alchemy code lab

developing the future

CSS layout techniques

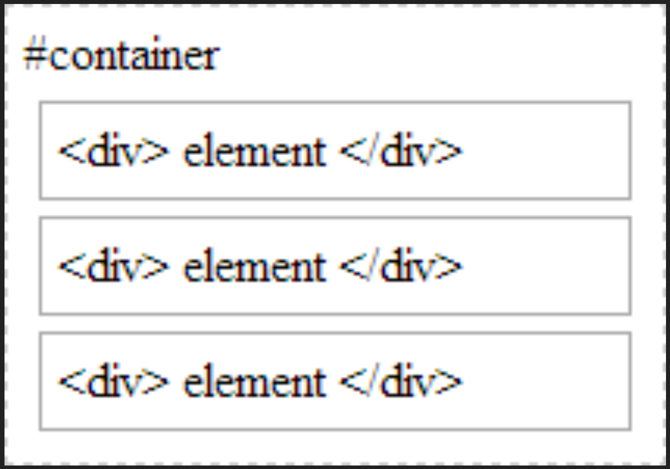
alchemy code lab

developing the future

Part one: CSS positioning

position: static

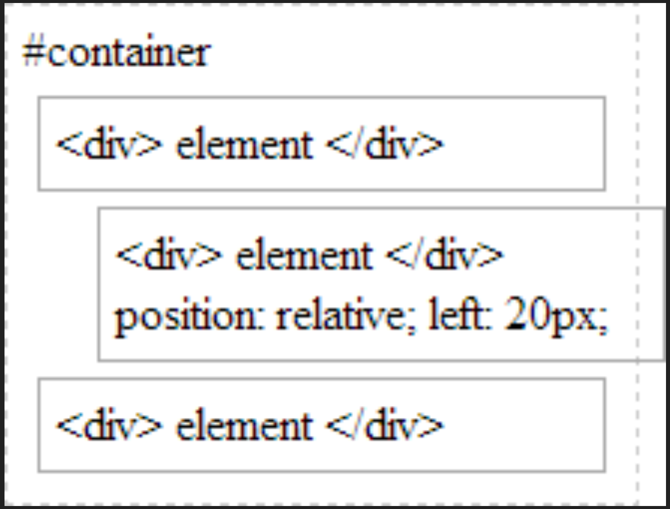
The element is positioned according normal document flow. The `top`, `right`, `bottom`, `left`, and `z-index` properties have no effect. This is the default value.



http://bit.ly/2iWiXqQ

position: relative

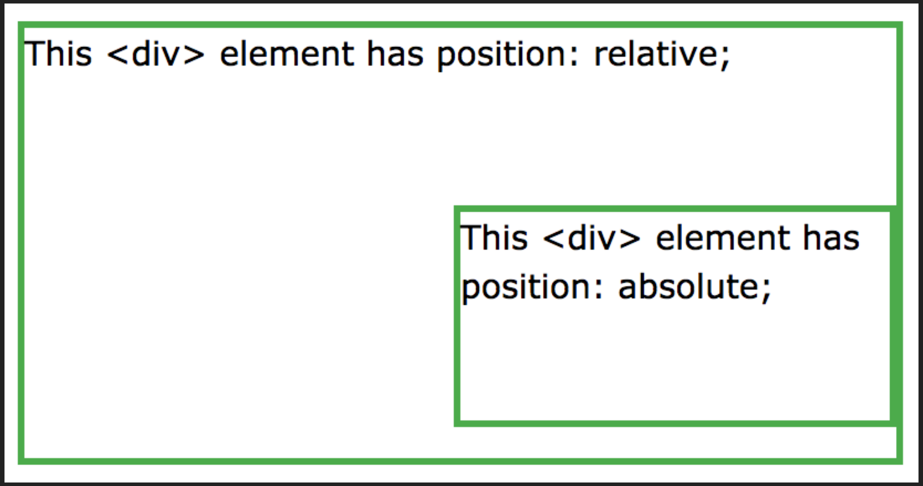
The element is positioned according to the normal flow of the document, and then offset *relative to itself* based on the values of `top`, `right`, `bottom`, and `left`.



http://bit.ly/2iWiXqQ

position: absolute

The element is positioned relative to the nearest positioned ancestor; or, if it has none, it uses the document body, and moves along with page scrolling.


This <div> element has position: relative;
This <div> element has position: absolute;

w3schools.com

position: fixed

The element is positioned relative to the viewport and doesn't move when scrolled. Its position is determined by the values of `top`, `right`, `bottom`, and `left`.

position: sticky

Sticky positioning is a hybrid of relative and fixed positioning. The element is treated as relative positioned until it crosses a specified threshold, at which point it is treated as fixed positioned.

position: sticky

Sticky positioning is a hybrid of relative and fixed positioning. The element is treated as relative positioned until it crosses a specified threshold, at which point it is treated as fixed positioned.

(supported by about 75% of modern browsers.
See https://caniuse.com/#feat=css-sticky)

alchemy code lab
developing the future

Part two: CSS display types

display: none

The element, and all its descendants, will not display.

display: none

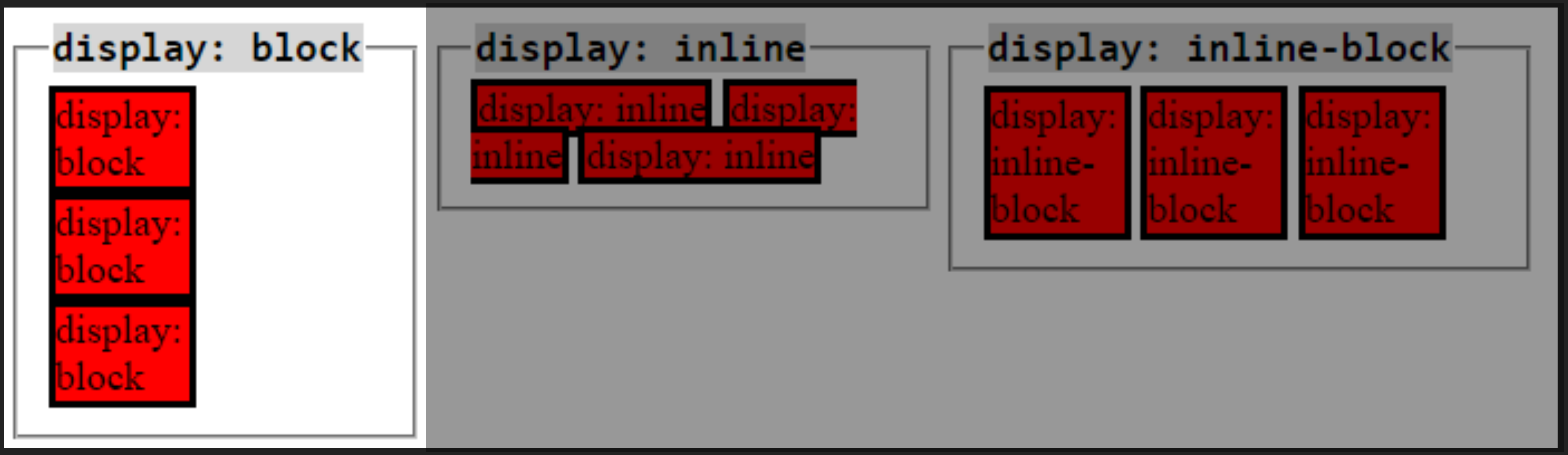
The element, and all its descendants, will not display.

Do not use this if you want content to be read by a screen reader, as it will not render to the DOM!

Alternate methods are found at https://webaim.org/techniques/css/invisiblecontent/

display: block

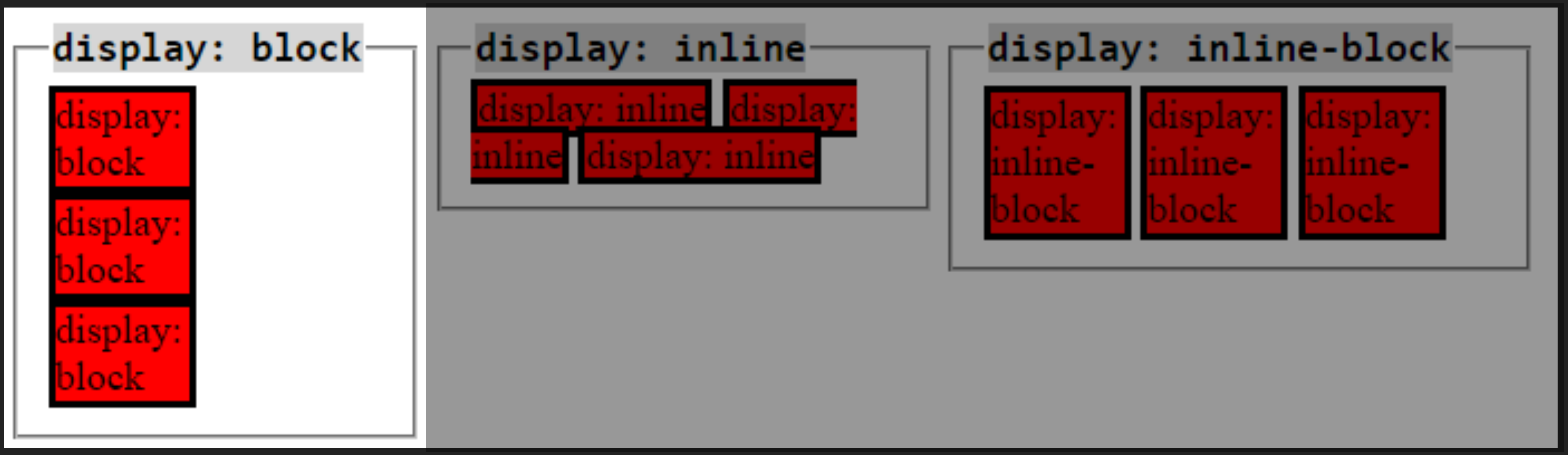
The element will be displayed as a block.
This is the default setting for `p`, `h1`, and `div`.



http://dustwell.com

display: block

The element will be displayed as a block.
This is the default setting for `p`, `h1`, and `div`.

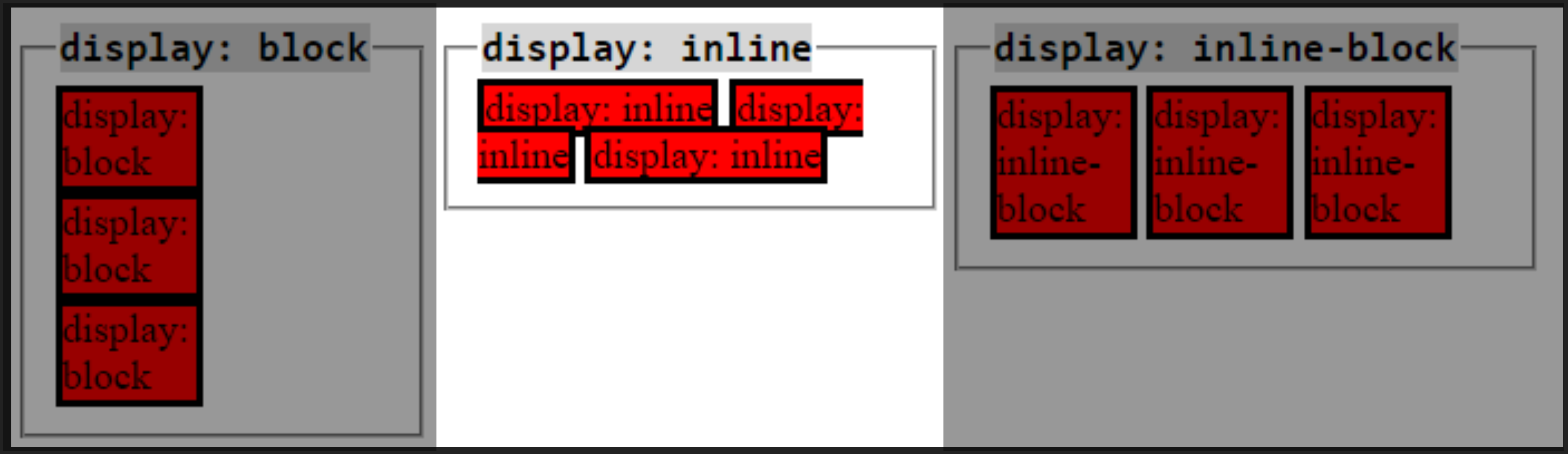


http://dustwell.com

A block tolerates no HTML elements next to it, except when ordered otherwise (using float, for instance.)

display: inline

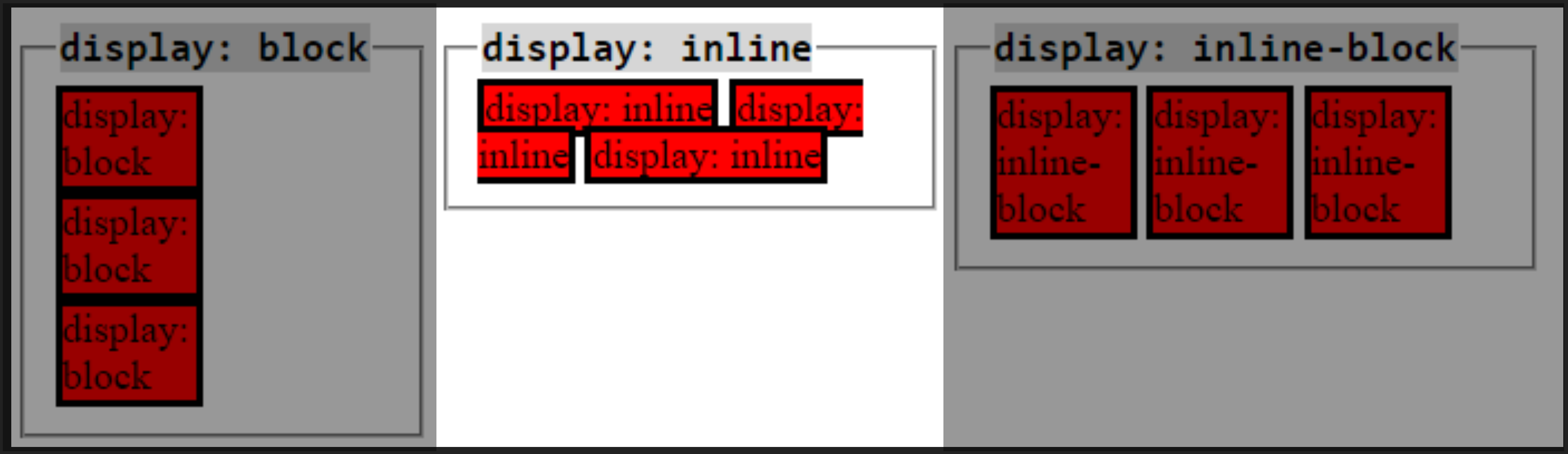
The element will be displayed inline.
This is the default setting for `a` and `span`.



http://dustwell.com

display: inline

The element will be displayed inline.
This is the default setting for `a` and `span`.

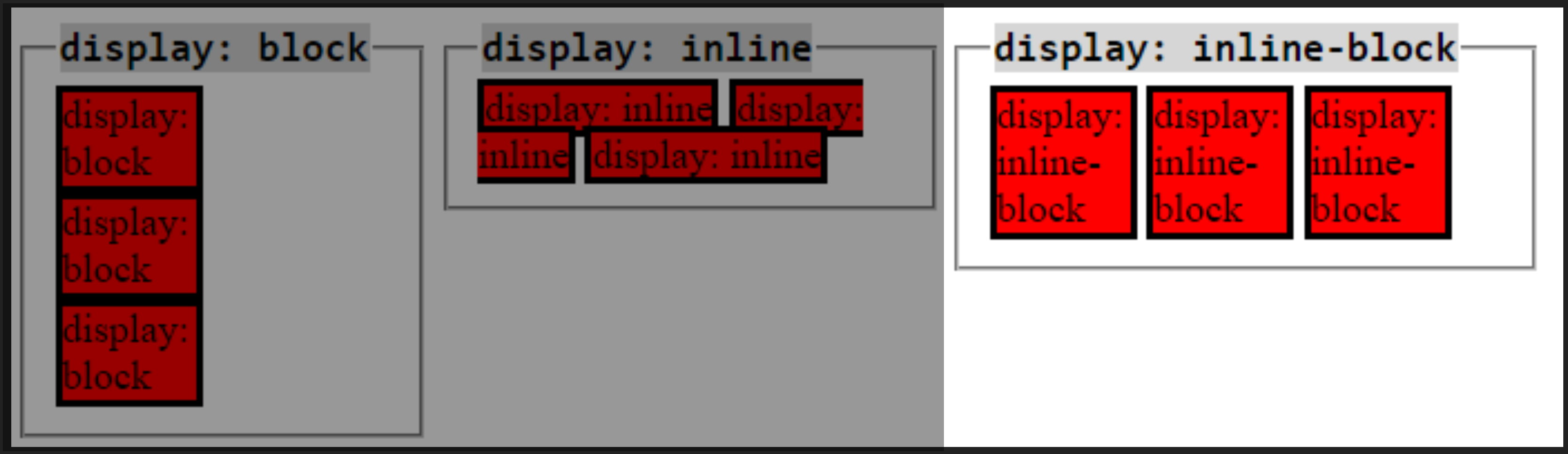


http://dustwell.com

An inline element has no line break before or after it, and tolerates HTML elements next to it.

display: inline-block

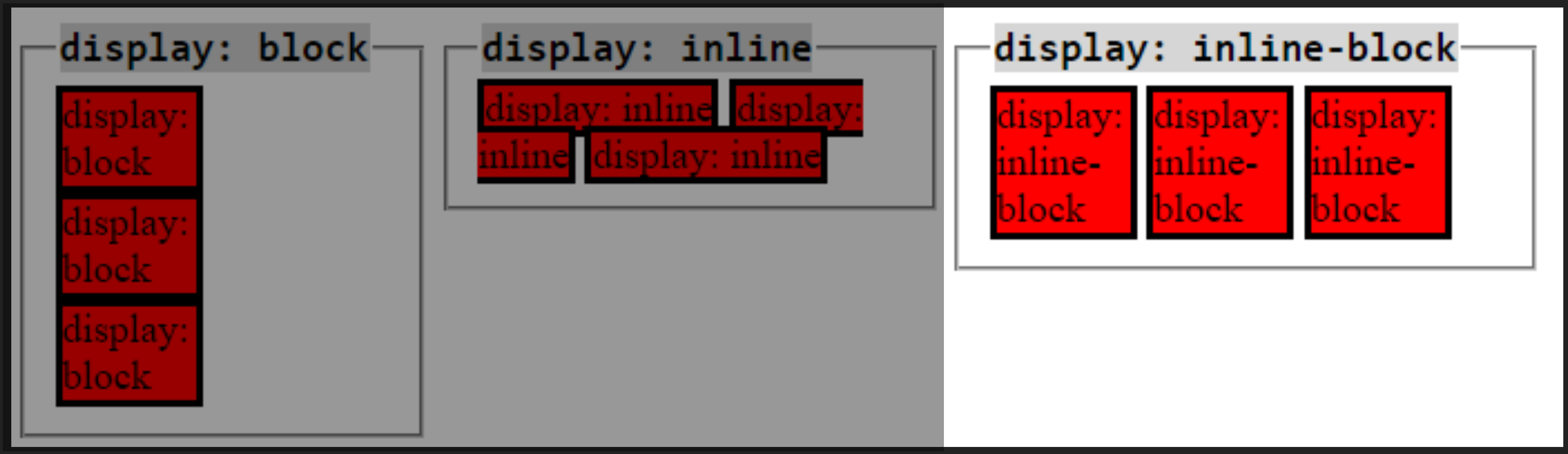
Very few HTML elements are inline-block by default, although `button` and `select` are.



http://dustwell.com

display: inline-block

Very few HTML elements are inline-block by default, although `button` and `select` are.



http://dustwell.com

The element is formatted as block-level (meaning width, height, padding can be applied), but it is placed next to adjacent content like an inline element.

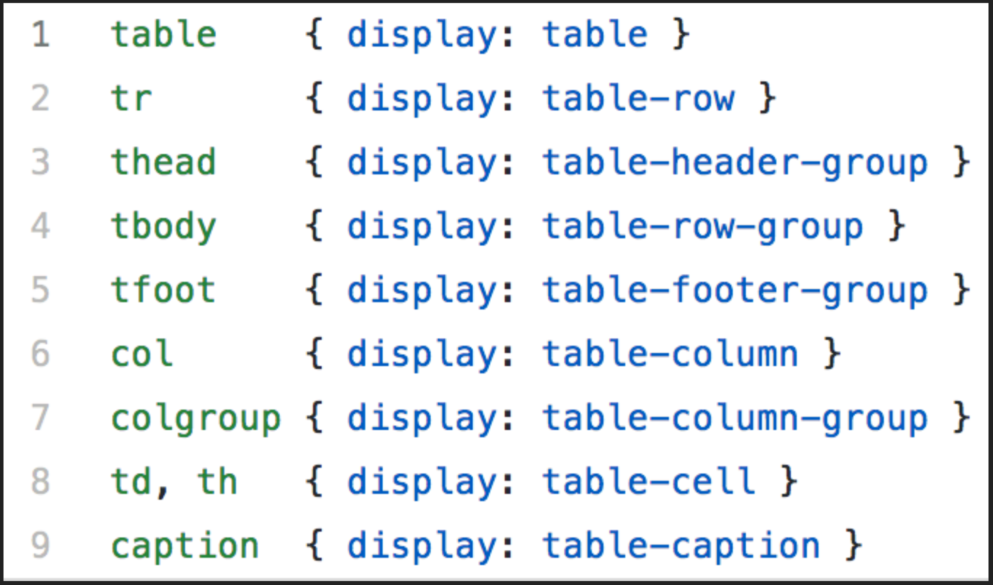
display: table

It’s well known that `<table>` and its related tags are used only to format tabular data, not to create layouts.

display: table

It's well known that `<table>` and its related tags are used only to format tabular data, not to create layouts.

However, these display types exist to force non-table elements to behave like table cells.



```css
table    { display: table }
tr       { display: table-row }
thead    { display: table-header-group }
tbody    { display: table-row-group }
tfoot    { display: table-footer-group }
col      { display: table-column }
colgroup { display: table-column-group }
td, th   { display: table-cell }
caption  { display: table-caption }
```

colintoh.com

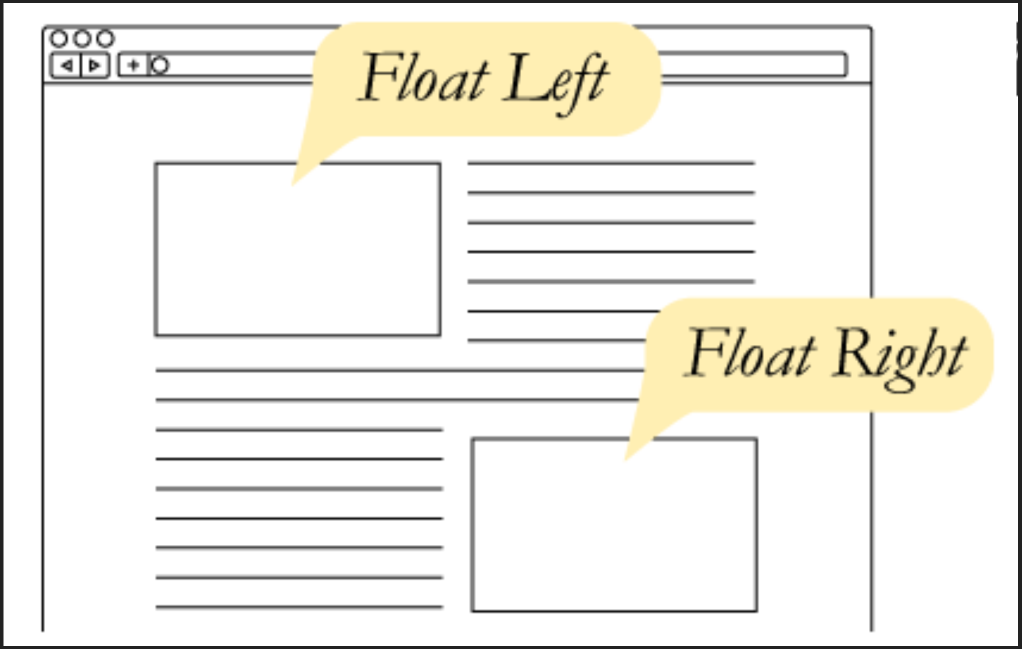
alchemy code lab

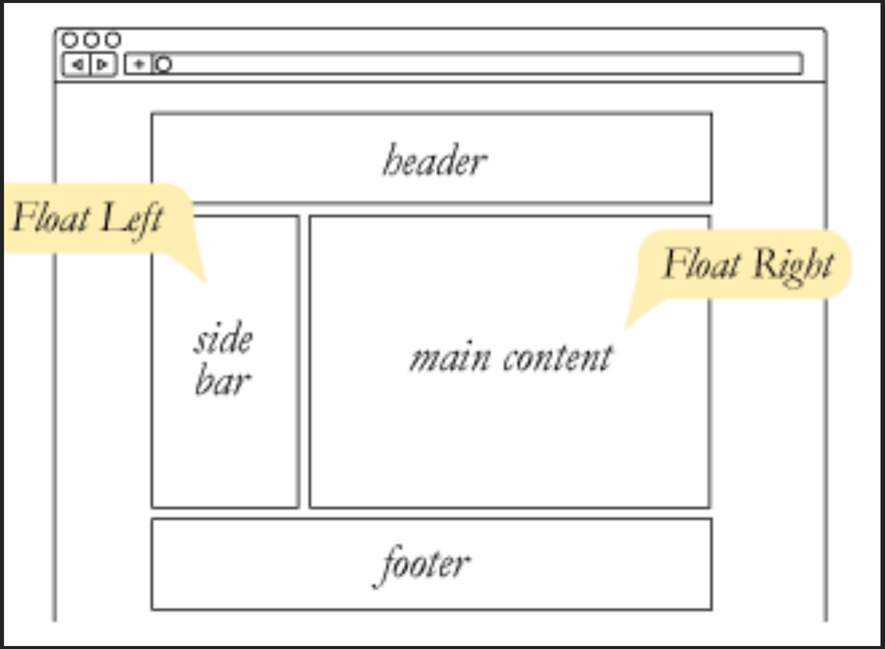
developing the future

Part three: Float and clear

float

A float is a box that is shifted to the left or right on the current line. Content flows down the right side of a left-floated box and down the left side of a right-floated box.





csstricks.com

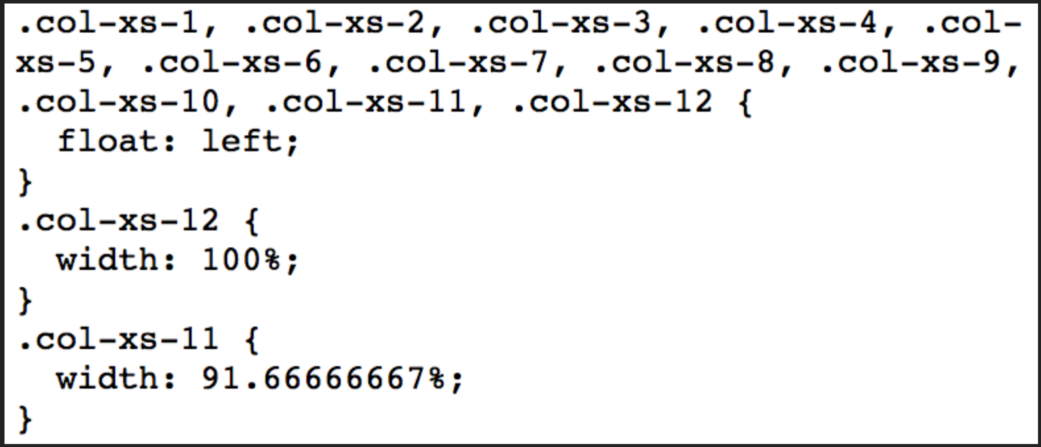
float

Floats are commonly used to create multiple column layouts due to their widespread browser support.

float

Floats are commonly used to create multiple column layouts due to their widespread browser support.

For example, popular front-end framework Bootstrap built its CSS grid using floats.

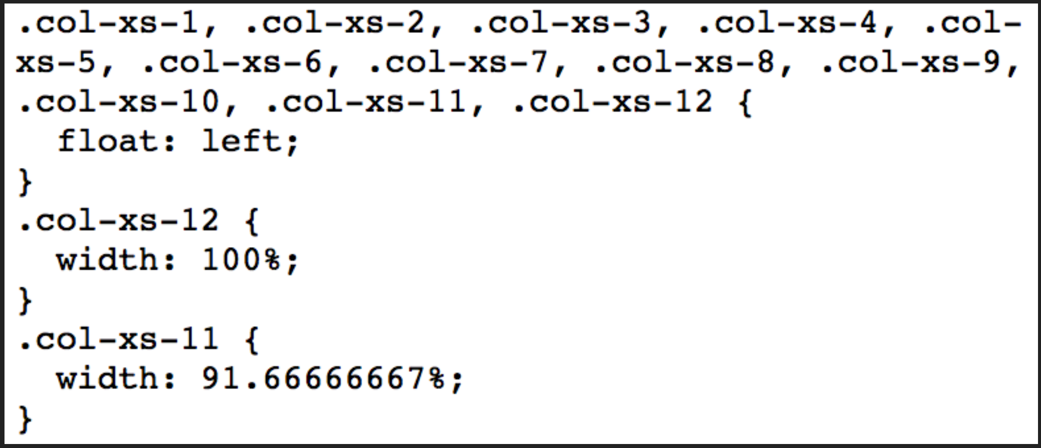


```
.col-xs-1, .col-xs-2, .col-xs-3, .col-xs-4, .col-
xs-5, .col-xs-6, .col-xs-7, .col-xs-8, .col-xs-9,
.col-xs-10, .col-xs-11, .col-xs-12 {
  float: left;
}
.col-xs-12 {
  width: 100%;
}
.col-xs-11 {
  width: 91.66666667%;
}
```

float

Floats are commonly used to create multiple column layouts due to their widespread browser support.

For example, popular front-end framework Bootstrap built its CSS grid using floats.



```
.col-xs-1, .col-xs-2, .col-xs-3, .col-xs-4, .col-
xs-5, .col-xs-6, .col-xs-7, .col-xs-8, .col-xs-9,
.col-xs-10, .col-xs-11, .col-xs-12 {
  float: left;
}
.col-xs-12 {
  width: 100%;
}
.col-xs-11 {
  width: 91.66666667%;
}
```

This is despite the fact that floats were not really intended for this job, and have issues that have to be dealt with.

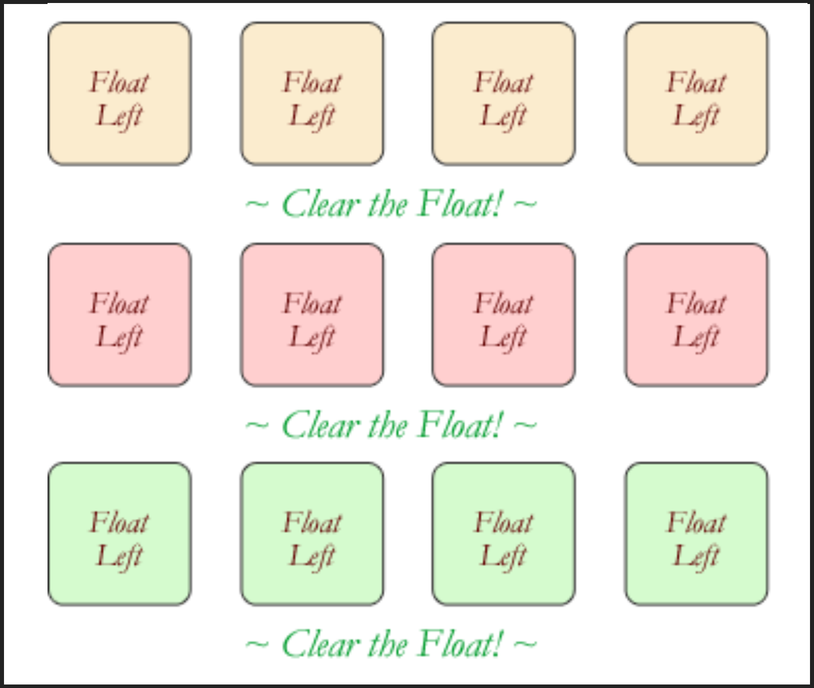
clear

One problem is any content below the floats (that isn’t floated itself) will wrap around the floated elements.

clear

One problem is any content below the floats (that isn’t floated itself) will wrap around the floated elements.

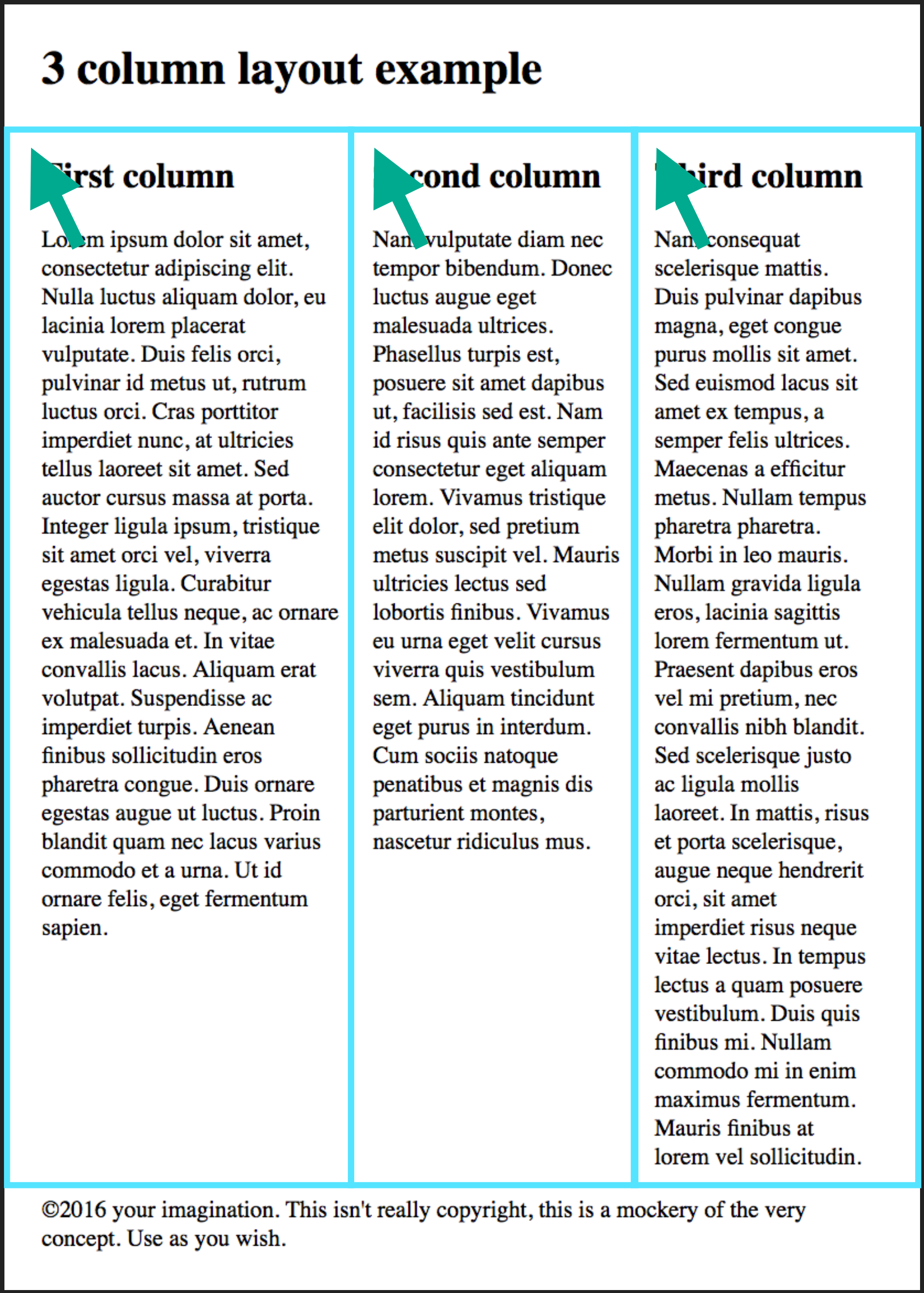
The solution is clear, which specifies that this element and those after it will not wrap around the floated elements.



csstricks.com

clear

In this example, the three column divs are set to `float: left`



3 column layout example

First column

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus aliquam dolor, eu lacinia lorem placerat vulputate. Duis felis orci, pulvinar id metus ut, rutrum luctus orci. Cras porttitor imperdiet nunc, at ultricies tellus laoreet sit amet. Sed auctor cursus massa at porta. Integer ligula ipsum, tristique sit amet orci vel, viverra egestas ligula. Curabitur vehicula tellus neque, ac ornare ex malesuada et. In vitae convallis lacus. Aliquam erat volutpat. Suspendisse ac imperdiet turpis. Aenean finibus sollicitudin eros pharetra congue. Duis ornare egestas augue ut luctus. Proin blandit quam nec lacus varius commodo et a urna. Ut id ornare felis, eget fermentum sapien.

Second column

Nam vulputate diam nec tempor bibendum. Donec luctus augue eget malesuada ultrices. Phasellus turpis est, posuere sit amet dapibus ut, facilisis sed est. Nam id risus quis ante semper consectetur eget aliquam lorem. Vivamus tristique elit dolor, sed pretium metus suscipit vel. Mauris ultricies lectus sed lobortis finibus. Vivamus eu urna eget velit cursus viverra quis vestibulum sem. Aliquam tincidunt eget purus in interdum. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Third column

Nam consequat scelerisque mattis. Duis pulvinar dapibus magna, eget congue purus mollis sit amet. Sed euismod lacus sit amet ex tempus, a semper felis ultrices. Maecenas a efficitur metus. Nullam tempus pharetra pharetra. Morbi in leo mauris. Nullam gravida ligula eros, lacinia sagittis lorem fermentum ut. Praesent dapibus eros vel mi pretium, nec convallis nibh blandit. Sed scelerisque justo ac ligula mollis laoreet. In mattis, risus et porta scelerisque, augue neque hendrerit orci, sit amet imperdiet risus neque vitae lectus. In tempus lectus a quam posuere vestibulum. Duis quis finibus mi. Nullam commodo mi in enim maximus fermentum. Mauris finibus at lorem vel sollicitudin.

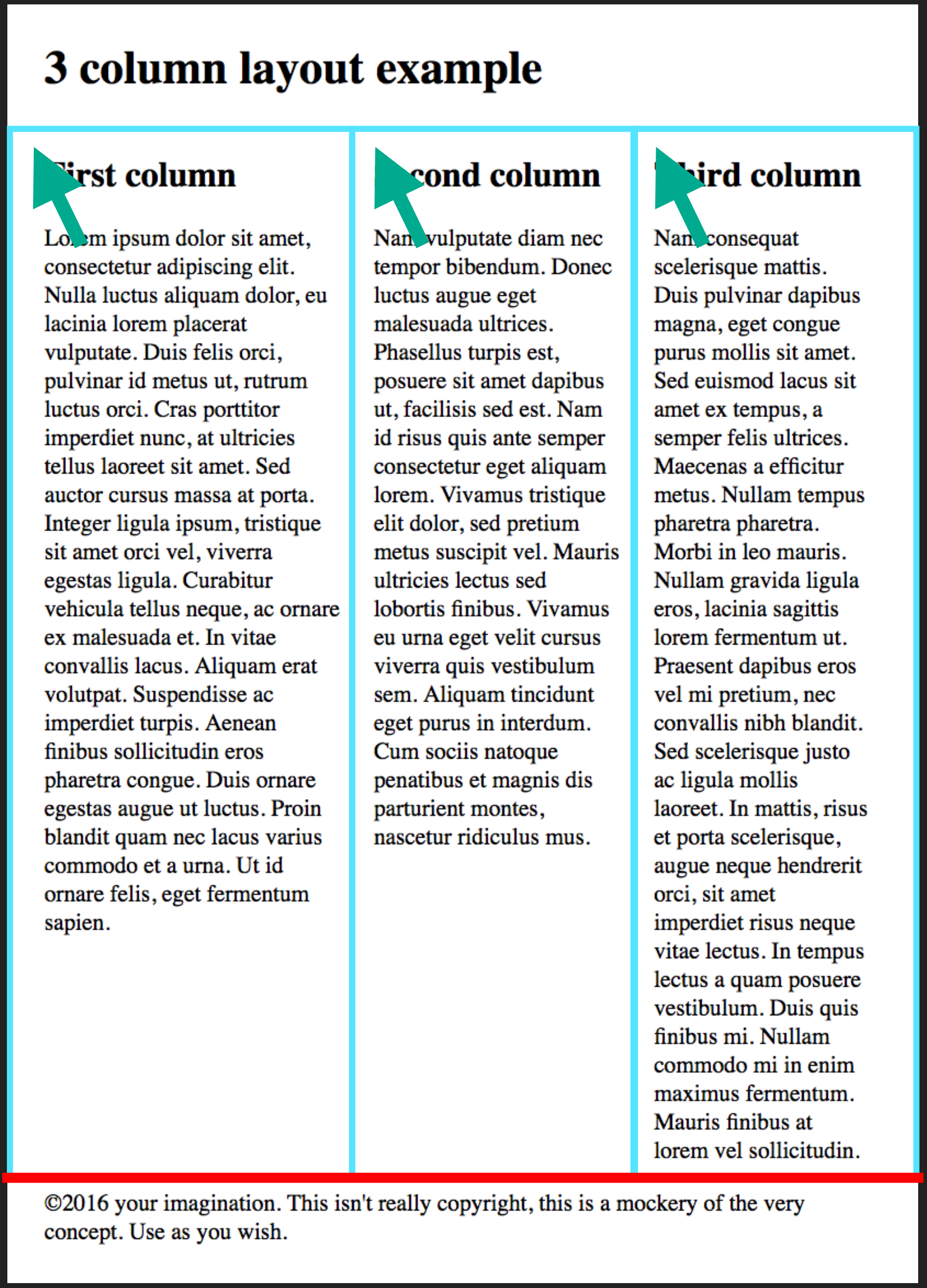
©2016 your imagination. This isn't really copyright, this is a mockery of the very concept. Use as you wish.

developer.mozilla.org

clear

In this example, the three column divs are set to `float: left`

while the footer is set to `clear: both`



3 column layout example

irst column

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus aliquam dolor, eu lacinia lorem placerat vulputate. Duis felis orci, pulvinar id metus ut, rutrum luctus orci. Cras porttitor imperdiet nunc, at ultricies tellus laoreet sit amet. Sed auctor cursus massa at porta. Integer ligula ipsum, tristique sit amet orci vel, viverra egestas ligula. Curabitur vehicula tellus neque, ac ornare ex malesuada et. In vitae convallis lacus. Aliquam erat volutpat. Suspendisse ac imperdiet turpis. Aenean finibus sollicitudin eros pharetra congue. Duis ornare egestas augue ut luctus. Proin blandit quam nec lacus varius commodo et a urna. Ut id ornare felis, eget fermentum sapien.

cond column

Nam vulputate diam nec tempor bibendum. Donec luctus augue eget malesuada ultrices. Phasellus turpis est, posuere sit amet dapibus ut, facilisis sed est. Nam id risus quis ante semper consectetur eget aliquam lorem. Vivamus tristique elit dolor, sed pretium metus suscipit vel. Mauris ultricies lectus sed lobortis finibus. Vivamus eu urna eget velit cursus viverra quis vestibulum sem. Aliquam tincidunt eget purus in interdum. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

hird column

Nam consequat scelerisque mattis. Duis pulvinar dapibus magna, eget congue purus mollis sit amet. Sed euismod lacus sit amet ex tempus, a semper felis ultrices. Maecenas a efficitur metus. Nullam tempus pharetra pharetra. Morbi in leo mauris. Nullam gravida ligula eros, lacinia sagittis lorem fermentum ut. Praesent dapibus eros vel mi pretium, nec convallis nibh blandit. Sed scelerisque justo ac ligula mollis laoreet. In mattis, risus et porta scelerisque, augue neque hendrerit orci, sit amet imperdiet risus neque vitae lectus. In tempus lectus a quam posuere vestibulum. Duis quis finibus mi. Nullam commodo mi in enim maximus fermentum. Mauris finibus at lorem vel sollicitudin.

©2016 your imagination. This isn't really copyright, this is a mockery of the very concept. Use as you wish.

developer.mozilla.org

alchemy code lab
developing the future

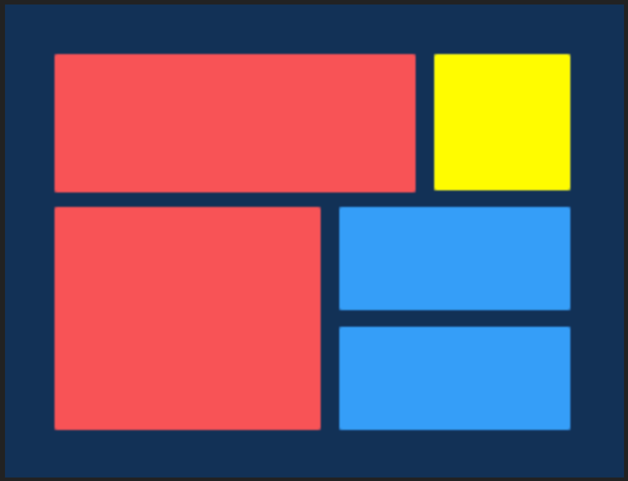
Part four: Flexbox

Flexbox and CSS grid

While `float` and even `display: table` can be used to achieve responsive layouts, neither was designed for that purpose.

Flexbox and CSS grid

While `float` and even `display: table` can be used to achieve responsive layouts, neither was designed for that purpose.



tutorialzine.com

CSS offers two layout modes specific to complex, responsive layouts: flexbox and CSS grid.

Flexbox and CSS grid

CSS Grid is the newer of the two methods. Chrome and Firefox support traces back only to March 2017. See https://caniuse.com/#feat=css-grid.

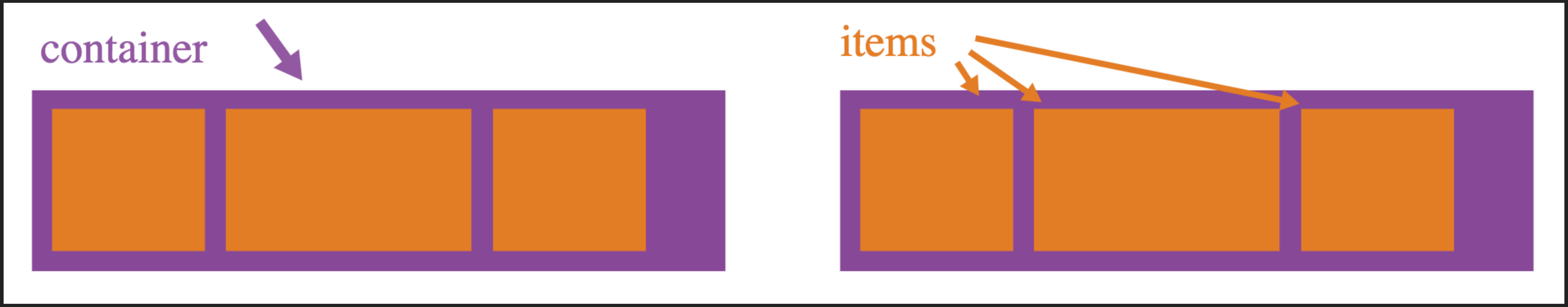
Flexbox and CSS grid

CSS Grid is the newer of the two methods. Chrome and Firefox support traces back only to March 2017. See https://caniuse.com/#feat=css-grid.

Flexbox and grid can be combined in the same layout, and we'll discuss them both, but let's start with the more universally browser-supported flexbox.

display: flex

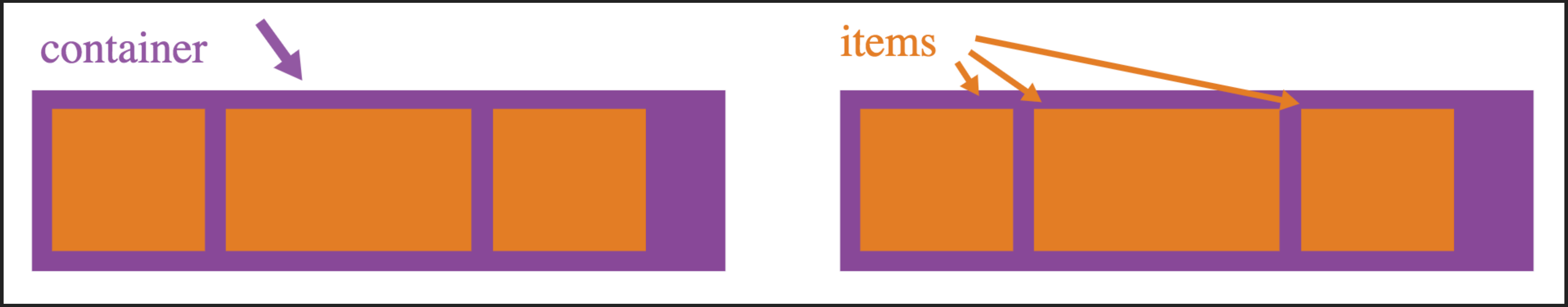
This defines a flex container and enables flex context for all the container's direct children.



css-tricks.com

display: flex

This defines a flex container and enables flex context for all the container’s direct children.



css-tricks.com

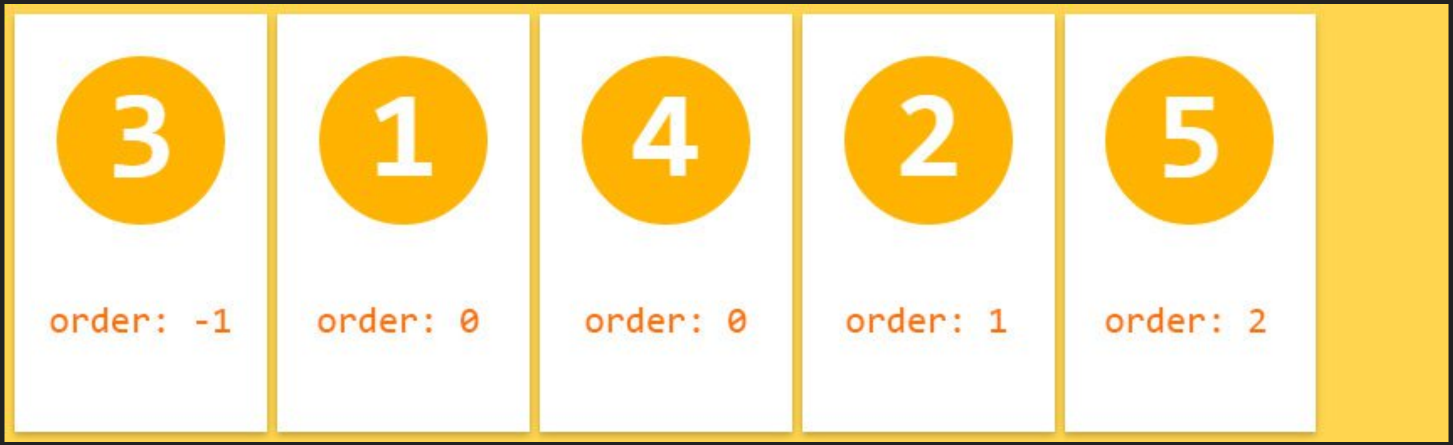
(Your container can also be set to `inline-flex` to act like a flexible inline element, but this is not common.)

order

By default, flex items are laid out in the source order. However, the `order` property controls the order of appearance in the flex container.

order

By default, flex items are laid out in the source order. However, the `order` property controls the order of appearance in the flex container.

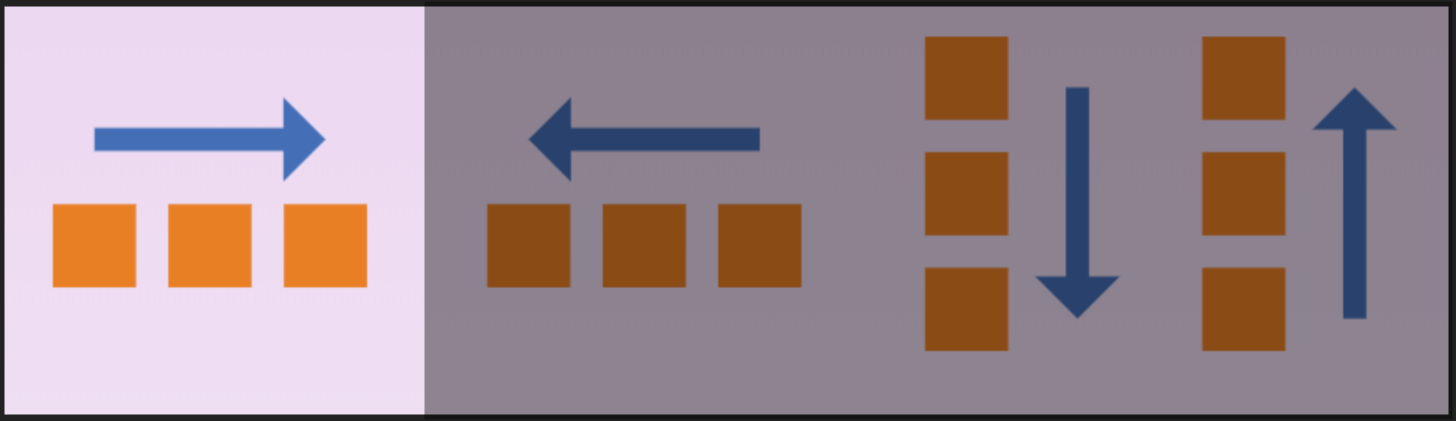


scotch.io

This allows the developer to keep HTML content in proper semantic order while rearranging the display.

flex-direction

Defines the direction flex items are placed in the flex container.

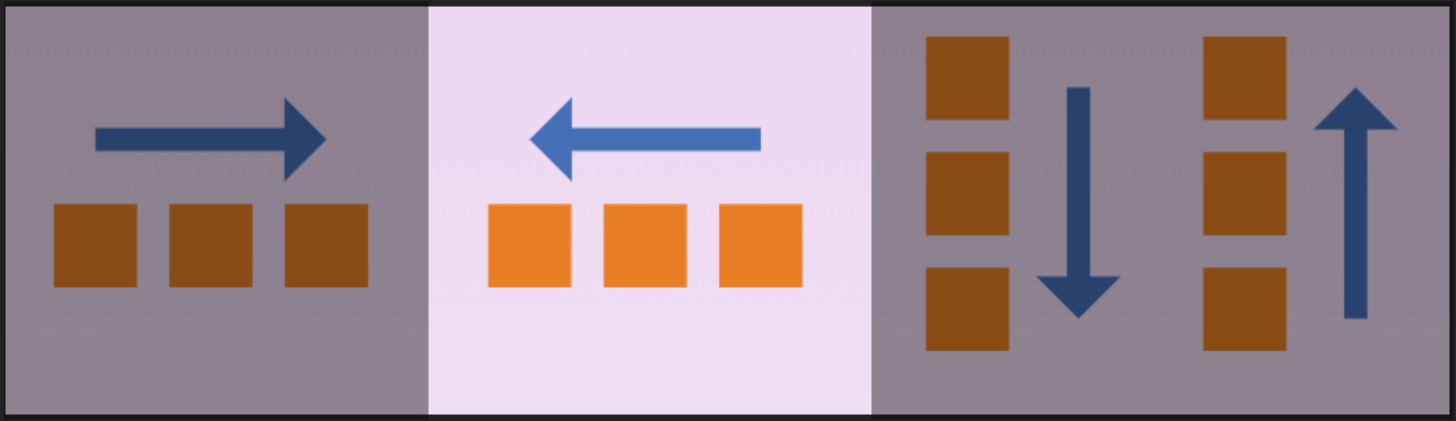


css-tricks.com

`flex-direction: row` displays flex items as a horizontal row. This is the default behavior.

flex-direction

Defines the direction flex items are placed in the flex container.

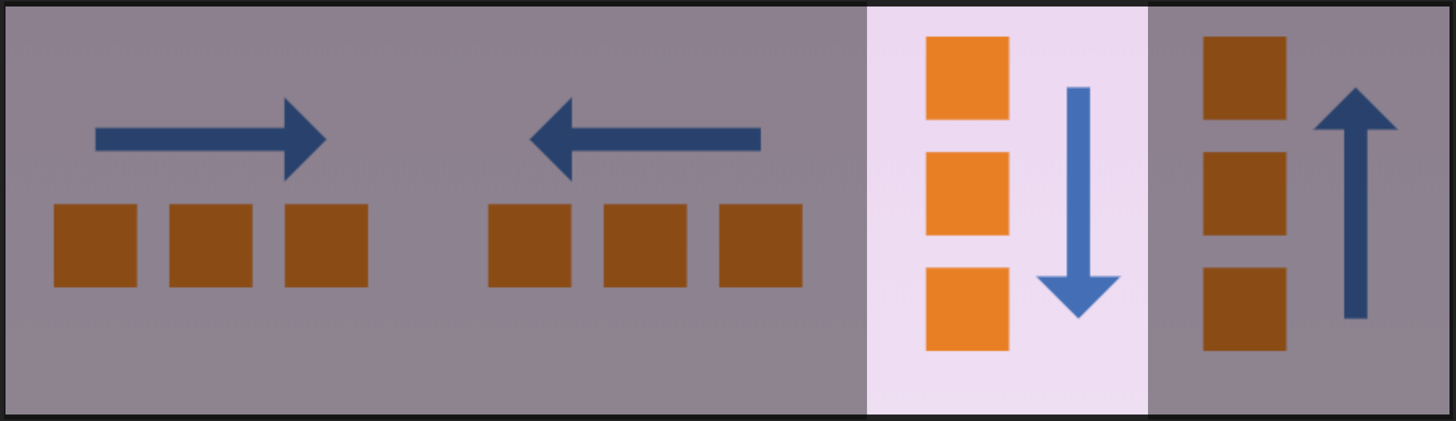


css-tricks.com

`flex-direction: row-reverse` reverses the horizontal order.

flex-direction

Defines the direction flex items are placed in the flex container.

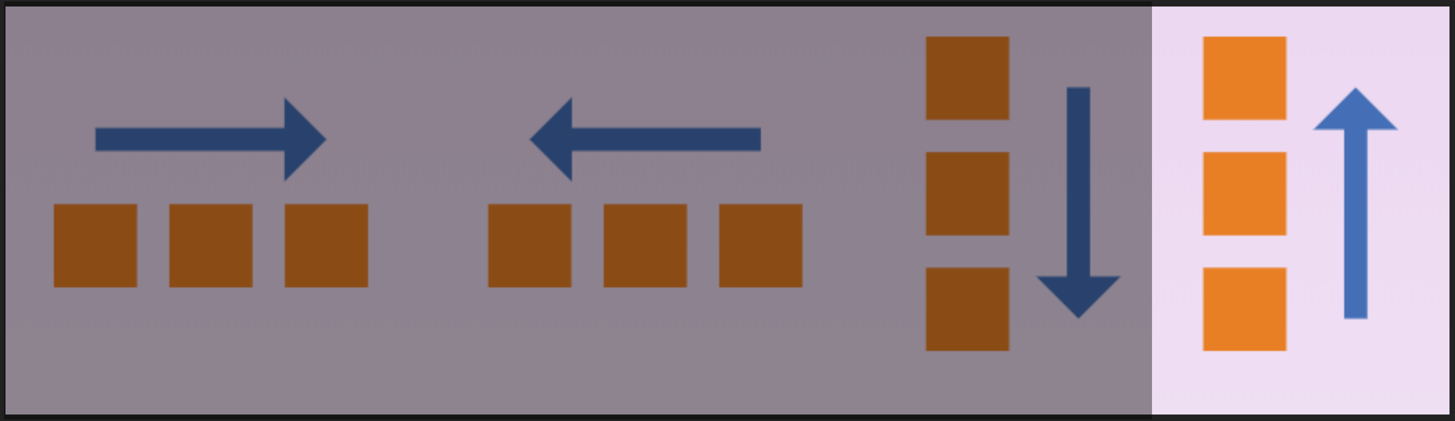


css-tricks.com

`flex-direction: column` displays flex items as a vertical column.

flex-direction

Defines the direction flex items are placed in the flex container.



css-tricks.com

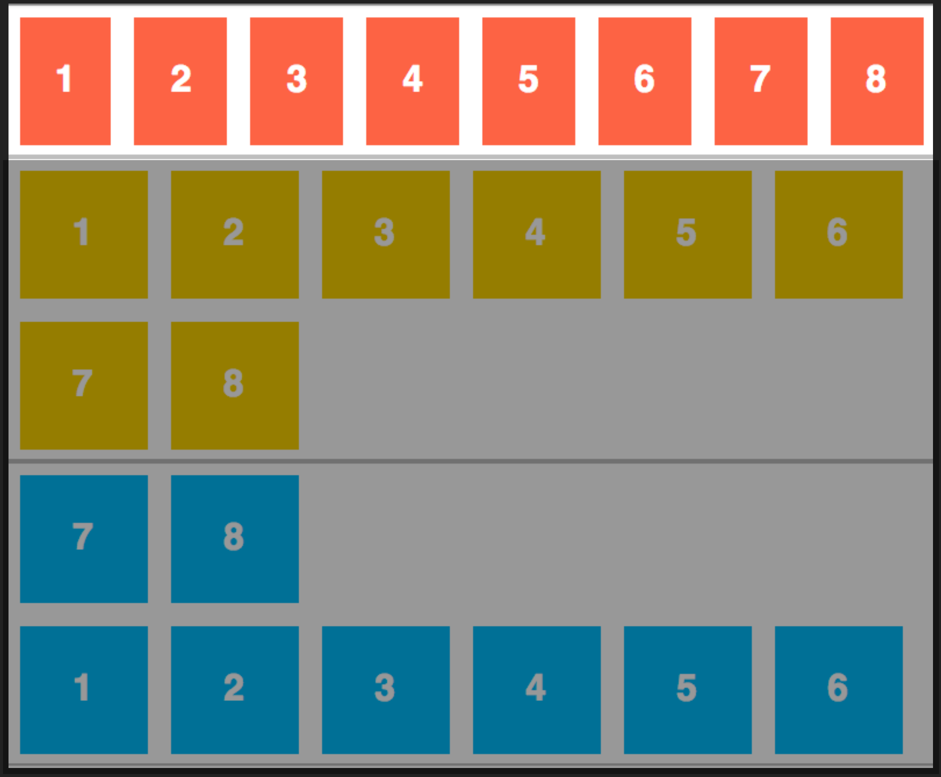
`flex-direction: column-reverse` reverses the horizontal order.

flex-wrap

defines whether the flex items are forced into a single line or can be flowed into multiple lines.

`nowrap`: content forced into a single line.

This is the default behavior.



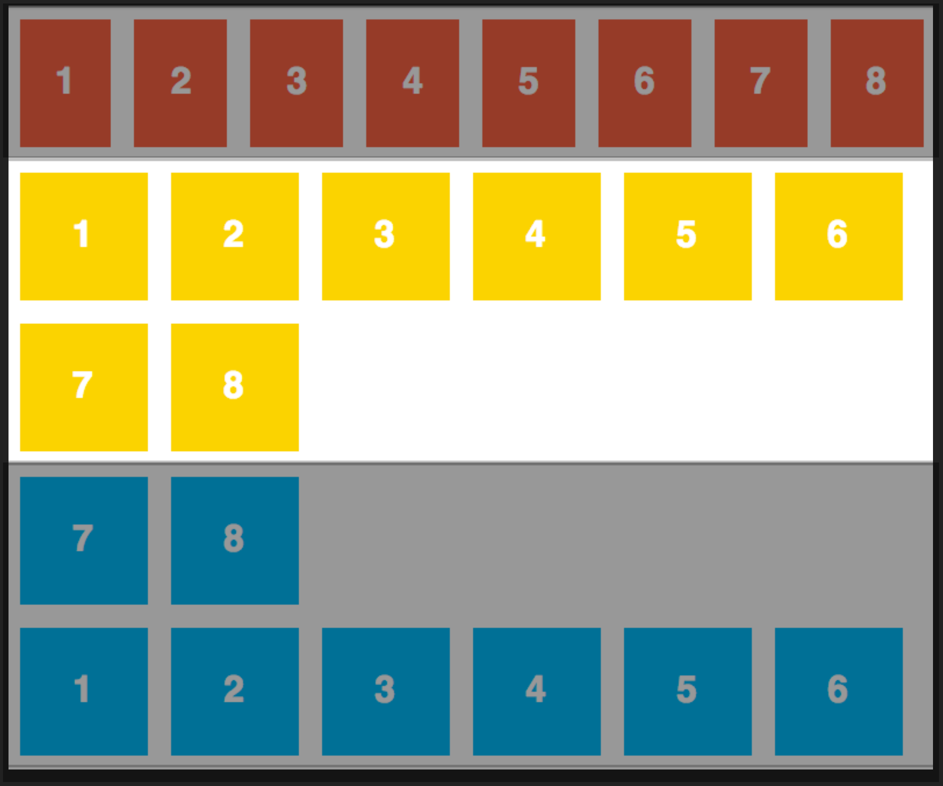
css-tricks.com

flex-wrap

defines whether the flex items are forced into a single line or can be flowed into multiple lines.

`wrap`: content flows onto multiple lines.

Direction is defined by `flex-direction`.



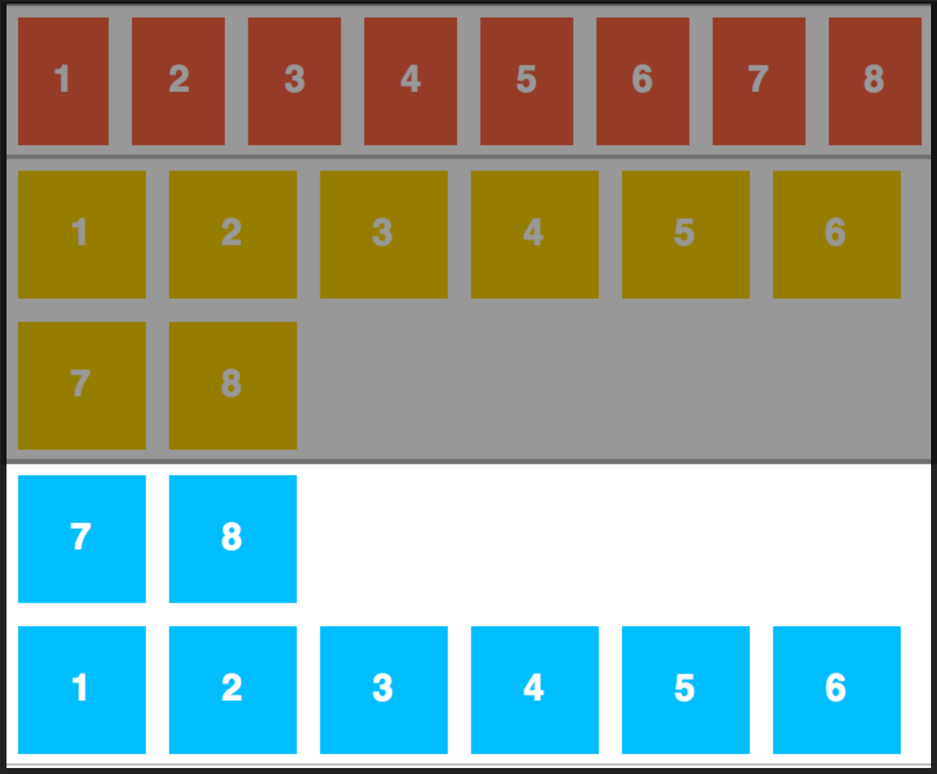
css-tricks.com

flex-wrap

defines whether the flex items are forced into a single line or can be flowed into multiple lines.

`wrap-reverse`: content flows onto multiple lines.

Direction is the **opposite** of what is defined by `flex-direction`.



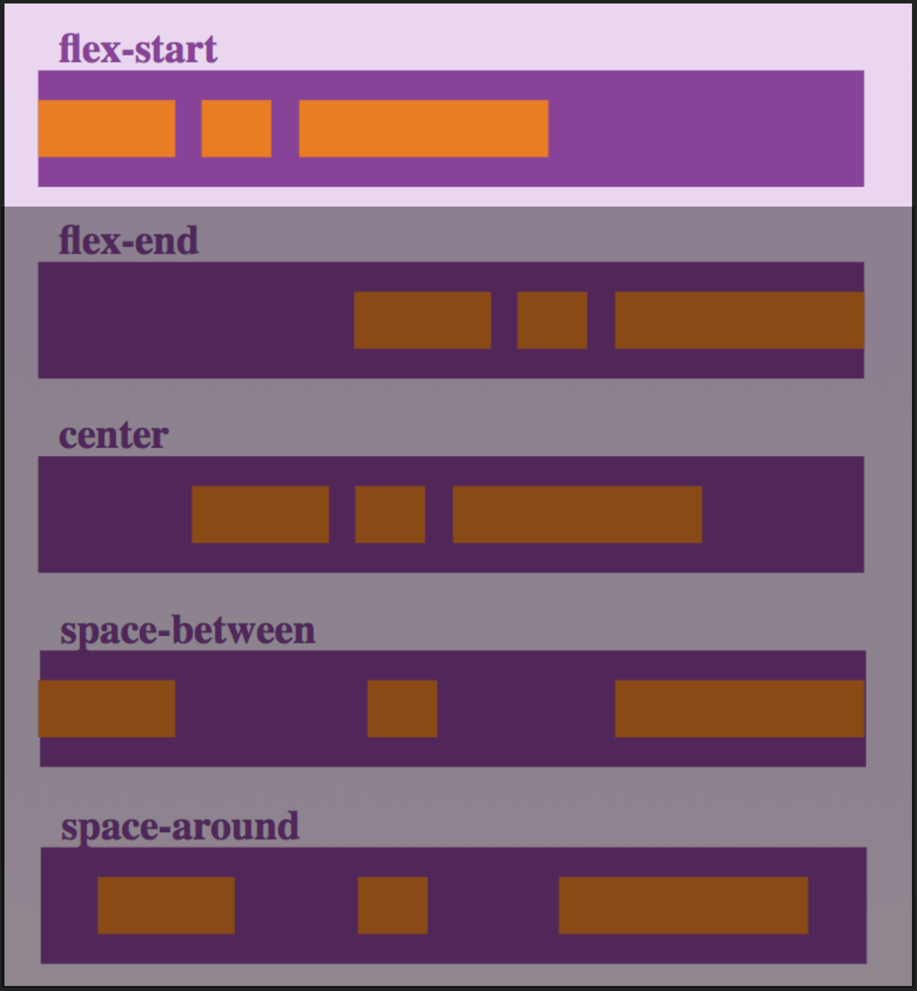
css-tricks.com

justify-content

defines alignment along the primary axis. This could be horizontal or vertical, depending on `flex-direction`.

`flex-start`: flex items packed toward start line.

This is the default behavior.

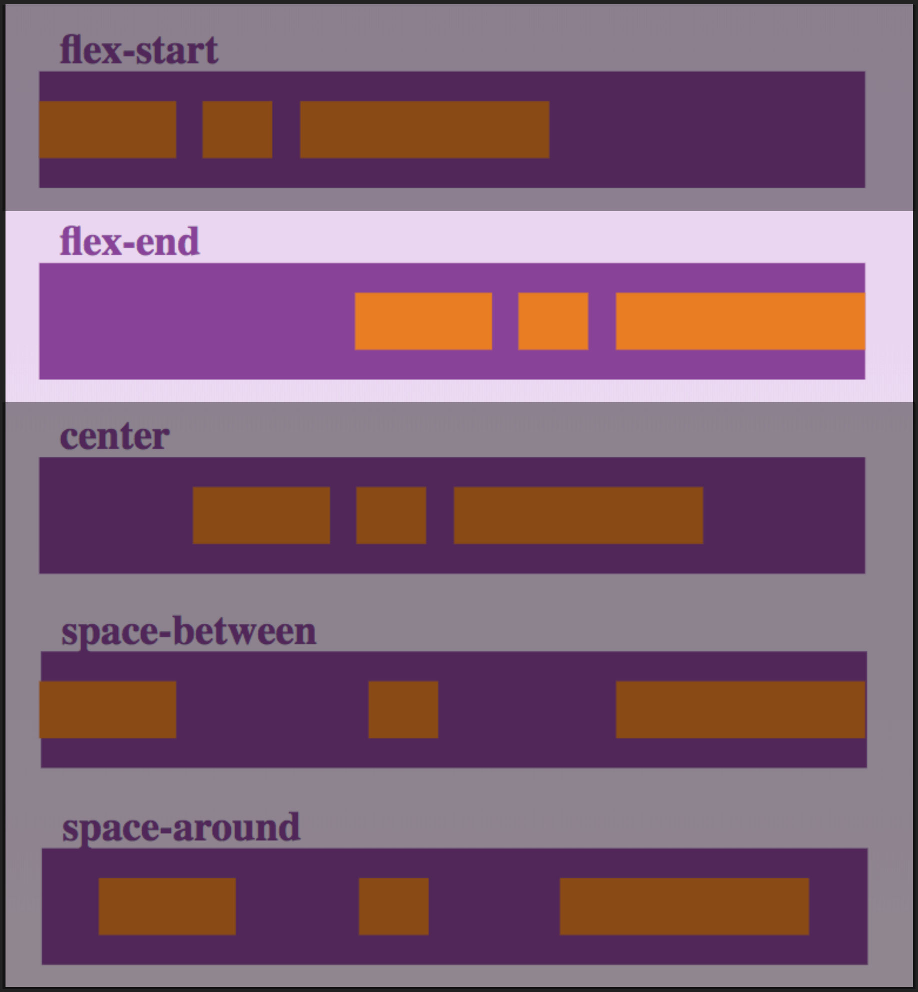


css-tricks.com

justify-content

defines alignment along the primary axis. This could be horizontal or vertical, depending on `flex-direction`.

`flex-end`: flex items packed toward end line.

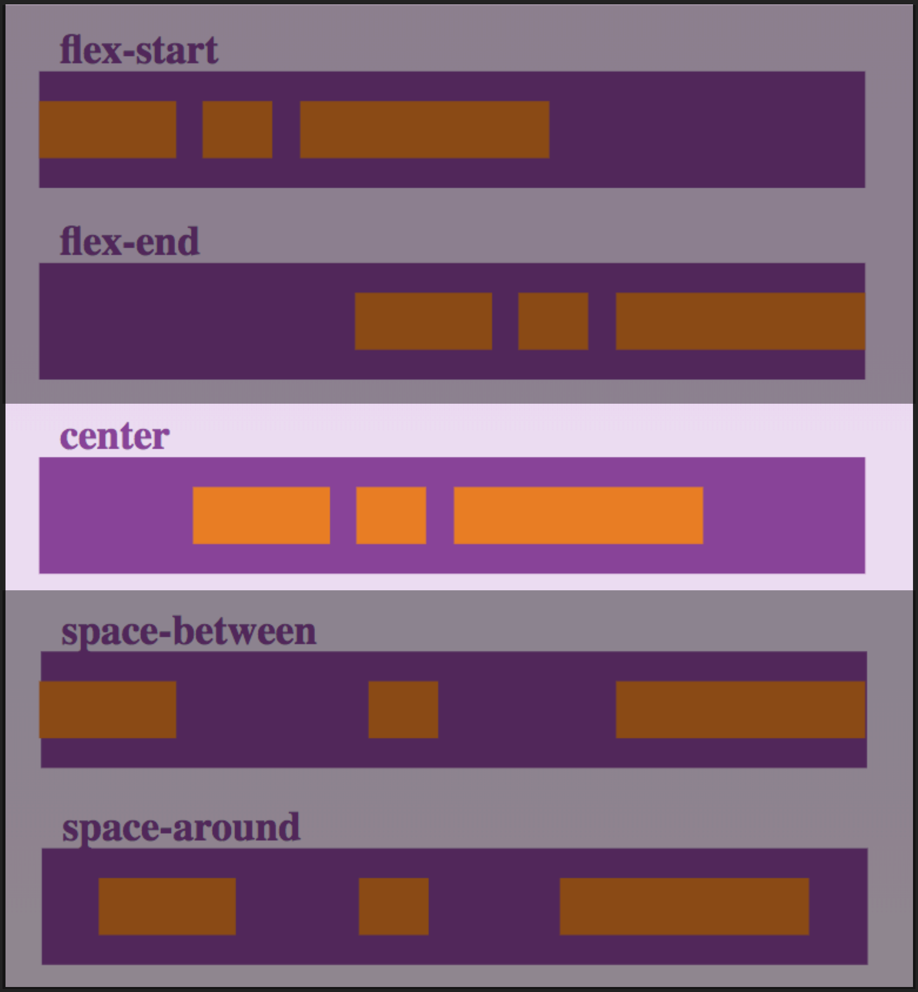


css-tricks.com

justify-content

defines alignment along the primary axis. This could be horizontal or vertical, depending on `flex-direction`.

`center`: flex items centered.

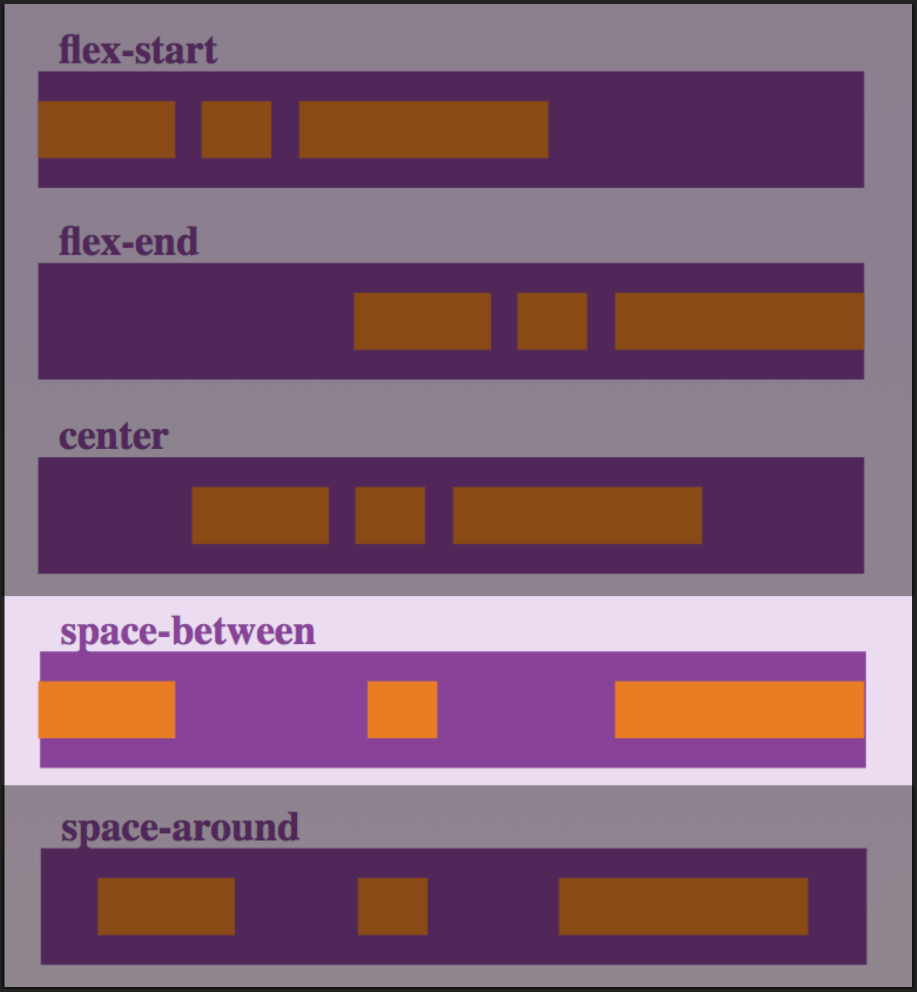


css-tricks.com

justify-content

defines alignment along the primary axis. This could be horizontal or vertical, depending on `flex-direction`.

`space-between`: flex items evenly distributed. First item on start line, last item on end line.

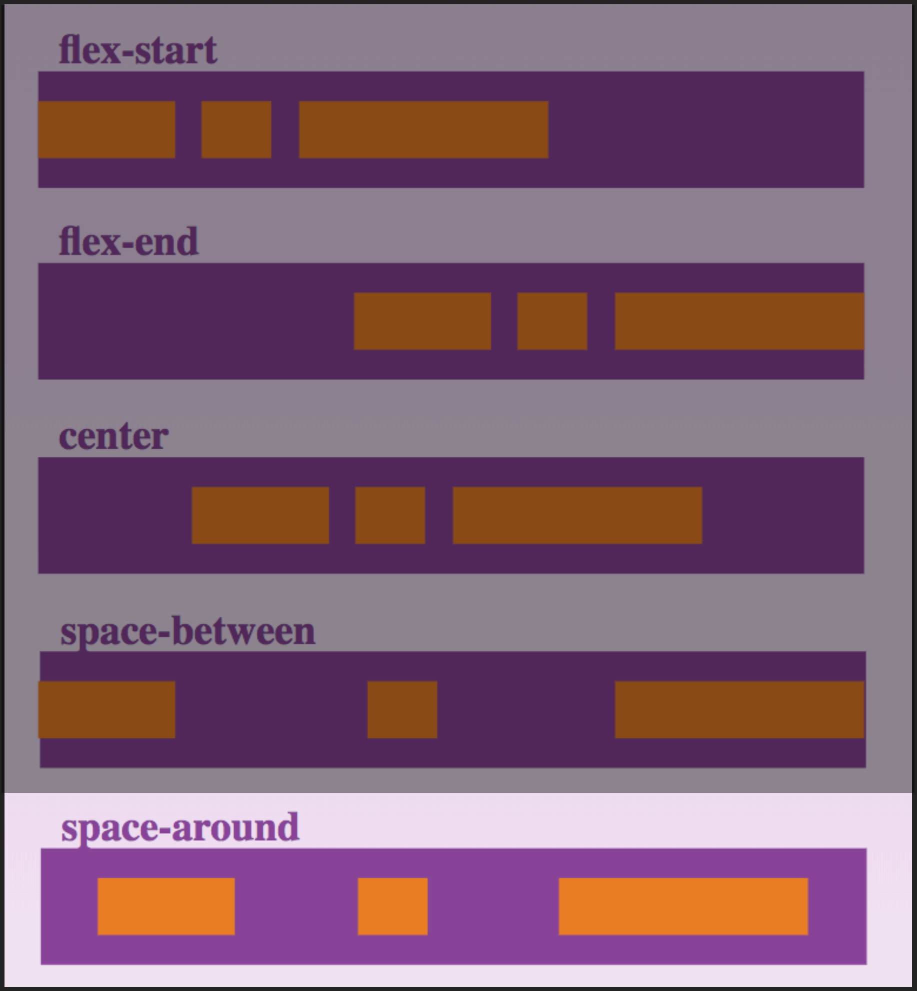


css-tricks.com

justify-content

defines alignment along the primary axis. This could be horizontal or vertical, depending on `flex-direction`.

`space-around`: flex items evenly distributed, with equal space around them.

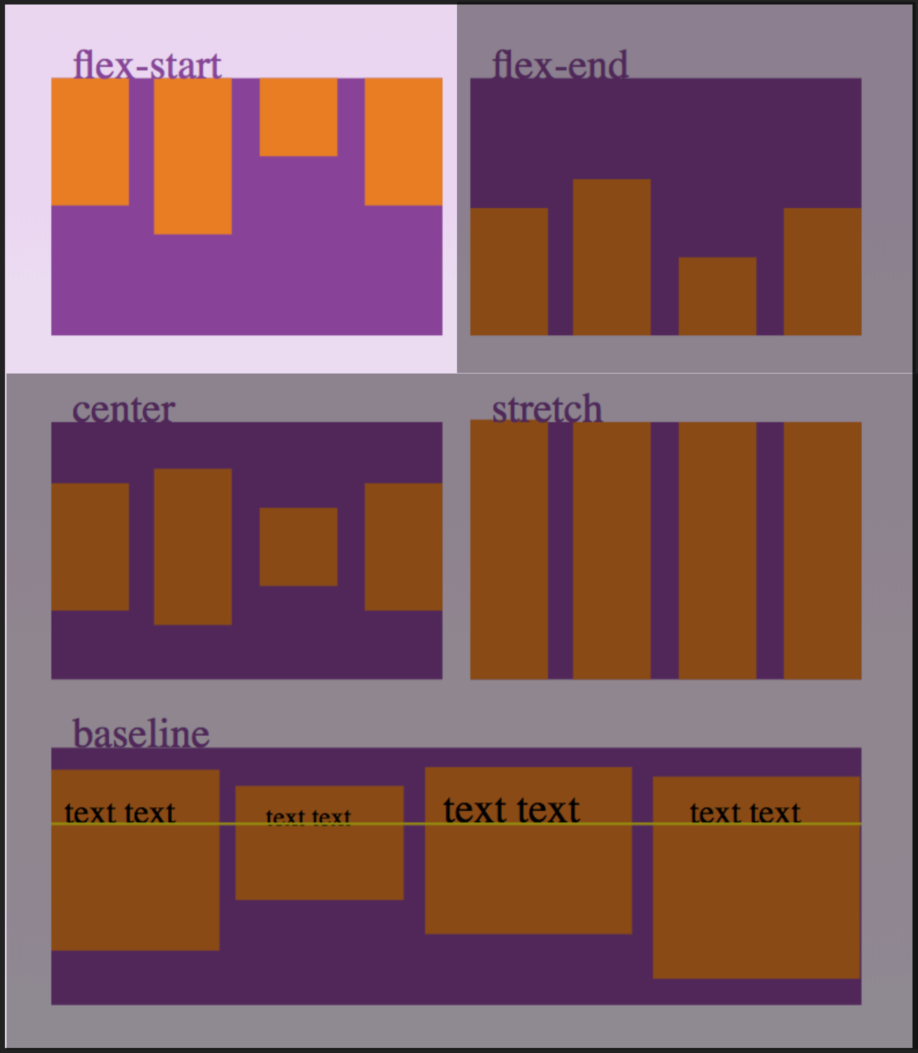


css-tricks.com

align-items

defines alignment along the secondary axis. This could be horizontal or vertical, depending on `flex-direction`.

`flex-start`: flex items aligned with start line.

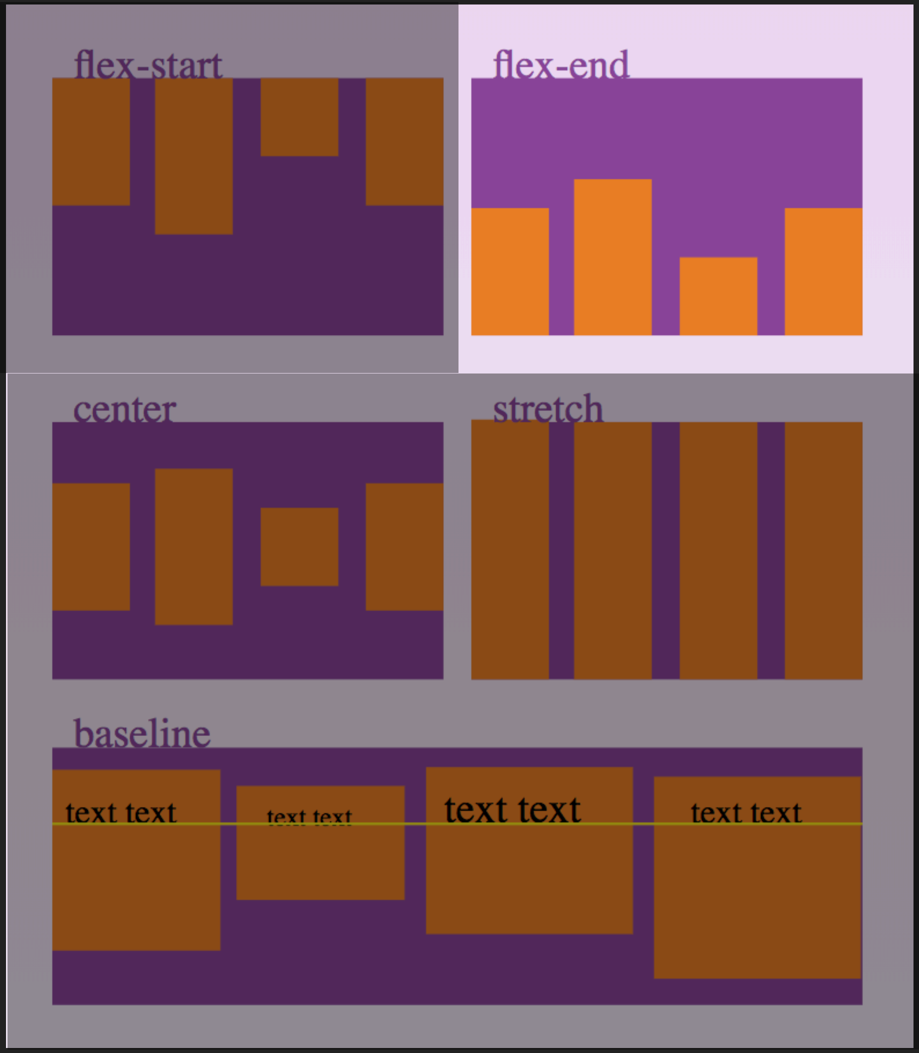


css-tricks.com

align-items

defines alignment along the secondary axis. This could be horizontal or vertical, depending on `flex-direction`.

`flex-end`: flex items aligned with end line.

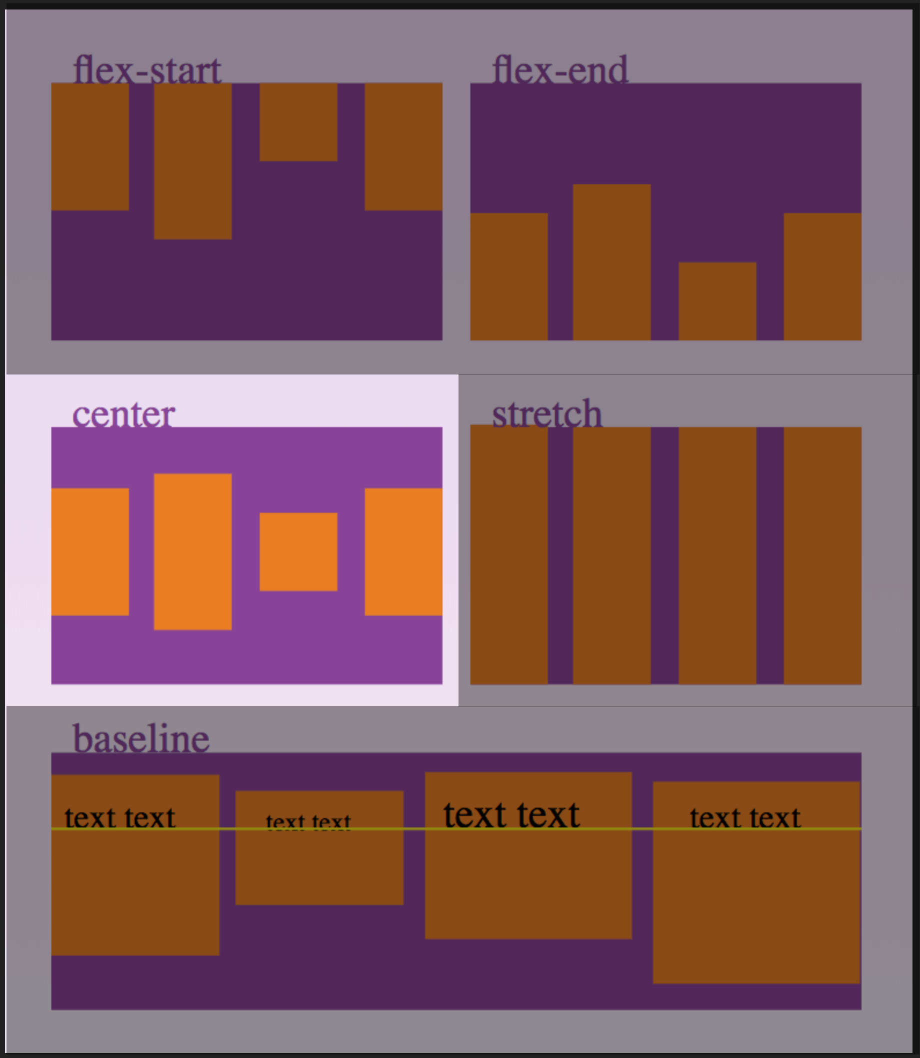


css-tricks.com

align-items

defines alignment along the secondary axis. This could be horizontal or vertical, depending on `flex-direction`.

`center`: flex items centered along secondary axis.

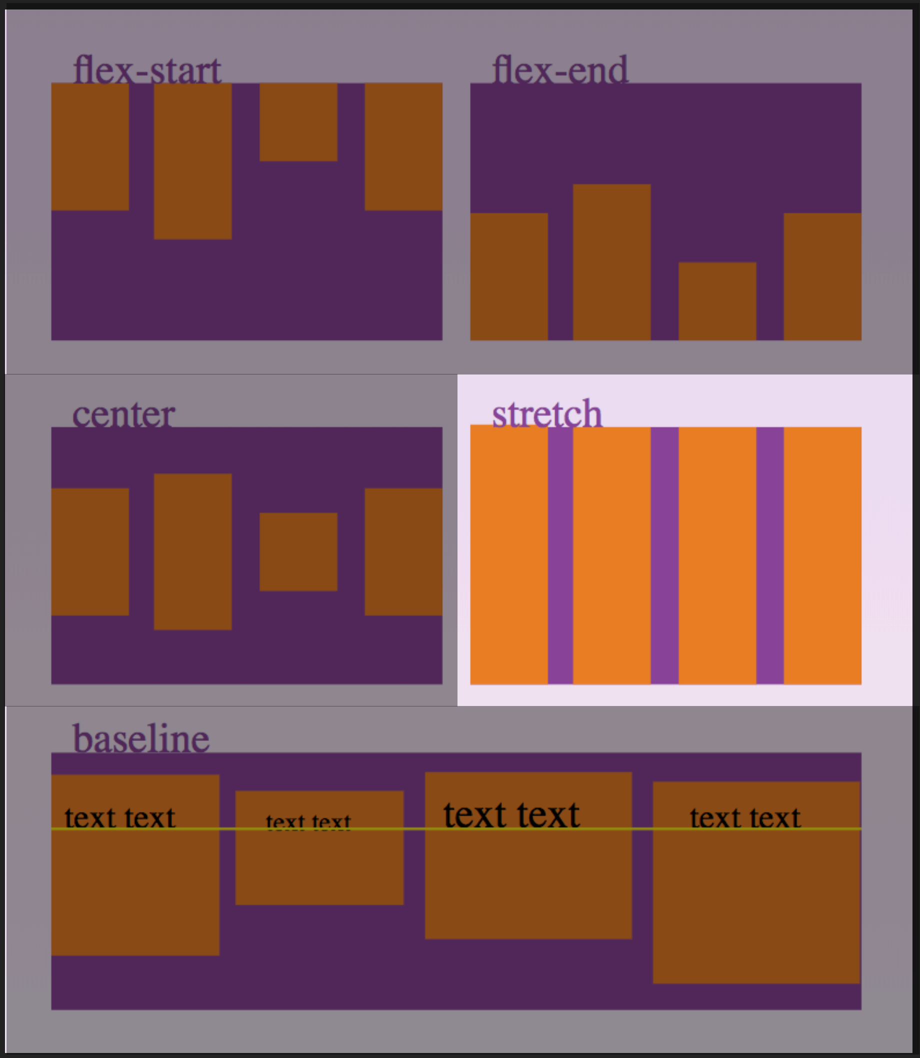


css-tricks.com

align-items

defines alignment along the secondary axis. This could be horizontal or vertical, depending on `flex-direction`.

`stretch`: flex items stretch to fill container.

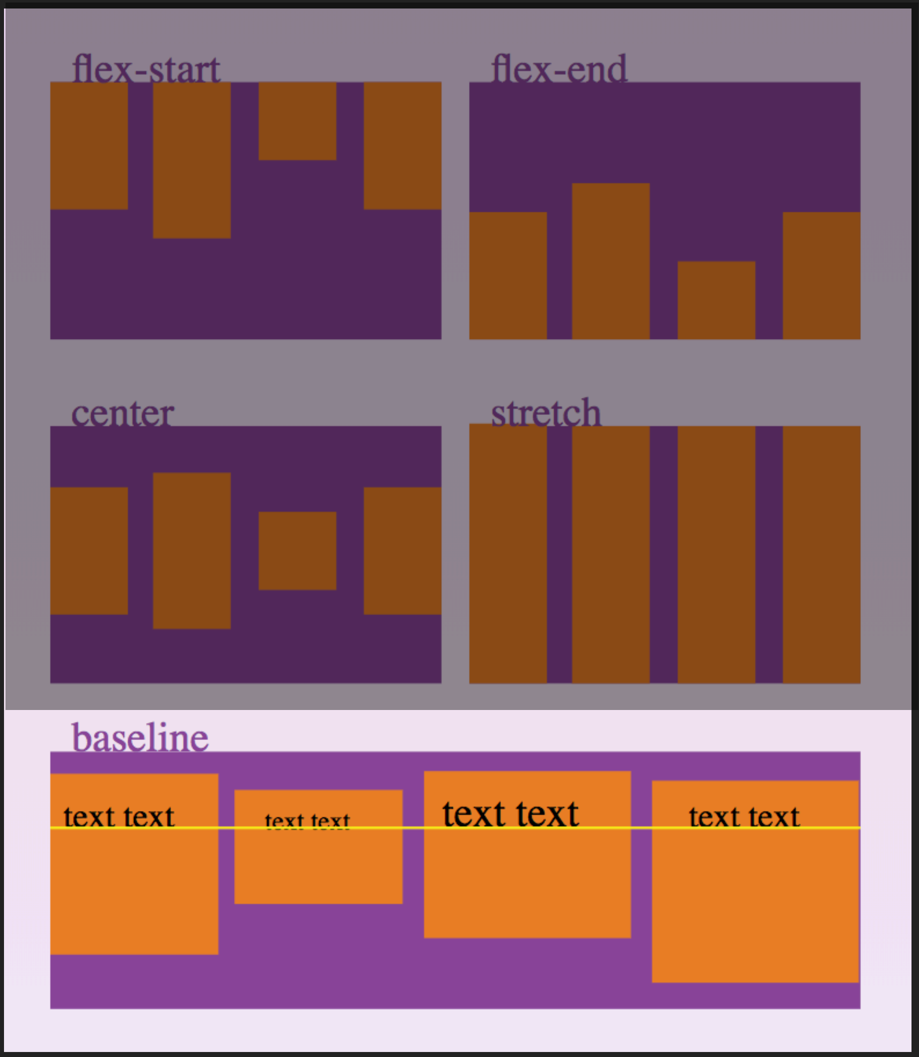


css-tricks.com

align-items

defines alignment along the secondary axis. This could be horizontal or vertical, depending on `flex-direction`.

`baseline`: flex items align along content baseline.



css-tricks.com

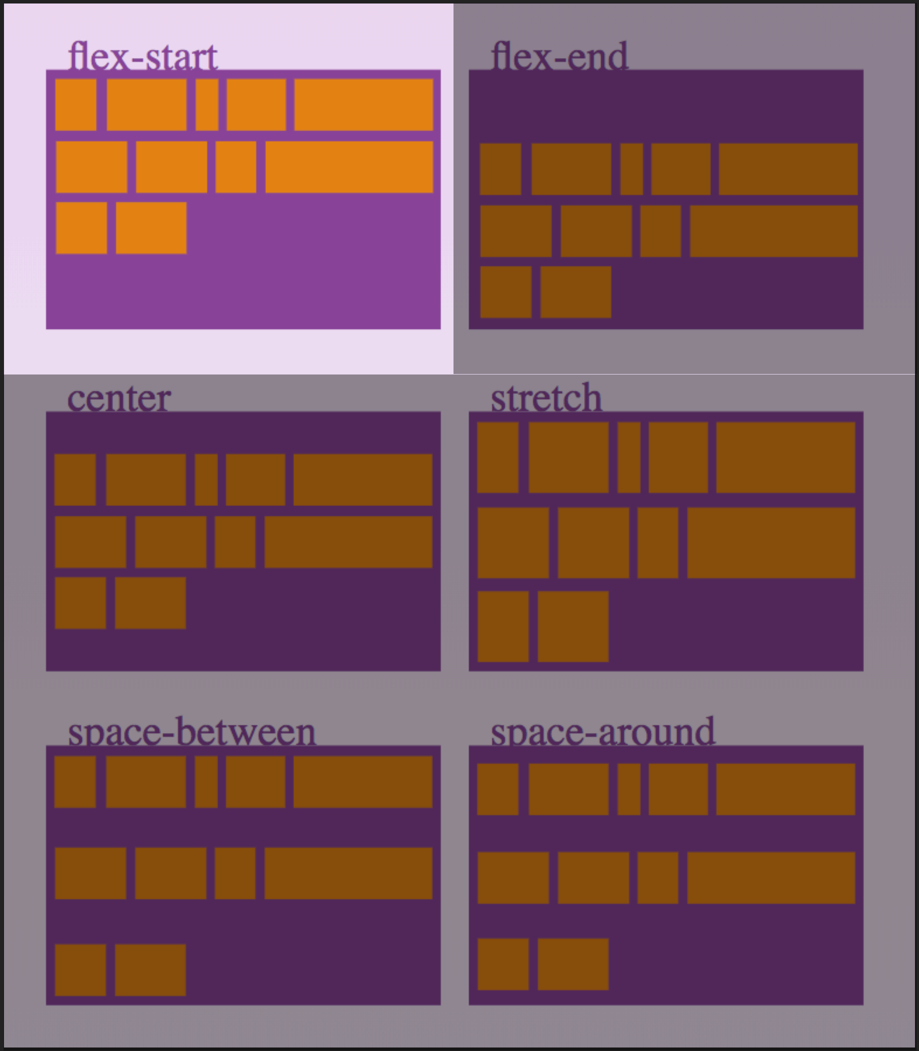
align-content

When flex items are in a single line, `align-content` has no effect.

align-content

When flex containers have multiple lines of content, `align-content` distributes lines within the container using the secondary axis.

`flex-start`: flex items packed to start of container.

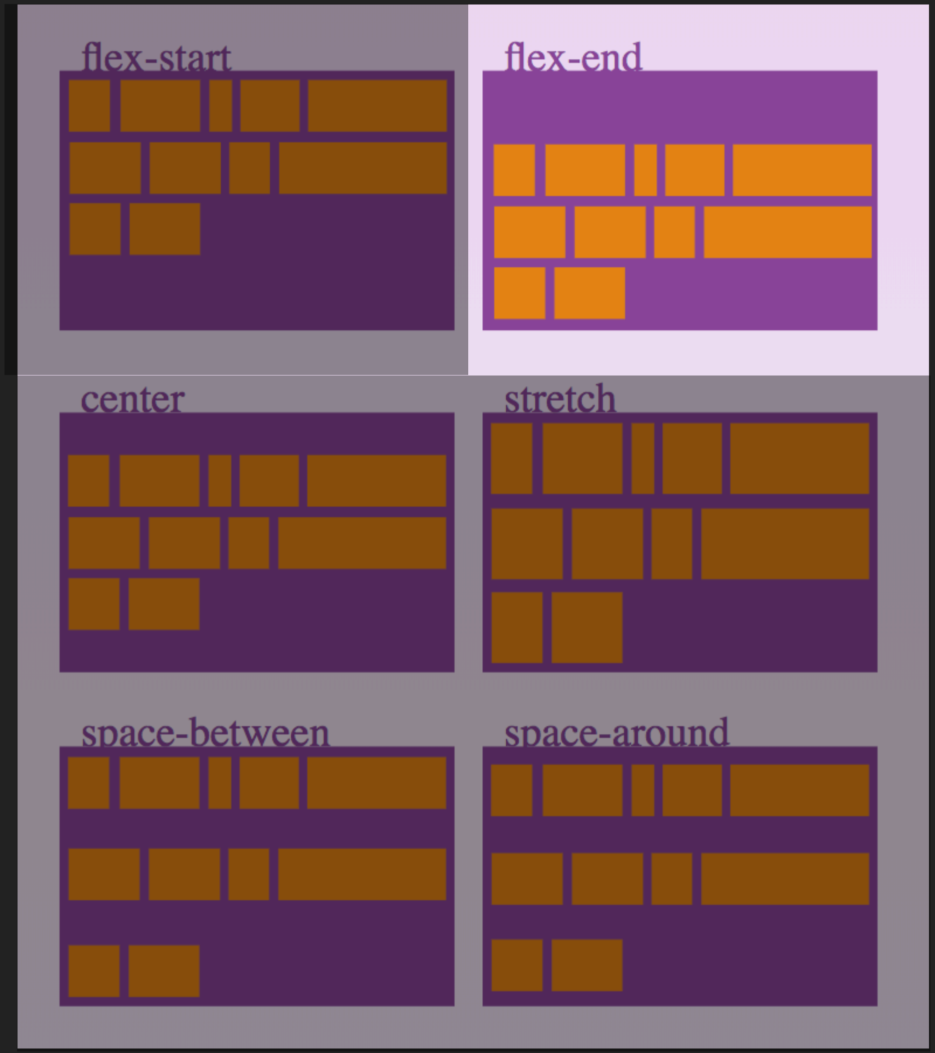


css-tricks.com

align-content

When flex containers have multiple lines of content, `align-content` distributes lines within the container using the secondary axis.

`flex-end`: flex items packed to end of container.

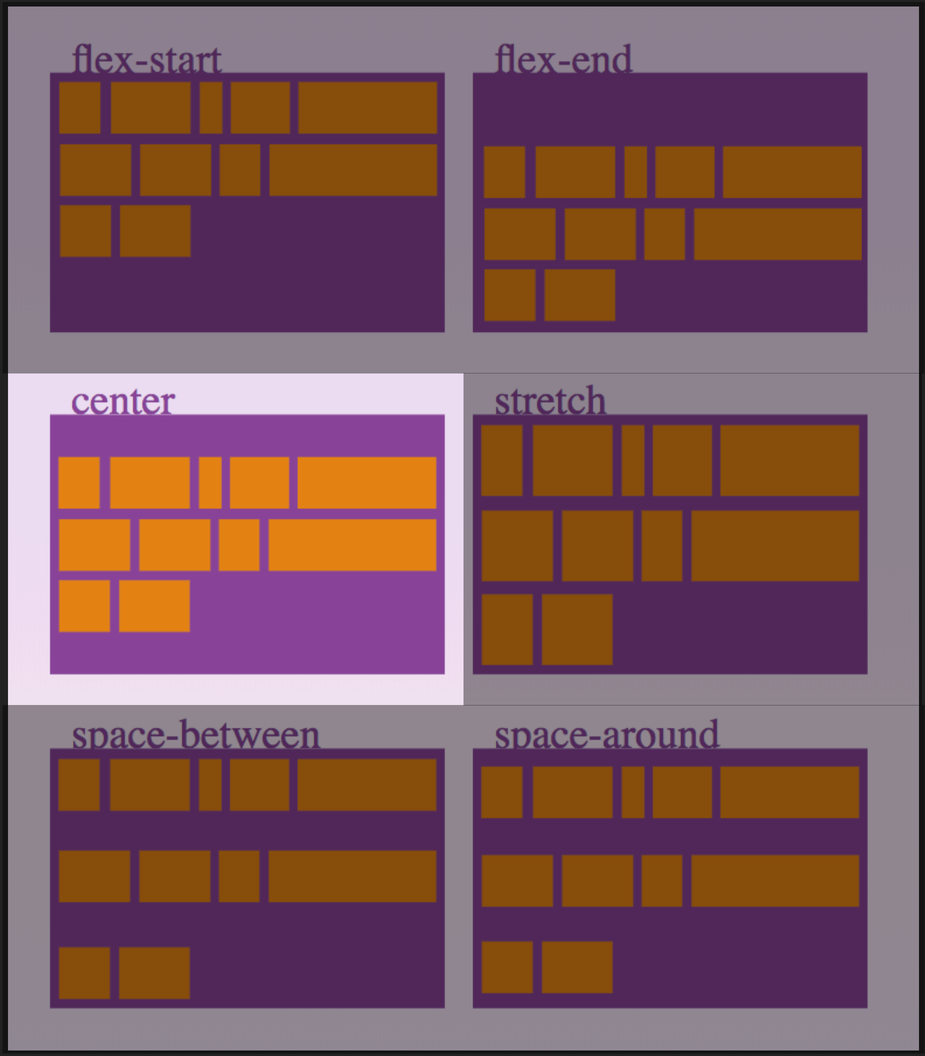


css-tricks.com

align-content

When flex containers have multiple lines of content, `align-content` distributes lines within the container using the secondary axis.

`center`: flex items packed to center of container.

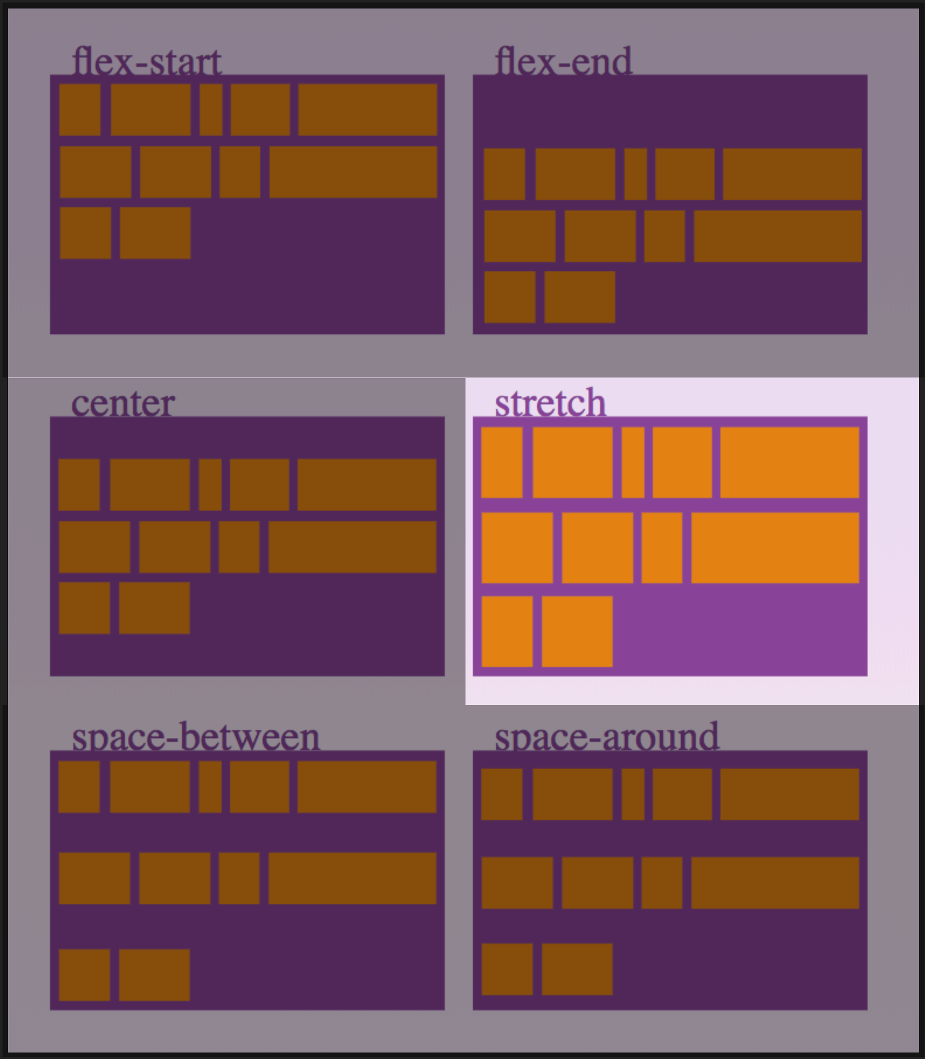


css-tricks.com

align-content

When flex containers have multiple lines of content, `align-content` distributes lines within the container using the secondary axis.

`stretch`: flex items stretch to fill container space.

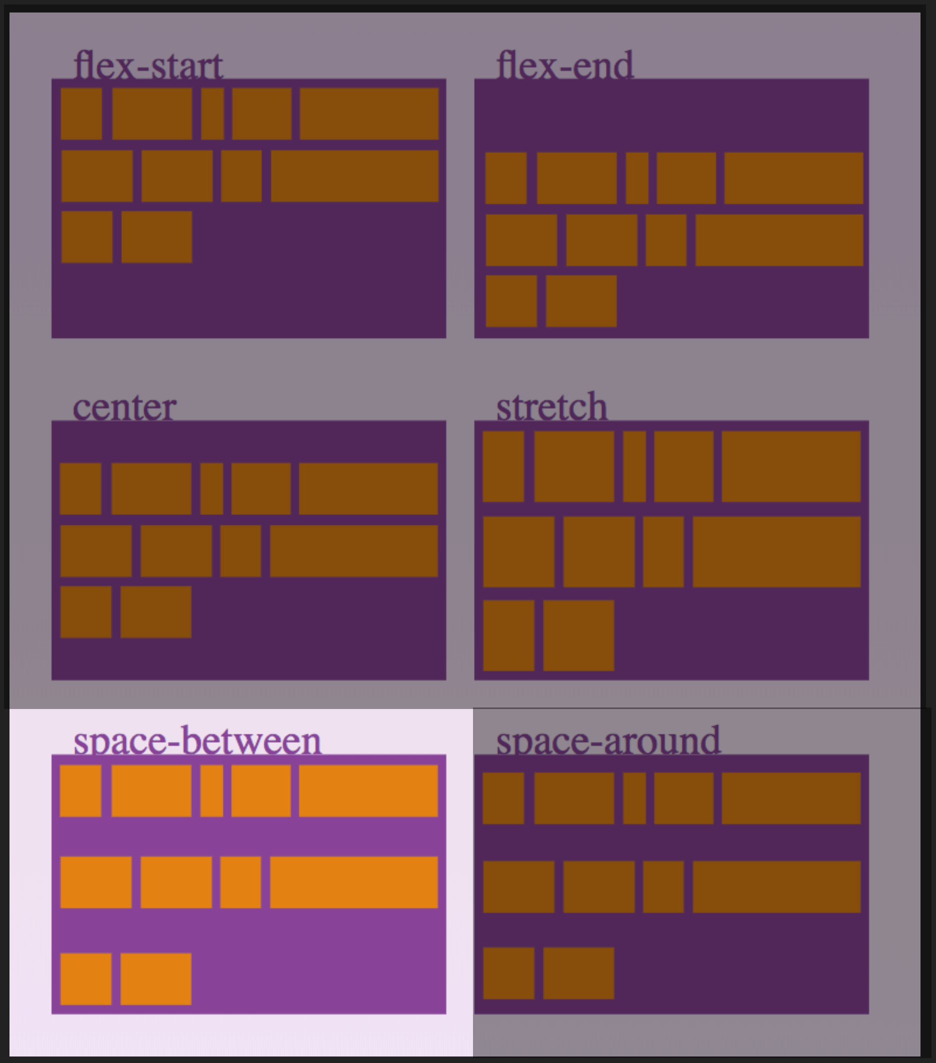


css-tricks.com

align-content

When flex containers have multiple lines of content, `align-content` distributes lines within the container using the secondary axis.

`space-between`: flex items evenly distributed. First item on start line, last item on end line.

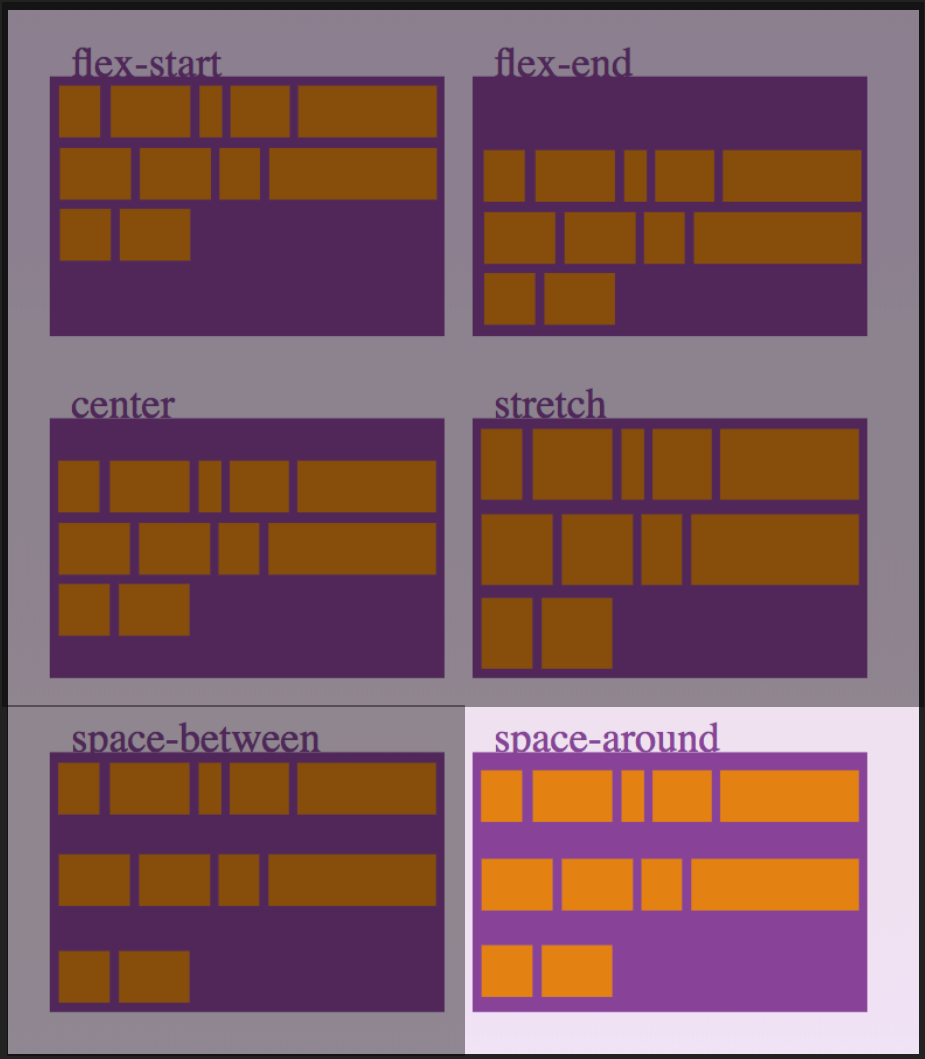


css-tricks.com

align-content

When flex containers have multiple lines of content, `align-content` distributes lines within the container using the secondary axis.

`space-around`: flex items evenly distributed, with equal space around them.

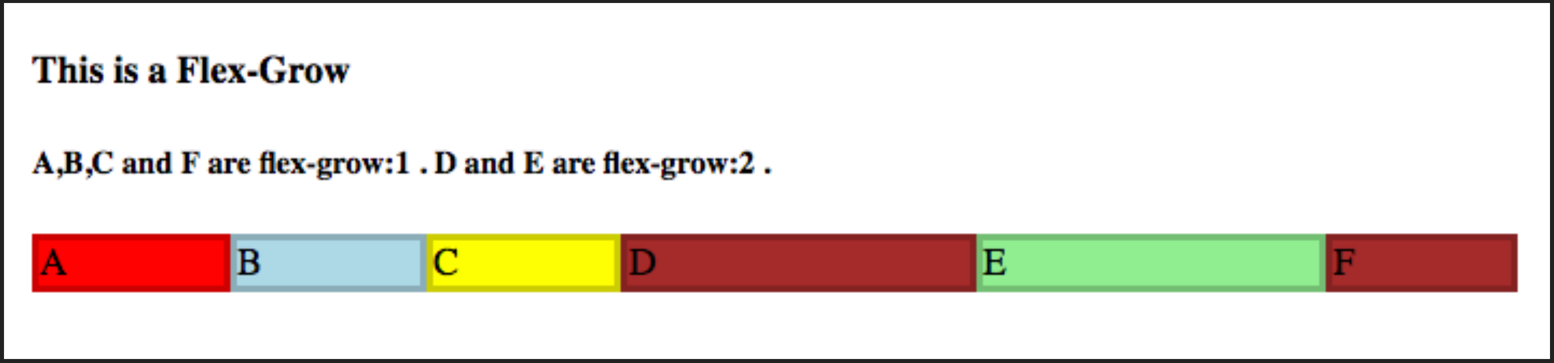


css-tricks.com

flex-grow

This specifies the flex grow factor of a flex item, meaning the amount of space inside the flex container the item should take up.

The flex grow factor of a flex item is relative to the size of the other children in the flex-container.



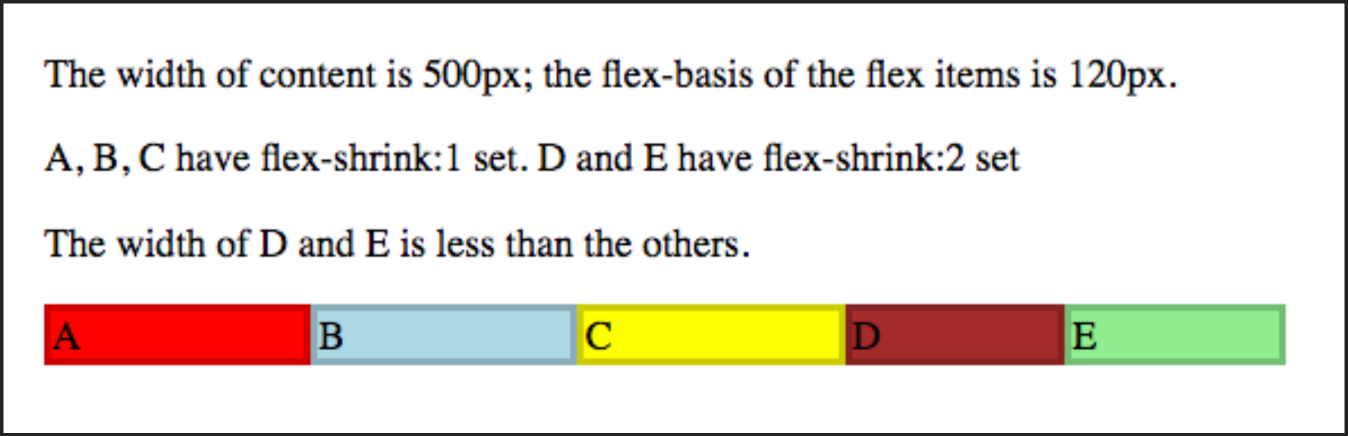
This is a Flex-Grow

A,B,C and F are flex-grow:1 . D and E are flex-grow:2 .

developer.mozilla.org

flex-shrink

This specifies the flex shrink factor of a flex item. Flex items will shrink to fill the container according to the flex-shrink number, when the default width of flex items is wider than the flex container.



The width of content is 500px; the flex-basis of the flex items is 120px.

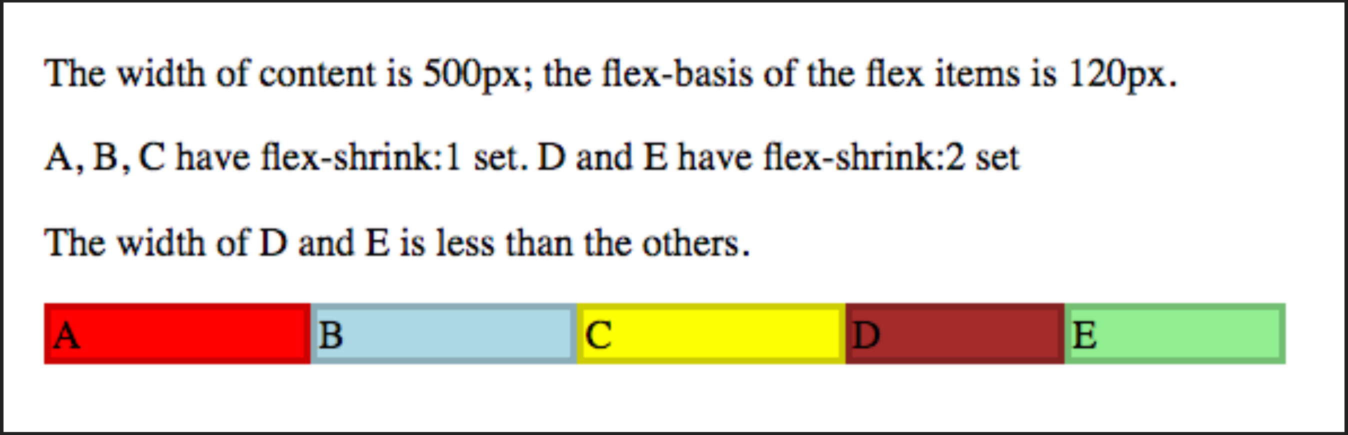
A, B, C have flex-shrink:1 set. D and E have flex-shrink:2 set

The width of D and E is less than the others.

developer.mozilla.org

flex-basis

This specifies the initial main size of a flex item. This property determines the size of the content-box unless specified otherwise using `box-sizing`.



The width of content is 500px; the flex-basis of the flex items is 120px.

A, B, C have flex-shrink:1 set. D and E have flex-shrink:2 set

The width of D and E is less than the others.

developer.mozilla.org

flex (shorthand)

Three flex properties

- `flex-grow`
- `flex-shrink`
- `flex-basis`

Can be set at once using the flex shorthand.

```css
aside {
    flex: 0 0 240px;
}
```

flex (shorthand)

Three flex properties

- `flex-grow`
- `flex-shrink`
- `flex-basis`

Can be set at once using the flex shorthand.

```
aside {
    flex: 0 0 240px;
}
```

(In this example, the flex item remains a constant 240px and will neither grow nor shrink.)