"
    "third fourth fifth fifth"
    "sixth seventh . ."
    ". . eighth eighth"
    "ninth tenth tenth tenth";
}
```



```
.item-one {
  background-color: 🟩 yellowgreen;
  grid-area: first;
}
```



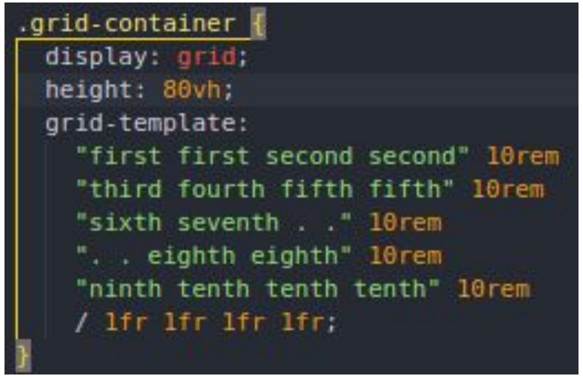
## grid-template

Shorthand for setting grid-template-rows, grid-template-columns and grid-template-areas

none - sets all properties to their initial values

It is not recommended to use this property, because it does not reset implicit grid properties - grid-auto-columns, grid-auto-rows, grid-auto-flow

grid property should be used instead of grid-template



```css
.grid-container {
  display: grid;
  height: 80vh;
  grid-template:
    "first first second second" 10rem
    "third fourth fifth fifth" 10rem
    "sixth seventh . ." 10rem
    ". . eighth eighth" 10rem
    "ninth tenth tenth tenth" 10rem
    / 1fr 1fr 1fr 1fr;
}
```

## gap, column-gap, row-gap properties

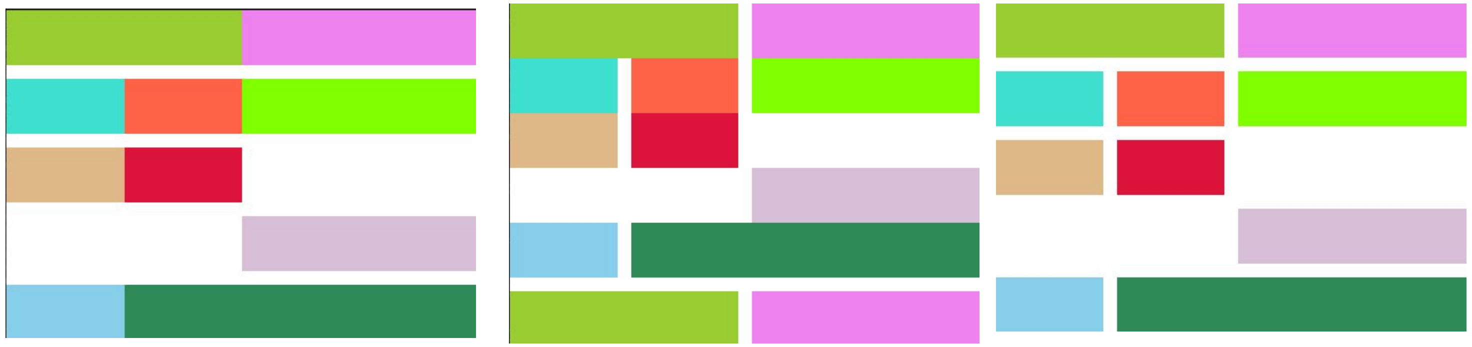
Specifies the size of the grid lines

Can be any length value

Space is only created between columns/rows, not on the outer edges

gap - shorthand property for row-gap and column-gap

If no row-gap is specified, it is set to the same value as column-gap

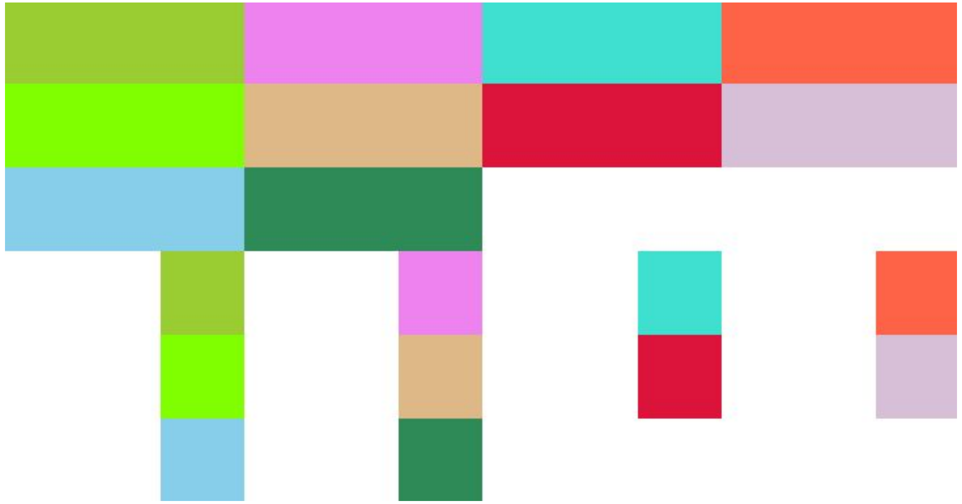


## justify-items property

Aligns grid items along the row axis

Value applies to all grid items inside the container

Different values - start, end, center, stretch (default value)

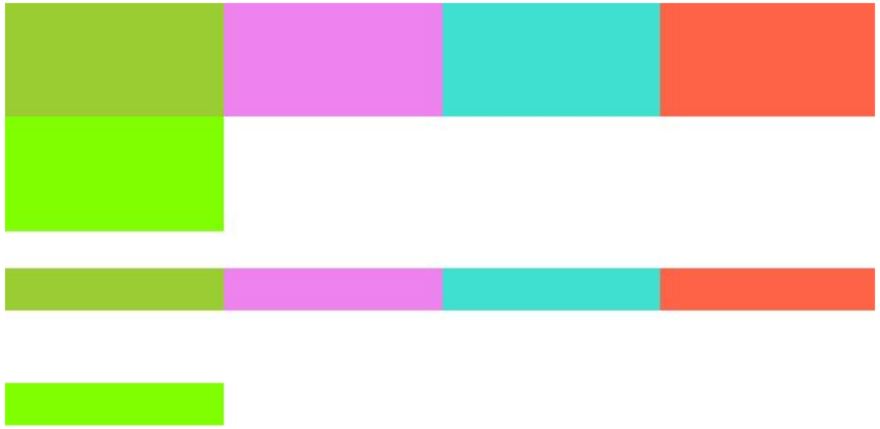


# align-items property

Aligns grid items along column axis

Value applies to all grid items inside the container

Different values - stretch (default), start, end, center, baseline (depends on the text, first baseline and last baseline modifiers in the case of multi line text)

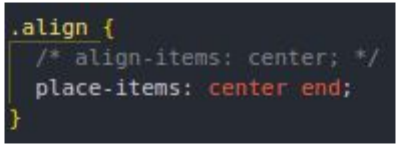


## place-items property

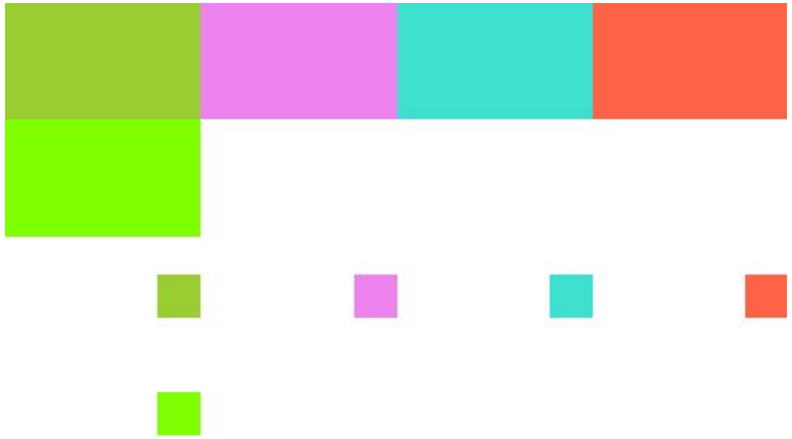
Sets both the align-items and justify-items properties in a single declaration

First value sets align-items, second value justify-items

If the second value is missing, the first value is assigned to both properties



```
.align {
  /* align-items: center; */
  place-items: center end;
}
```



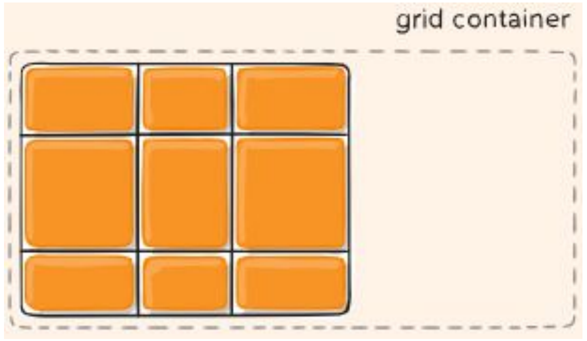
## justify-content property

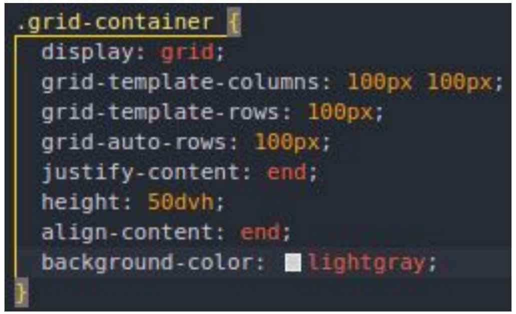
Set the alignment of the grid within the grid container

Aligns grid along row axis

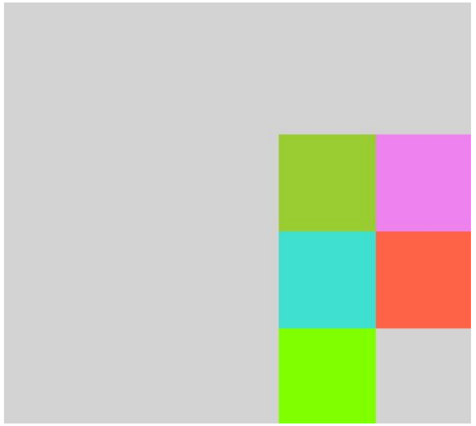
In case when the grid is smaller than the grid container

Different values - start, end, center, stretch, space-around, space-between, space-evenly





```
.grid-container {
    display: grid;
    grid-template-columns: 100px 100px;
    grid-template-rows: 100px;
    grid-auto-rows: 100px;
    justify-content: end;
    height: 50dvh;
    align-content: end;
    background-color: lightgray;
}
```



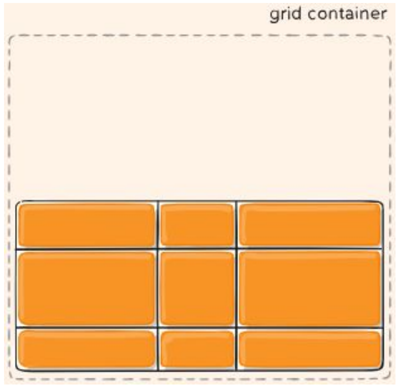
## align-content property

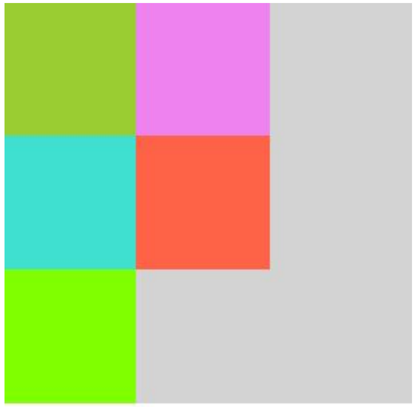
Aligns the grid along the column axis

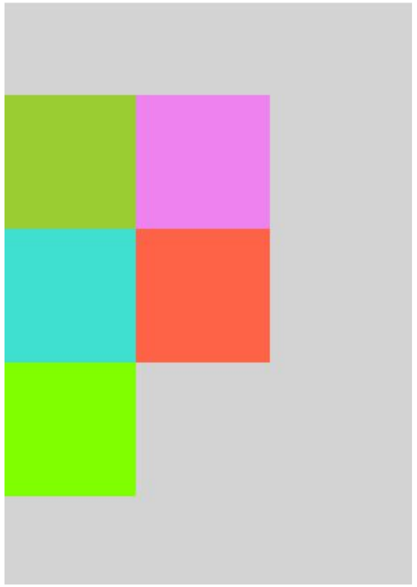
In case of smaller grid then grid container

Different options - start, end, center, stretch,

space-around, space-between, space-evenly





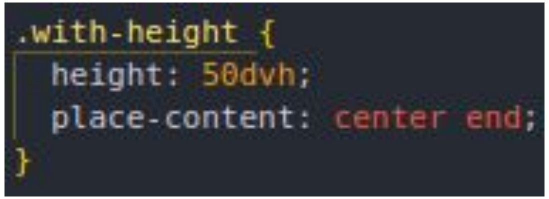


## place-content property

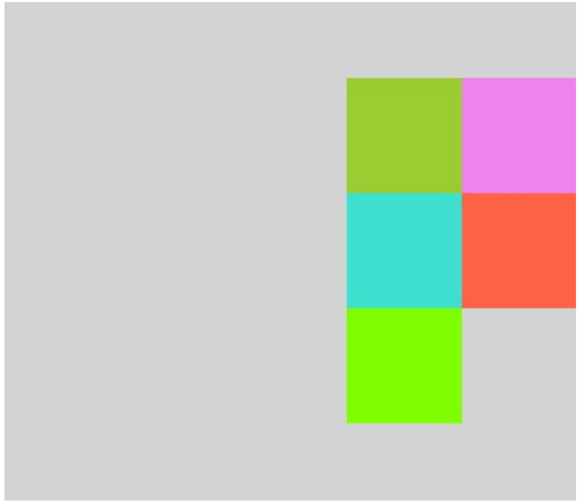
Sets both the align-content and justify-content properties

First value sets align-content, second value justify-content

If the second value is omitted, the first value is assigned to both properties



```
.with-height {
  height: 50dvh;
  place-content: center end;
}
```

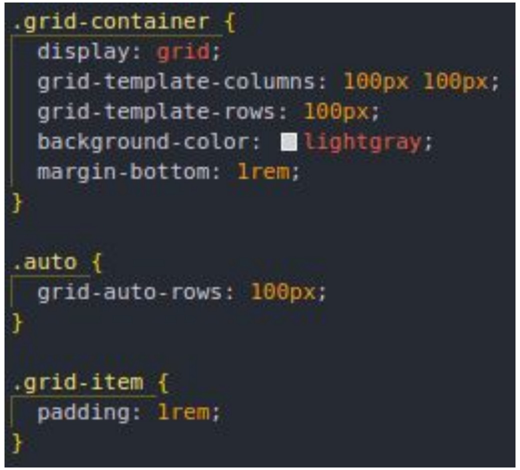


## grid-auto-columns && grid-auto-rows

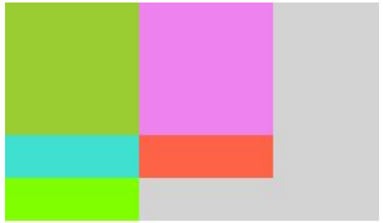
Specifies the size of any auto-generated grid tracks - implicit tracks

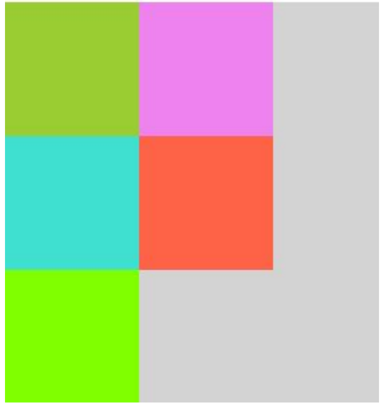
Implicit tracks are created when there are more grid items then cells in the grid or when grid item is placed outside of the explicit grid

Value can be any length - %, rem em etc. Also fr unit can be used



```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px;
  grid-template-rows: 100px;
  background-color: lightgray;
  margin-bottom: 1rem;
}

.auto {
  grid-auto-rows: 100px;
}

.grid-item {
  padding: 1rem;
}
```





## grid-auto-flow property

If we do not place grid item explicitly on the grid, then auto-placement will be used

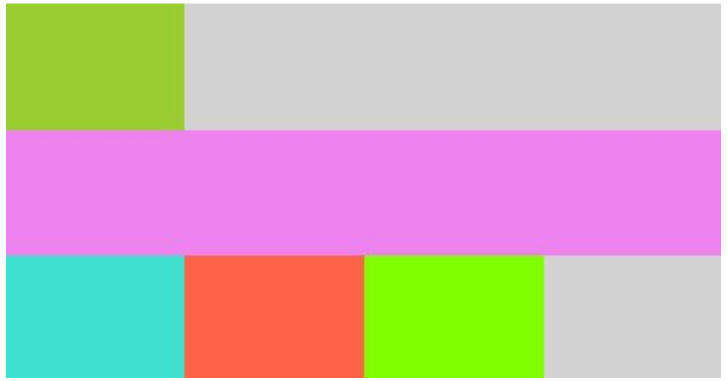
grid-auto-flow is used to control, how the auto-placement will work

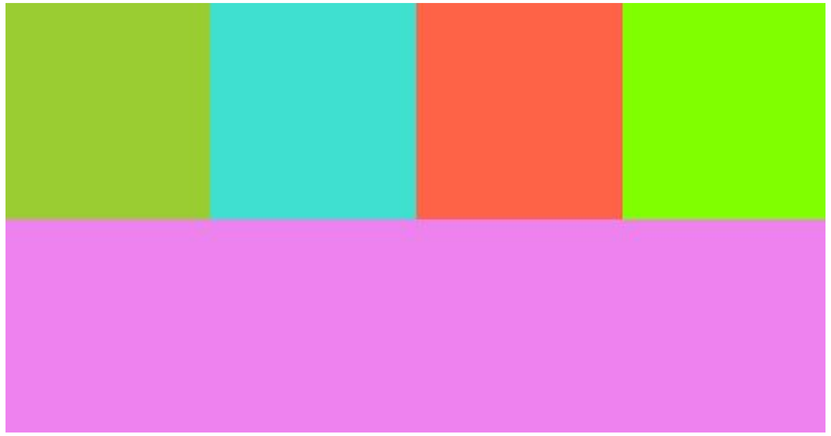
Different options - row, column, dense

Dense - will attempt to fill in holes in the grid if smaller items come up later

There can be problem with accessibility when using dense

## grid auto-flow dense example





## grid-column-start/end, grid-row-start/end

Determines a grid item`s location within the grid by referring to specific grid lines

grid-column-start, grid-row-start is the line where the item begins

grid-column-end/grid-row-end is the line where the item ends

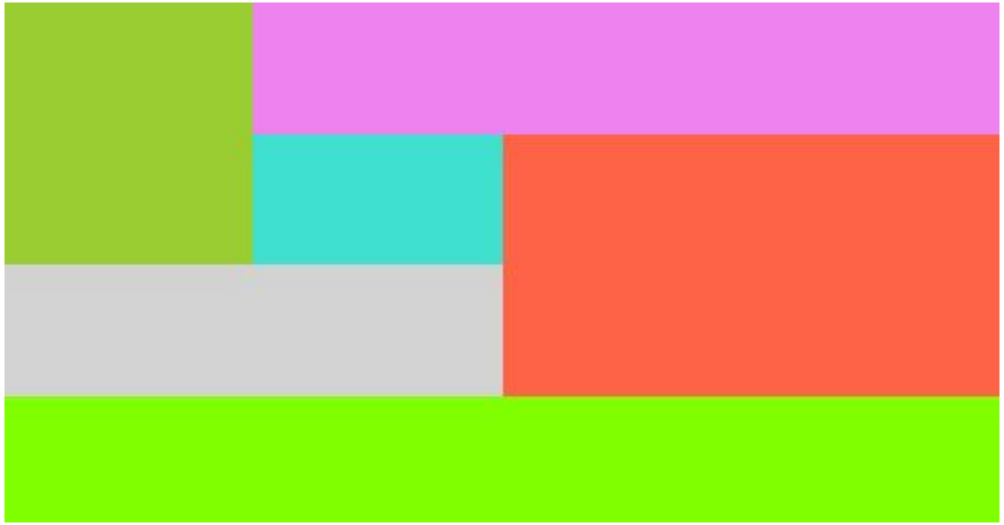
value of these properties can be number to refer to numbered grid line or a name to refer to a named grid line

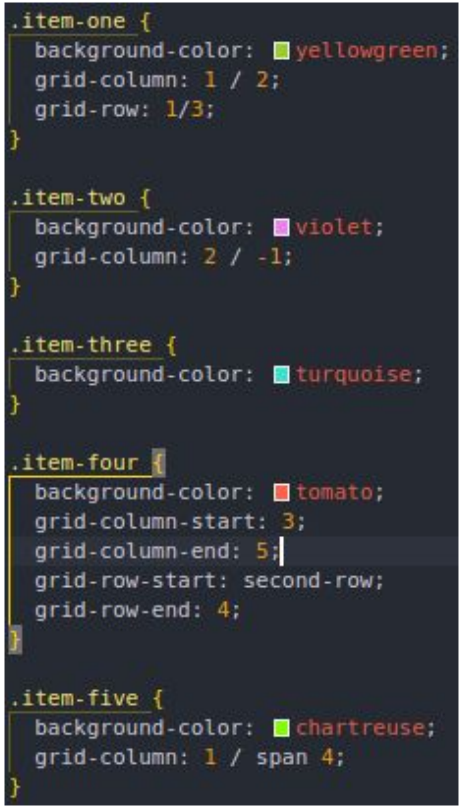
span keyword can be used as well - span <number> or span <name>

keyword auto indicates auto-placement - automatic span or a default span of one

When end is not specified, the item will span 1 track by default

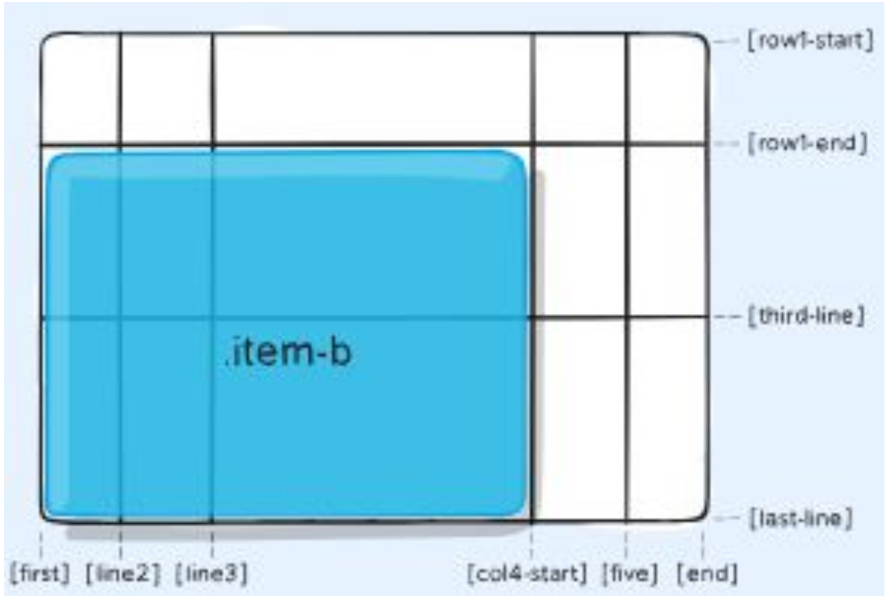
## grid-column-start... example

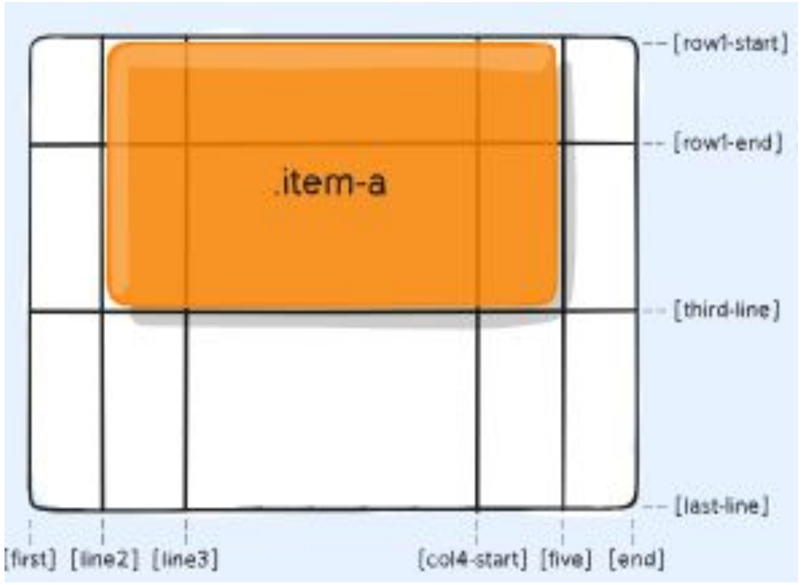




```
.item-one {
  background-color: yellowgreen;
  grid-column: 1 / 2;
  grid-row: 1/3;
}

.item-two {
  background-color: violet;
  grid-column: 2 / -1;
}

.item-three {
  background-color: turquoise;
}

.item-four {
  background-color: tomato;
  grid-column-start: 3;
  grid-column-end: 5;
  grid-row-start: second-row;
  grid-row-end: 4;
}

.item-five {
  background-color: chartreuse;
  grid-column: 1 / span 4;
}
```

## Example of previous slide





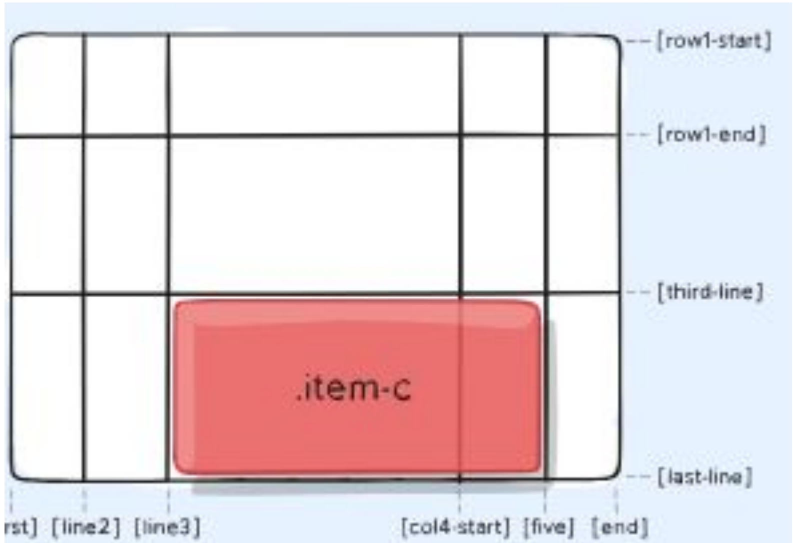
## grid-column, grid-row properties

Shorthand for grid-column-start + grid-column-end

Shorthand for grid-row-start + grid-row-end

Values are the same as for longhand version, including span

<start-line> / <end-line>



## grid-area property

GIves an item a name so that we can reference the item in grid template created with grid-template-areas property

Can be used as a shorthand as well for grid-row-start + grid-column-start + grid-row-end + grid-column-end

<name>

<row-start> / <column-start> / <row-end> / <column-end>

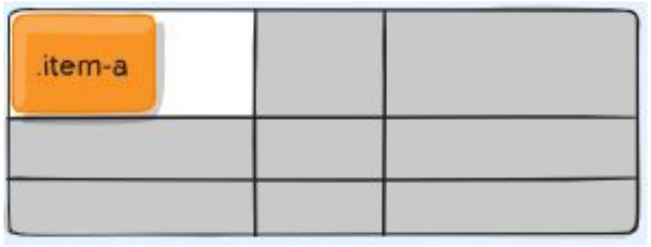
Values for shorthand version can be numbers or named lines

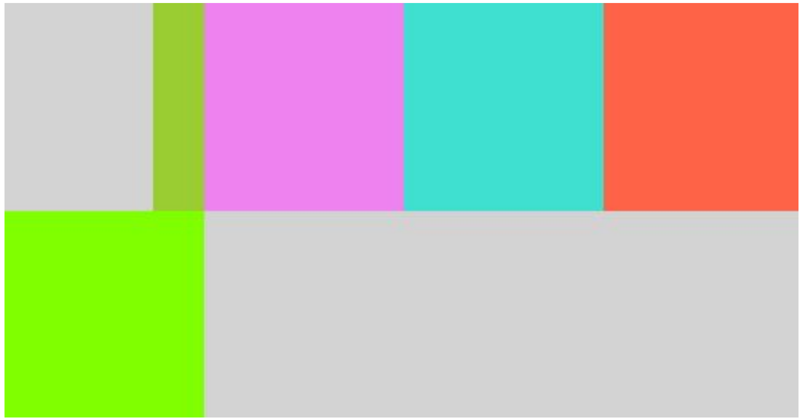
## justify-self property

Aligns a grid item inside a cell along the row asix

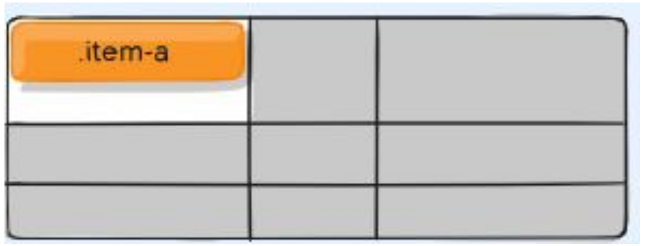
Value applies to a grid item inside a single cell

Possible values: start, end, center, stretch (default value)





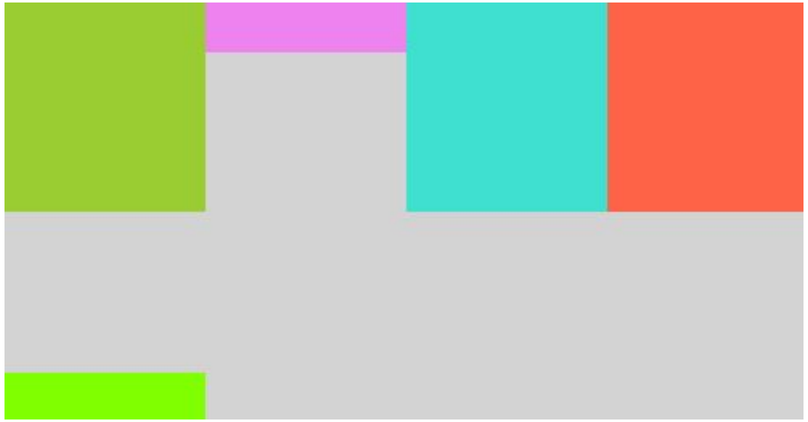
## align-self property



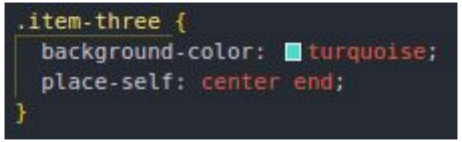
Aligns a grid item inside a cell along the column axis

Value applies to the content inside a single grid item

Possible values: start, end, center, stretch (default value)



## place-self property

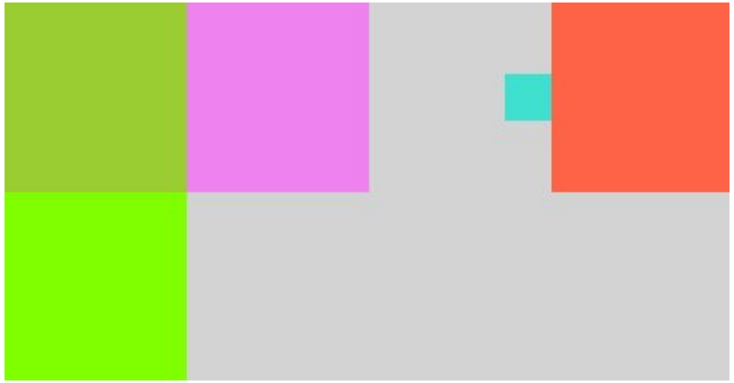


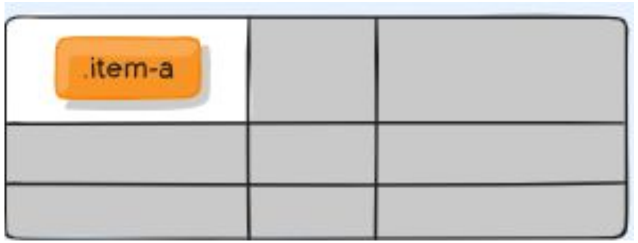
```
.item-three {
  background-color: turquoise;
  place-self: center end;
}
```

Can be used to set both align-self and justify-self properties in a single declaration

<align-self>/<justify-self>

If the second value is not declared, the first value is assigned to both properties





## sizing keywords and sizing functions

fr unit - portion of the remaining space

All lengths values can be used - px, rem, %, em etc.

min-content - minimum size of the content - E pluribus unum - width of pluribus

max-content - maximum size of the content - length of the whole sentence

auto - similar to fr unit but “loose” against fr unit when used together

fit-content() - never less than min-content and never more than max-content

minmax() - sets a minimum and maximum value of the length for the item

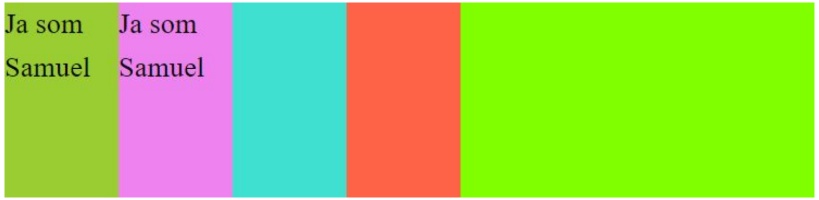
repeat() - reduces typing, repeats the column by specification

## sizing keywords example

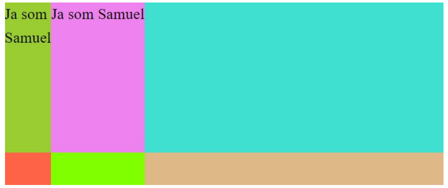
```css
grid-template-columns: 1fr 1fr 1fr 1fr;
grid-template-columns: repeat(4, 1fr);
```

```css
grid-template-columns: repeat(5, minmax(100px, 1fr));
```

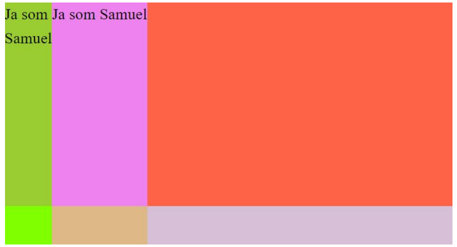
```css
grid-template-columns: repeat(4, 1fr) minmax(200px, 1fr);
```



```css
grid-template-columns: min-content max-content auto;
```



```css
grid-template-columns: min-content max-content auto 1fr;
```

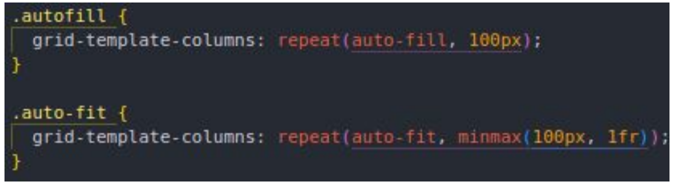


## auto-fill, auto-fit properties

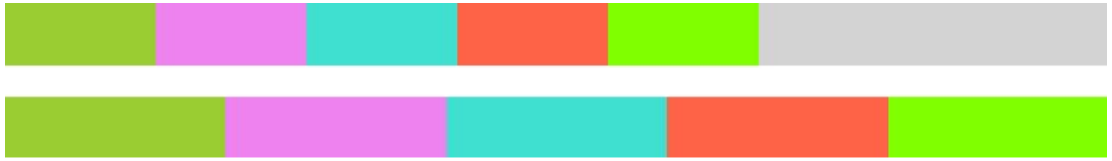
auto-fill - fit as many possible columns as possible on a row even when they are empty

auto-fit - fit whatever columns there are into the space, when there is less items than the width of container, these items will fill space, no empty columns

auto-fit should be used with minmax



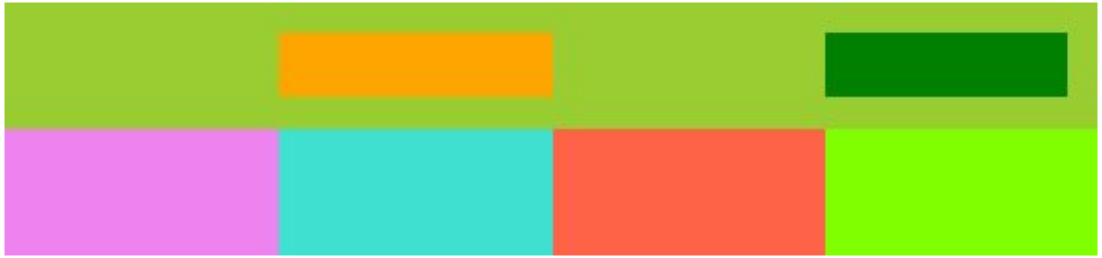
```
.autofill {
  grid-template-columns: repeat(auto-fill, 100px);
}

.auto-fit {
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
```

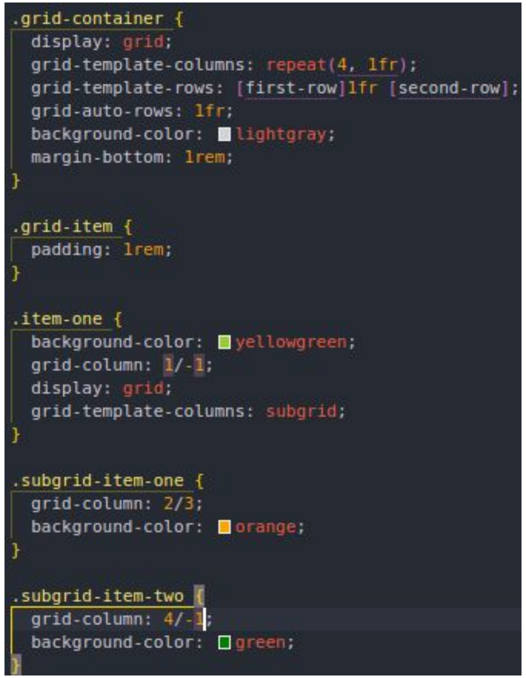


# subgrid

Allows grid items to have a grid of their own that inherits lines from the parent grid

grid-template-columns: subgrid





```css
.grid-container {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-template-rows: [first-row]1fr [second-row];
  grid-auto-rows: 1fr;
  background-color: lightgray;
  margin-bottom: 1rem;
}

.grid-item {
  padding: 1rem;
}

.item-one {
  background-color: yellowgreen;
  grid-column: 1/-1;
  display: grid;
  grid-template-columns: subgrid;
}

.subgrid-item-one {
  grid-column: 2/3;
  background-color: orange;
}

.subgrid-item-two {
  grid-column: 4/-1;
  background-color: green;
}
```

## When we should use grid layout?

Two-dimensional layout

Complex layout

Need to overlap elements

Layout-first design

## Should I use flexbox or grid?

There is no right answer

Flexbox and Grid serve different purposes and are both useful

The best results are when they are used together

Important thing is to decide, which one should be use - by practising things will get better

Rule of thumb - use grid for full page layouts and flex for everything else  - but again, it depends, sometimes the grid option will be better even for non page layout

## Questions?

## Resources

[Web layout history: How we got to grid and flex]

[Auto-fit Vs Auto-fill]

[A Complete Guide to CSS Grid]

[A Complete Guide to Flexbox | CSS-Tricks]

[Flexbox History]