Metropolitan Community College
WORKFORCE
INNOVATION DIVISION

CODE SCHOOL

CSS: Flexbox and CSS Grid

Derek Watson | October 8, 2020

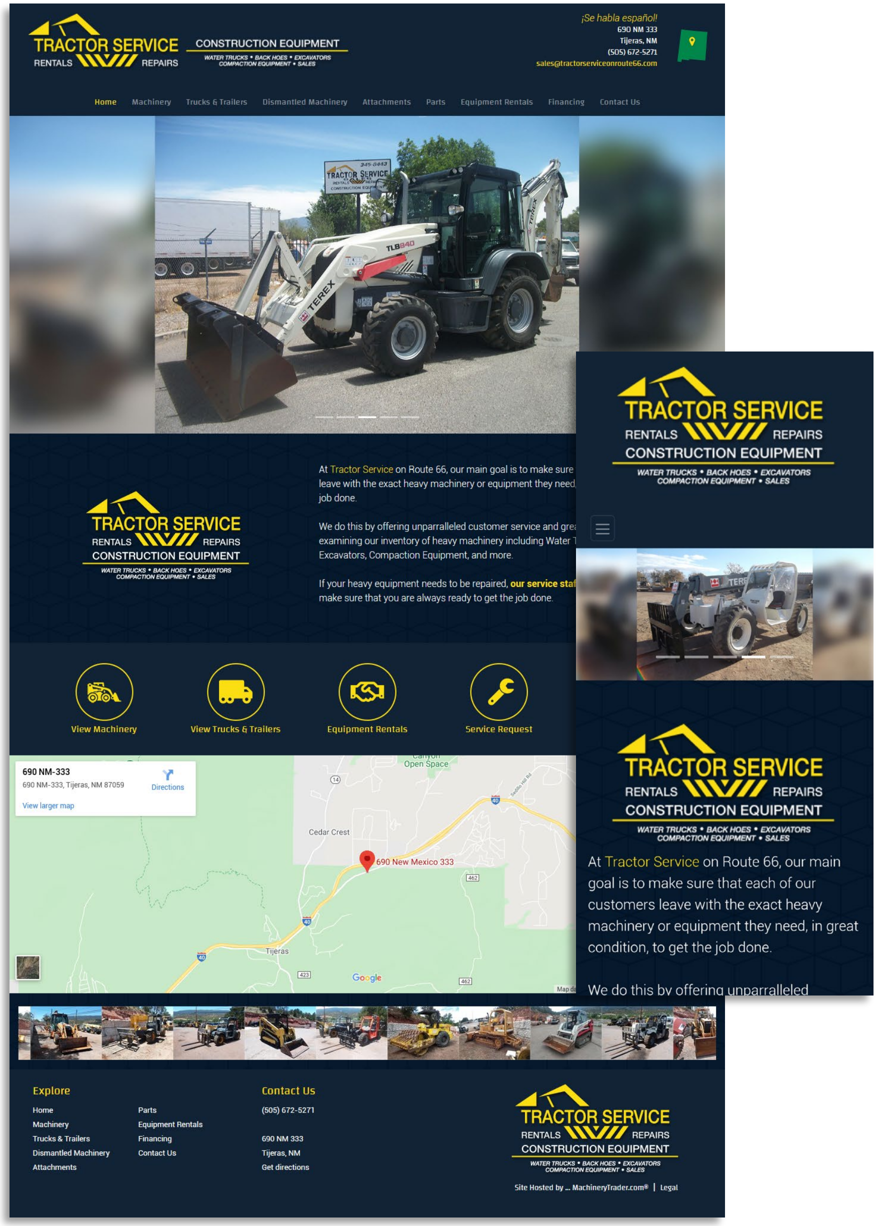
Introduction

Hi, my name is Derek.

CSS Basics: Layout

Columns, rows, alignment, justification.

Layout



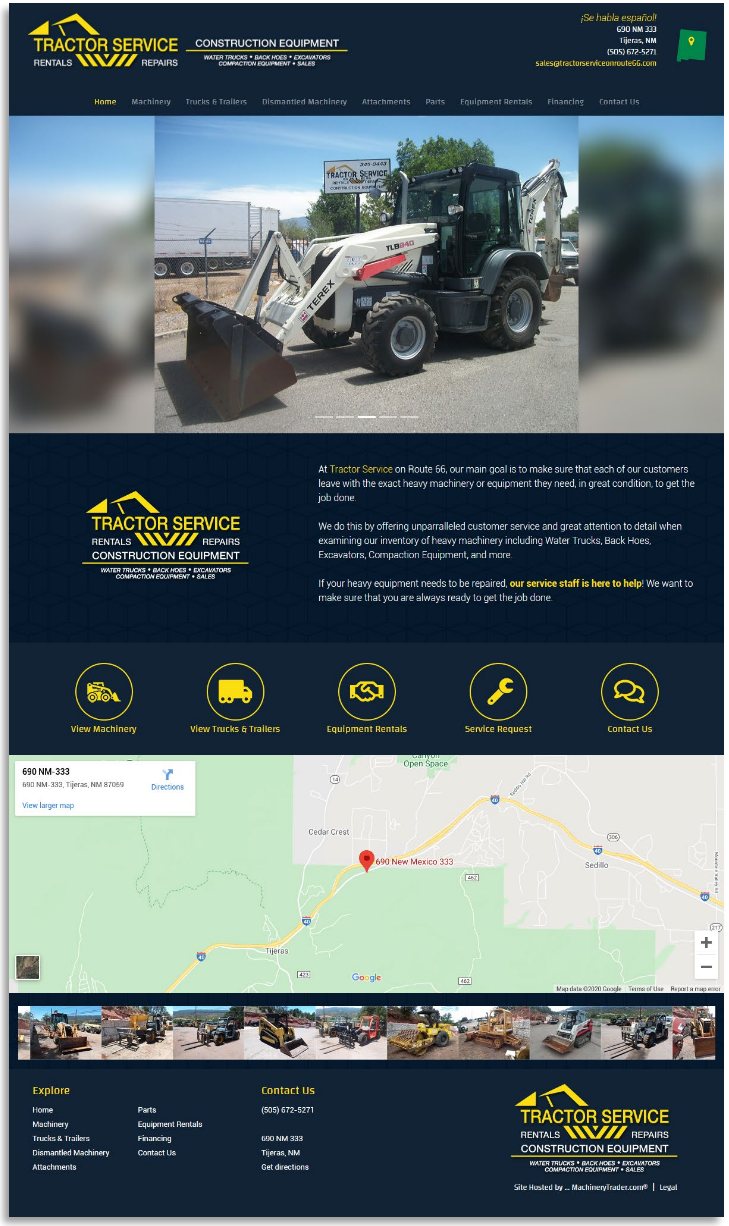
At Tractor Service on Route 66, our main goal is to make sure each of our customers leave with the exact heavy machinery or equipment they need, in great condition, to get the job done.

We do this by offering unparralleled customer service and great ... examining our inventory of heavy machinery including Water T... Excavators, Compaction Equipment, and more.

If your heavy equipment needs to be repaired, our service staff ... make sure that you are always ready to get the job done.

Arguably the most important as well as complex part of CSS is layout.

That is, how to get content in a page to be aligned the way you want them across browsers and screen sizes.



At Tractor Service on Route 66, our main goal is to make sure that each of our customers leave with the exact heavy machinery or equipment they need, in great condition, to get the job done.

We do this by offering unparralleled customer service and great attention to detail when examining our inventory of heavy machinery including Water Trucks, Back Hoes, Excavators, Compaction Equipment, and more.

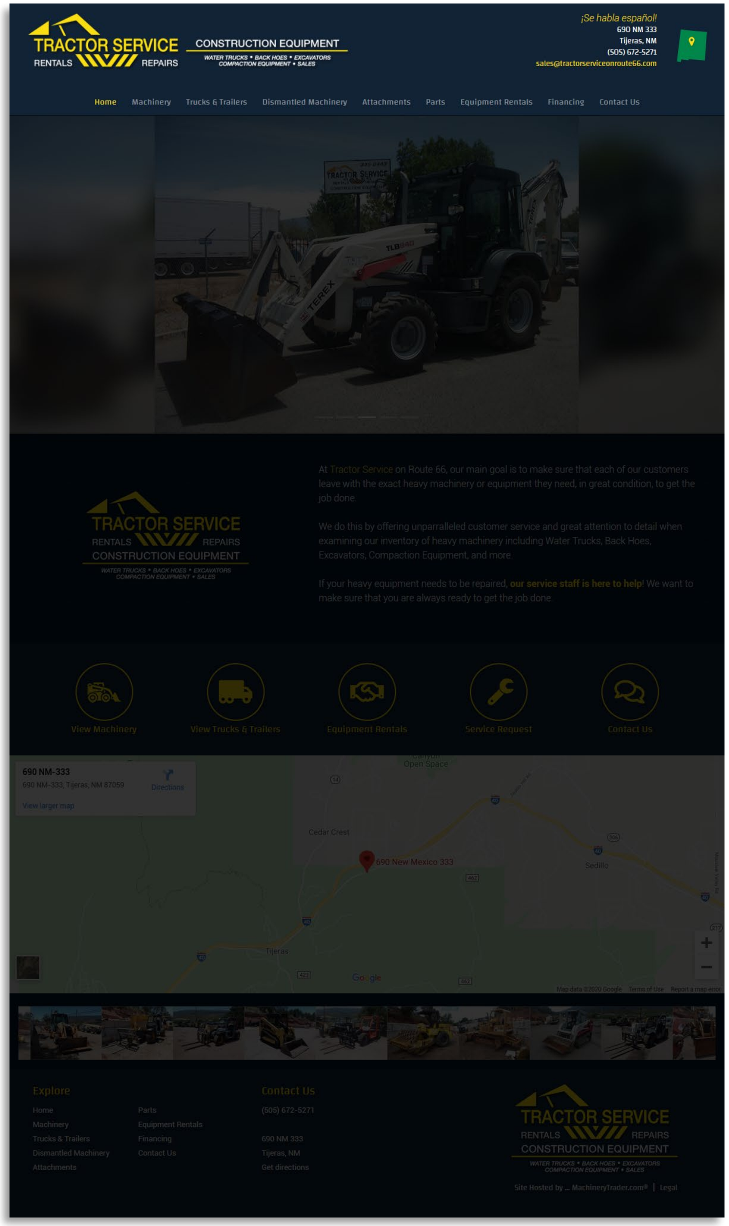
If your heavy equipment needs to be repaired, **our service staff is here to help**! We want to make sure that you are always ready to get the job done.

Example

Looking at the desktop version of this screen, you can easily see many nested columns and rows of information.

Example

Let's focus on the header.



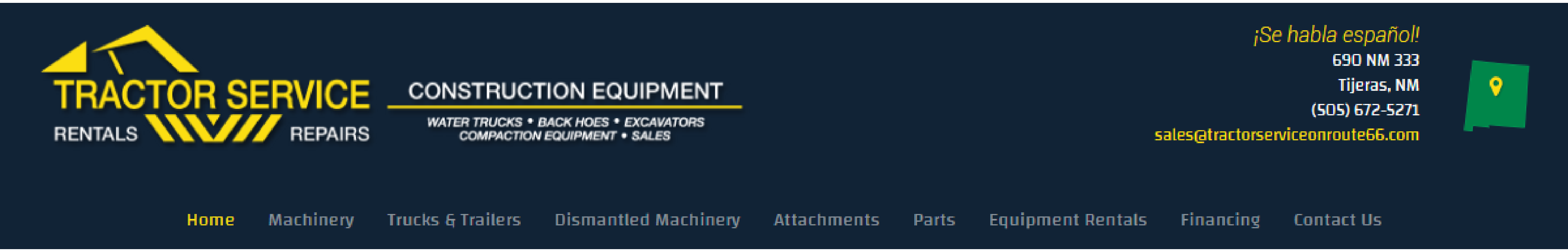
At Tractor Service on Route 66, our main goal is to make sure that each of our customers leave with the exact heavy machinery or equipment they need, in great condition, to get the job done.

We do this by offering unparralleled customer service and great attention to detail when examining our inventory of heavy machinery including Water Trucks, Back Hoes, Excavators, Compaction Equipment, and more.

If your heavy equipment needs to be repaired, **our service staff is here to help**! We want to make sure that you are always ready to get the job done.

Example

Let's focus on the header.

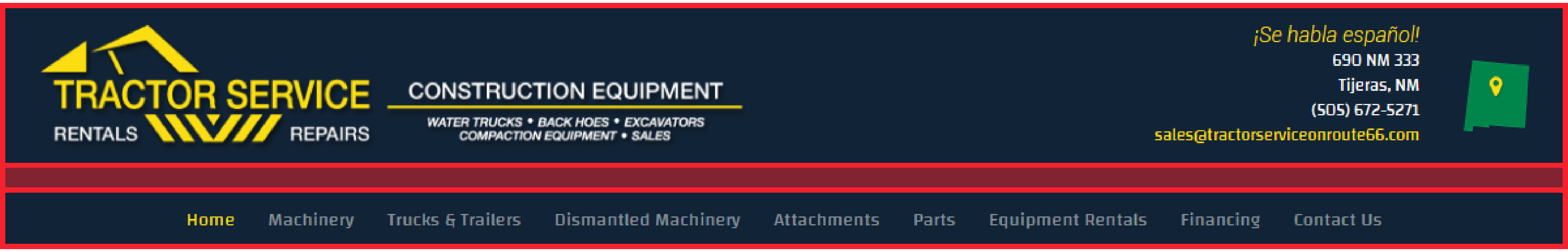

TRACTOR SERVICE
RENTALS REPAIRS
CONSTRUCTION EQUIPMENT
WATER TRUCKS • BACK HOES • EXCAVATORS
COMPACTION EQUIPMENT • SALES

¡Se habla español!
690 NM 333
Tijeras, NM
(505) 672-5271
sales@tractorserviceonroute66.com

Home Machinery Trucks & Trailers Dismantled Machinery Attachments Parts Equipment Rentals Financing Contact Us

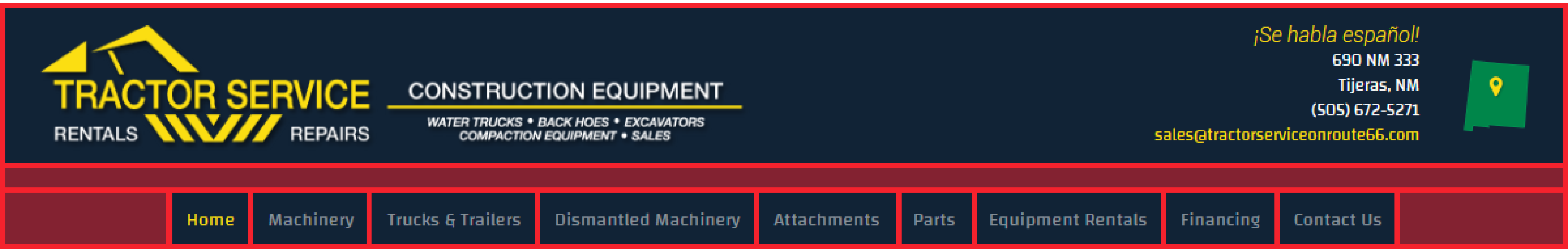
Example

First, we can slice this into a container with two rows, and some space in-between.



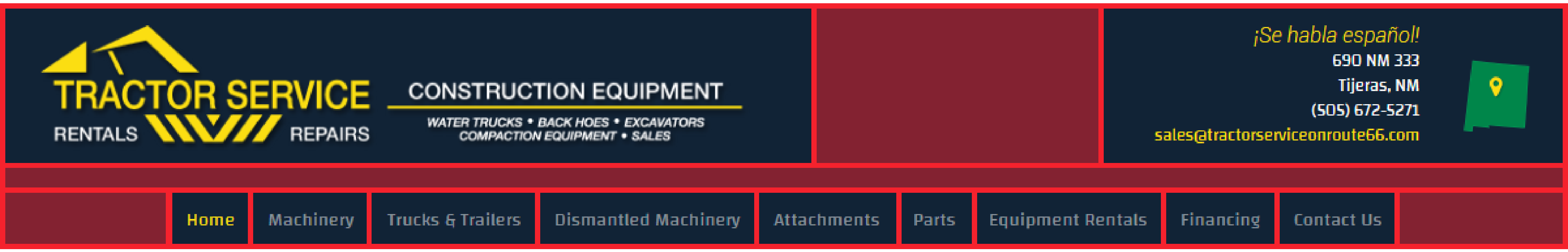
Example

Next, let's divide the nav bar row and divide it into uneven (content-based) columns, with some even spacing on the left and right.

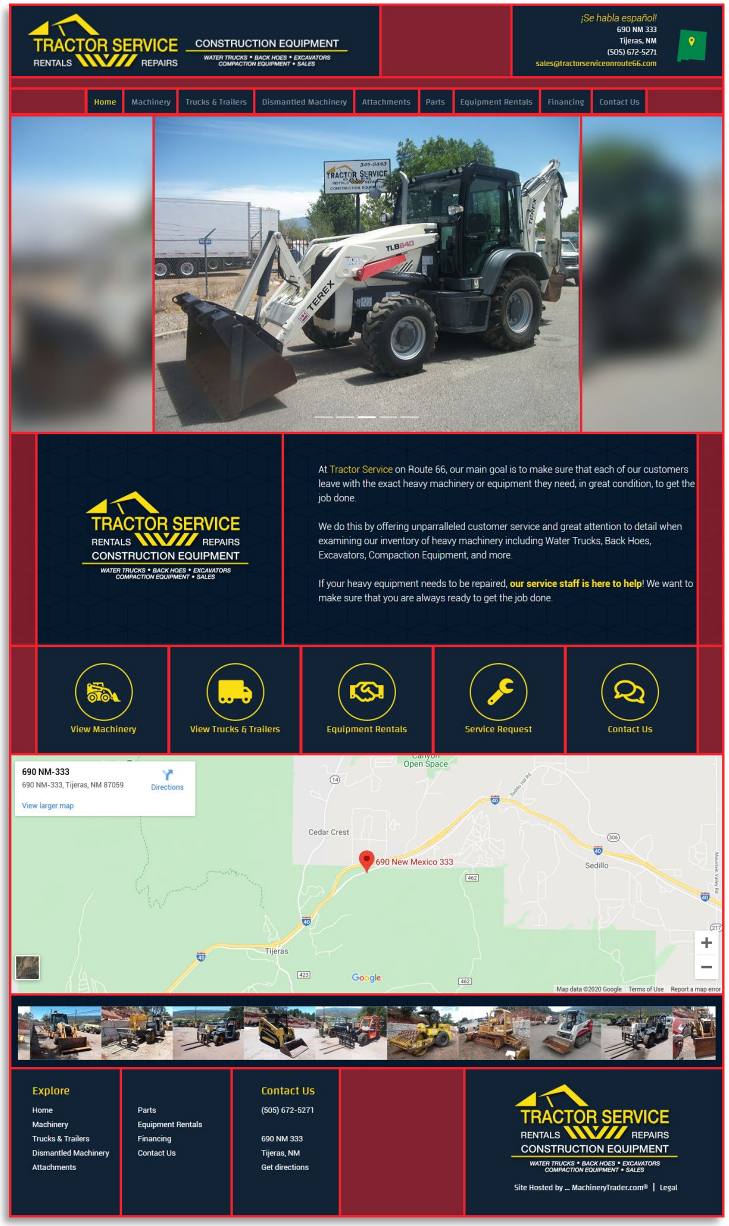


Example

Lastly, we can divide the top row into two columns with some space in between.



Example



TRACTOR SERVICE
RENTALS REPAIRS
CONSTRUCTION EQUIPMENT
WATER TRUCKS • BACK HOES • EXCAVATORS
COMPACTION EQUIPMENT • SALES

¡Se habla español!
690 NM 333
Tijeras, NM
(505) 672-5271
sales@tractorserviceonroute66.com

Home | Machinery | Trucks & Trailers | Dismantled Machinery | Attachments | Parts | Equipment Rentals | Financing | Contact Us

At Tractor Service on Route 66, our main goal is to make sure that each of our customers leave with the exact heavy machinery or equipment they need, in great condition, to get the job done.

We do this by offering unparralleled customer service and great attention to detail when examining our inventory of heavy machinery including Water Trucks, Back Hoes, Excavators, Compaction Equipment, and more.

If your heavy equipment needs to be repaired, **our service staff is here to help**! We want to make sure that you are always ready to get the job done.

Explore
Home
Machinery
Trucks & Trailers
Dismantled Machinery
Attachments

Parts
Equipment Rentals
Financing
Contact Us

Contact Us
(505) 672-5271

690 NM 333
Tijeras, NM
Get directions

Site Hosted by ... MachineryTrader.com® | Legal

So how on earth are we going to use CSS to get all of these elements to align?

And what about mobile??

A Tale of Two Methods

Flexbox and CSS Grid

A Tale of Two Methods

The problem:

Front end developers need a way to organize content on a web page that is consistent across browsers and responds to all screen sizes.

A Tale of Two Methods

The old way:

Using tables, floats, inline-block display, and plenty of other properties. These are almost impossible to maintain and come with many limitations (responsiveness, vertically centering, etc).

A Tale of Two Methods

The new way(s):

Flexible Box (Flexbox) and CSS Grid (or just Grid)!

A Tale of Two Methods

Flexbox

- Been around longer
- Used in Bootstrap's grid, among others
- Content-aware
- One-dimensional*
- Focus on flexibility, content, and space management

CSS Grid

- Slightly newer solution
- More content-agnostic
- Two-dimensional
- Focus on layout of page

A Tale of Two Methods

Flexbox

- Been around longer
- Used in Bootstrap's grid, among others
- Content-aware
- One-dimensional*
- Focus on content, flexibility, and flow

CSS Grid

- Slightly newer solution
- More content-agnostic
- Two-dimensional
- Focus on layout of page

Method 1: Flexbox

The basis for most modern grids, including Bootstrap, Material, and Ant.

Murach ch. 9

Flexbox: Objectives

At the end of today, you will be able to:

- Use Flexbox to lay out a simple, mobile responsive web page
- Describe the use of Flexbox
- Describe key terms: flexbox, flex item, main axis, cross axis, and flex direction
- Explain justify-content and align-items

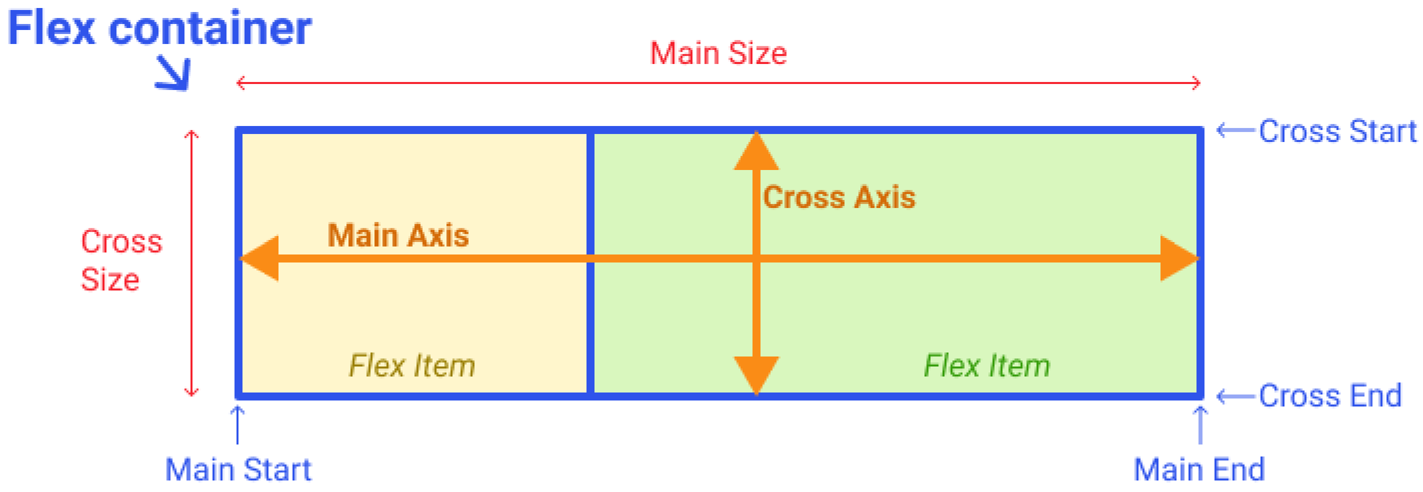
What is Flexible Box?

Flexbox Layout (aka Flexible Box), is a CSS module which…

> "…aims at providing a more efficient way to lay out, align, and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word 'flex')."

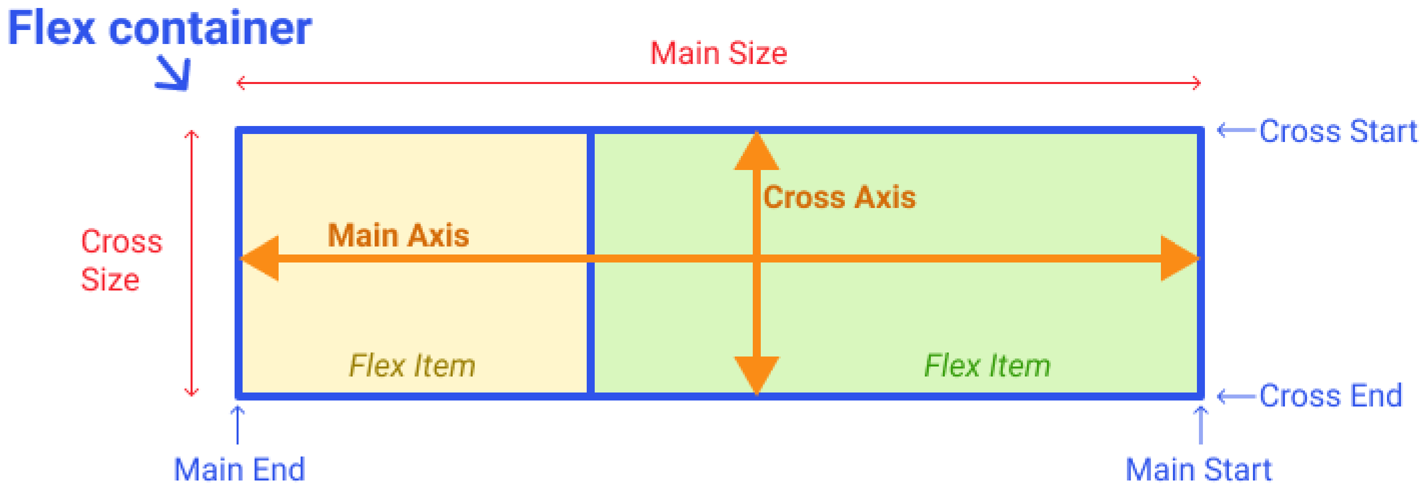
-A Complete Guide to Flexbox | CSS Tricks

The Flexible Box and `flex-direction`



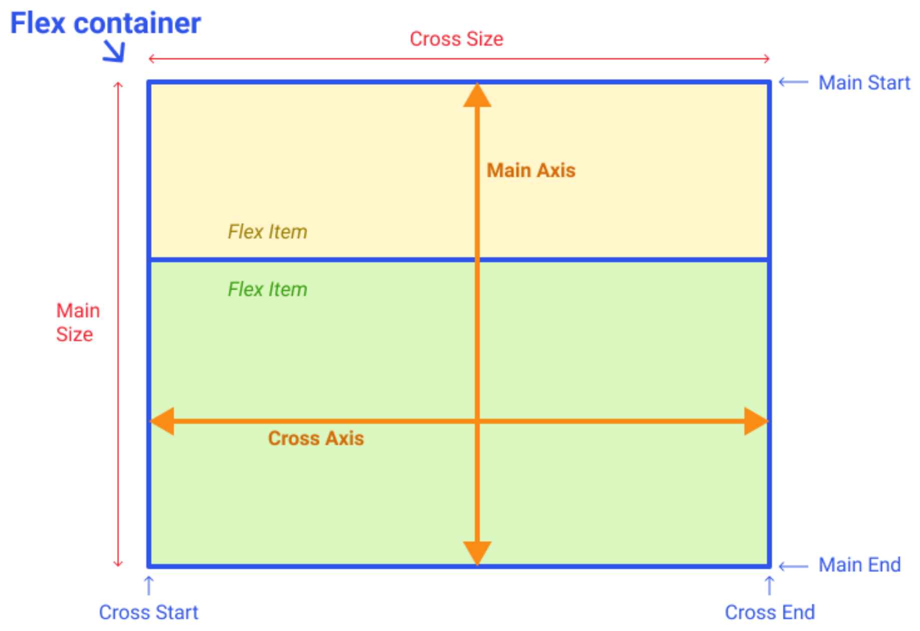
Flex container with `flex-direction = row`

The Flexible Box and `flex-direction`



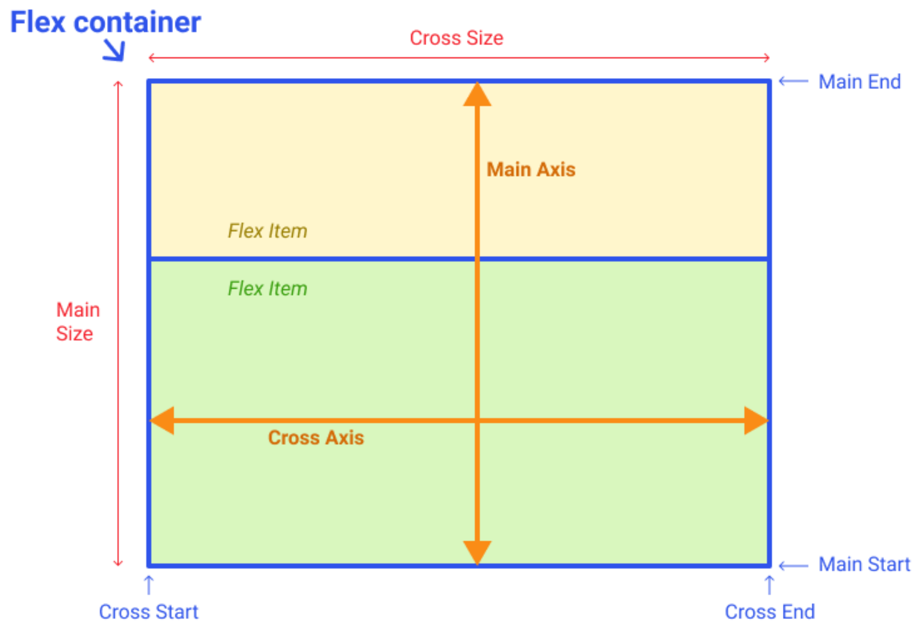
Flex container with `flex-direction = row-reverse`

The Flexible Box and `flex-direction`



Flex container with `flex-direction = column`

The Flexible Box and `flex-direction`



Flex container with `flex-direction = column-reverse`

Creating the Flex Container

Use the `display` CSS property to create a Flex container, using one of the following two values:

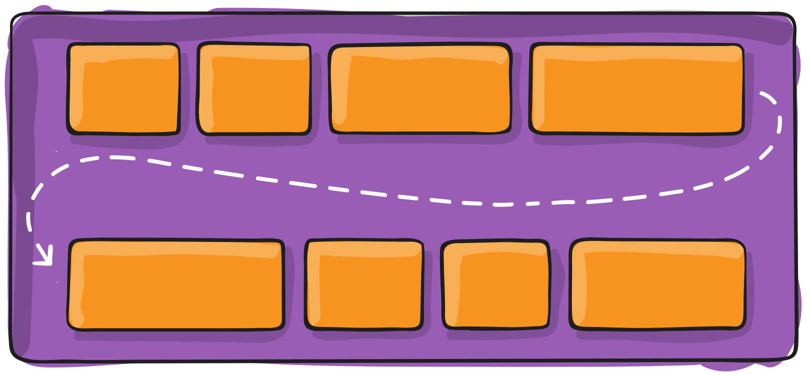
`display: flex`

Creates flex container with 100% width.

`display: inline-flex`

Creates flex container with width based on children.

`flex-wrap`



This property allows you to decide if and how to wrap your flex items to the next line.

Options:

`nowrap` (default)

Fits everything on one line.

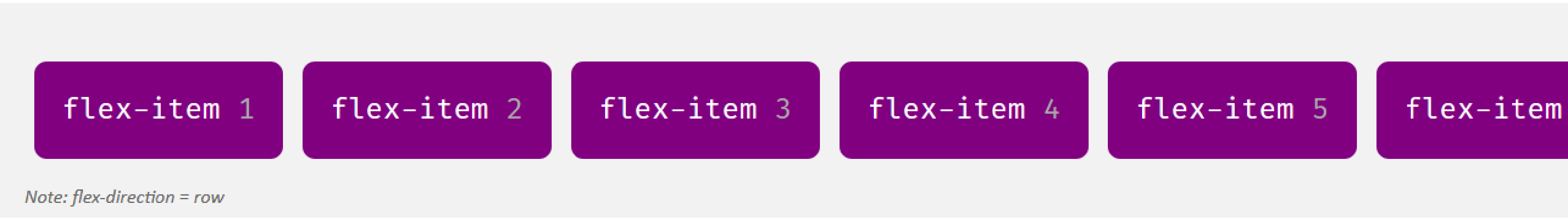
`wrap`

Wraps along cross-axis.

`wrap-reverse`

Wraps, but switches cross-axis.

flex-wrap



nowrap (default)

Fits everything on one line.

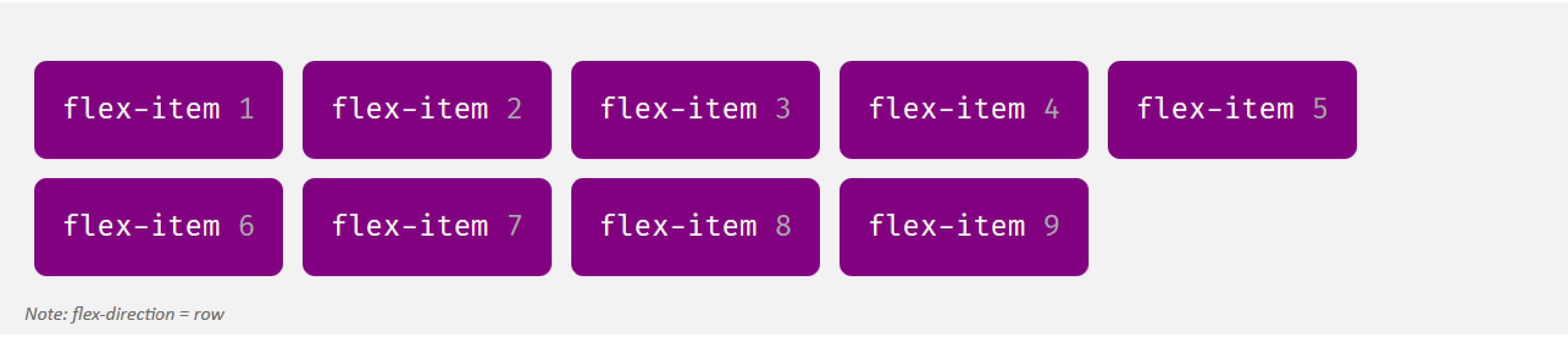
`wrap`

Wraps along cross-axis.

`wrap-reverse`

Wraps, but switches cross-axis.

flex-wrap



`nowrap` (default)

Fits everything on one line.

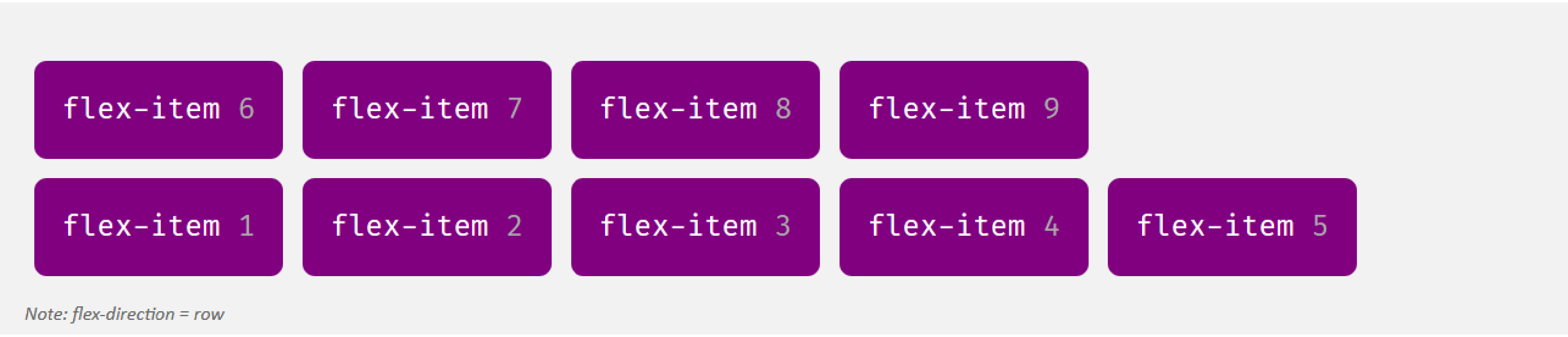
`wrap`

Wraps along cross-axis.

`wrap-reverse`

Wraps, but switches cross-axis.

flex-wrap



`nowrap` (default)

Fits everything on one line.

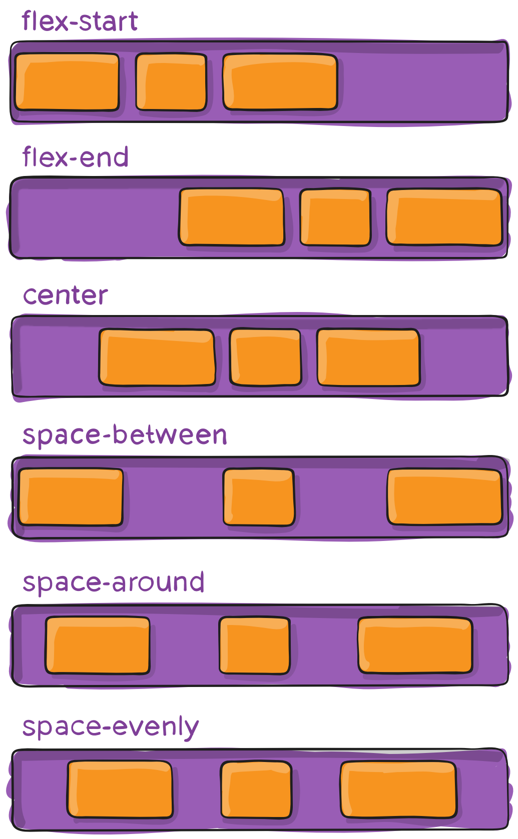
`wrap`

Wraps along cross-axis.

`wrap-reverse`

Wraps, but switches cross-axis.

`justify-content`



This property allows you to manage spacing for flex items along the **main axis** (except start and end)

Options:

`flex-start` (default)

`flex-end`

`start`

`end`

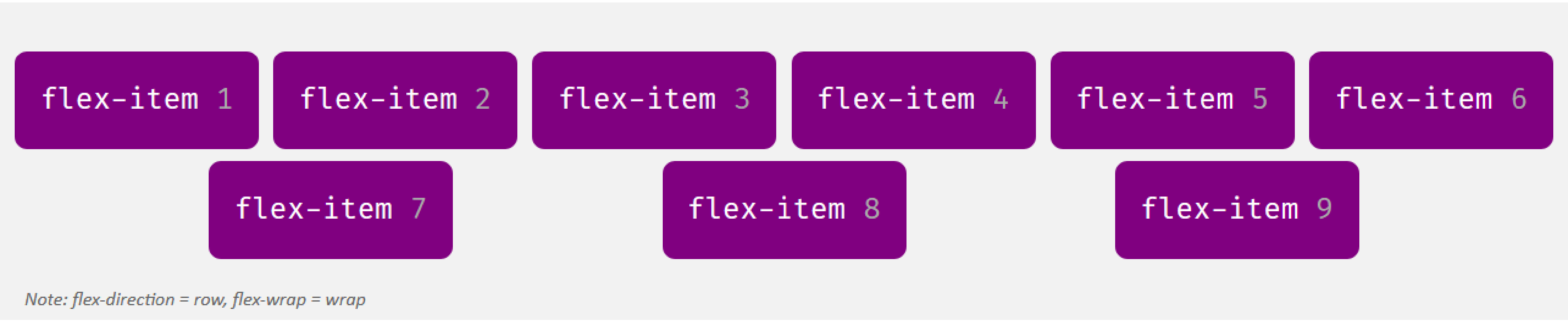
`center`

`space-between`

`space-around`

`space-evenly`

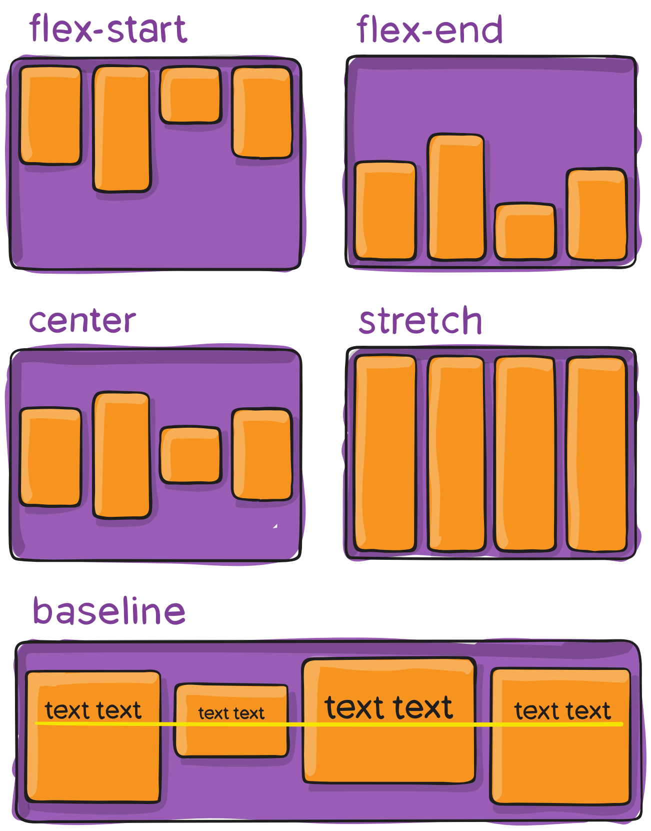
`justify-content`



Important!

When new rows or columns are started, they are treated as separate entities by flexbox. In this example, you can see that the spacing of the second row is much different than the first, although they are in one container with `justify-content: space-evenly`

align-items



This property allows you to manage spacing for flex items along the **cross axis**

Options:

`flex-start`

`flex-end`

`start`

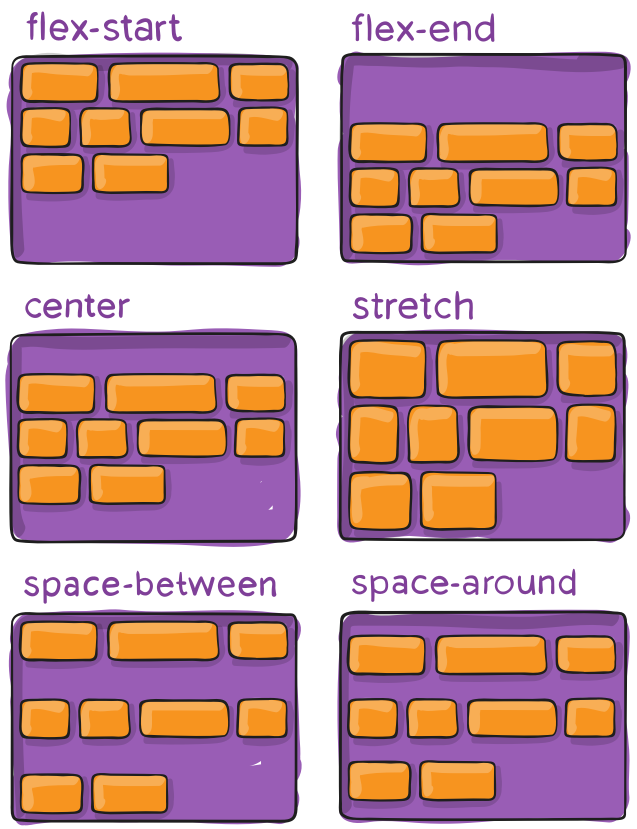
`end`

`center`

`stretch` (default)

`baseline`

align-content



This property allows you to manage spacing for rows/columns of flex items along the **cross axis**

Options:

`flex-start`

`flex-end`

`start`

`end`

`center`

`stretch` (default)

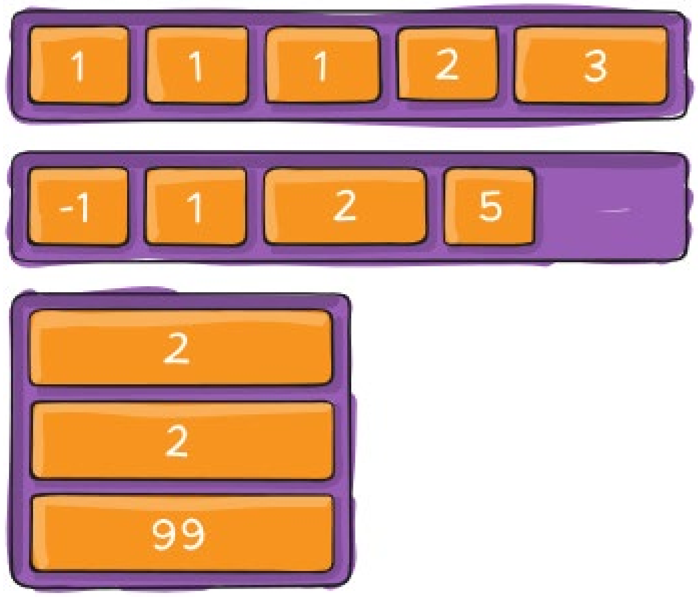
`space-between`

`space-around`

order

This property on the **flex item** allows you to reorder the items displayed in a **flex container** using any arbitrary numbers.

The lowest numbers are displayed at the **flex start**.



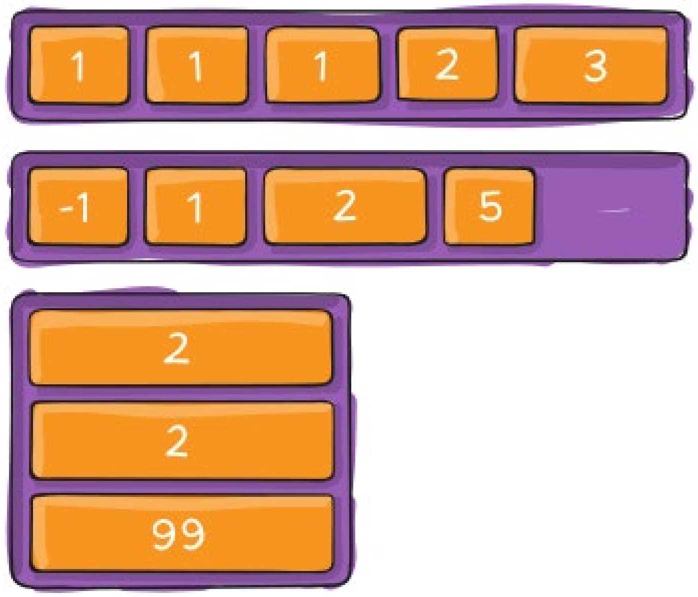
`order`

This property on the **flex item** allows you to reorder the items displayed in a **flex container** using any arbitrary numbers.

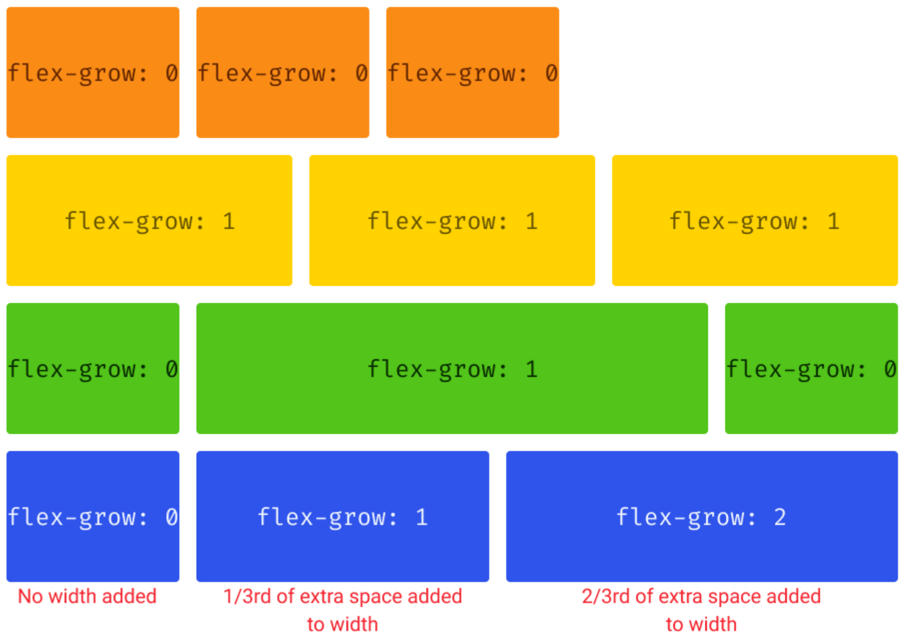
The lowest numbers are displayed at the **flex start**.

Important

This property only changes **visual** order and will not change the order for screen-readers and tabbing. Therefore, it should only be used for visual layout. If content and accessibility are factors, avoid using the `order` property.



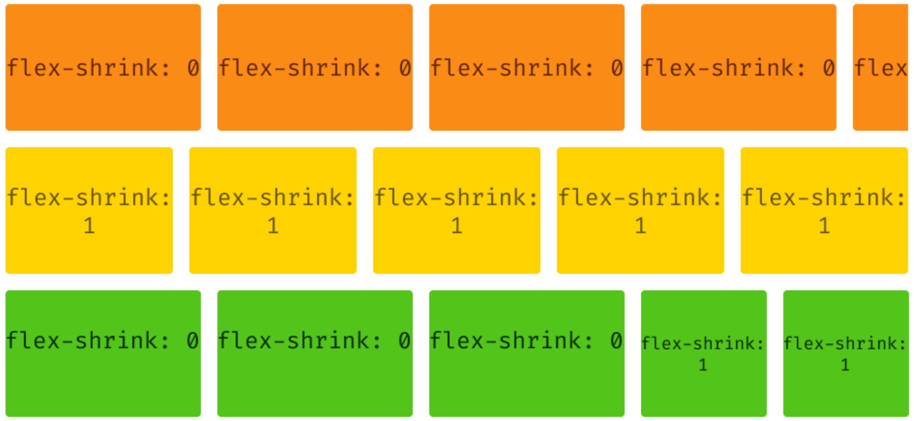
flex-grow



This property on the **flex item** affects items within rows (or columns) which have remaining space after calculating the size of the items.

The flex-grow property takes the remaining space and adds size to each flex item by the factor of the number assigned to that item.

flex-shrink



This property on the **flex item** affects items within rows (or columns) which do not wrap and have a collection of items that are larger than the container.

The flex-shrink property takes the overflow width and shrinks each flex item by the factor of the number assigned to that item.

Other Flex Properties

Flex Container

`flex-flow`

`flex-direction` and `flex-wrap` combined.

Flex Item

`flex-basis`

The initial size of the item, before extra space is configured. Default is “auto”.

`flex`

`flex-grow`, `flex-shrink`, and `flex-basis` combined.

`align-self`

Align single item along cross-axis.

Method 2: CSS Grid

The most direct way of changing layout using CSS.

Murach ch. 10

CSS Grid: Objectives

At the end of today, you will be able to:

- Use CSS Grid to lay out a simple, mobile responsive web page
- Describe the use of CSS Grid
- Describe key terms: grid, grid track, grid line, grid cell, grid area, and grid item
- Explain these ways to define the grid areas for the elements of a page: numbered lines, template areas, and the 12-column grid concept

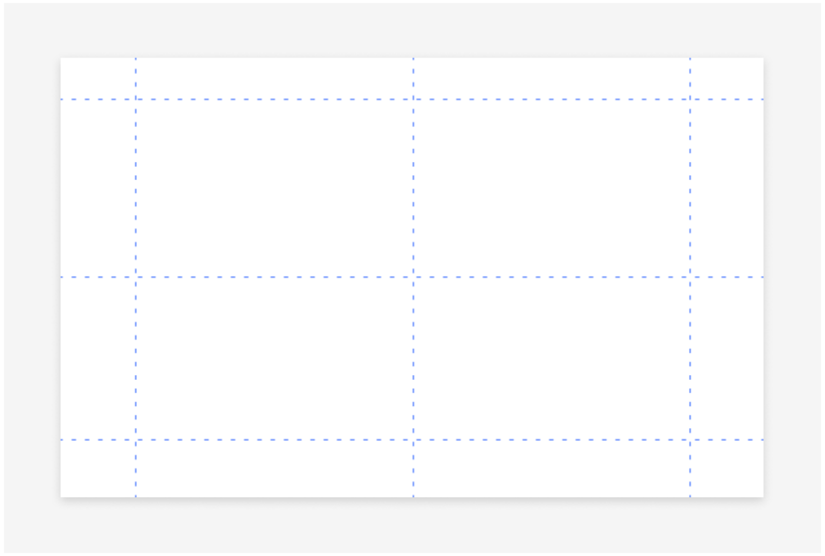
What is Grid?

CSS Grid (aka Grid), is a CSS module which…

> "…is the most powerful layout system available in CSS. It is a two-dimensional system, meaning it can handle both columns and rows

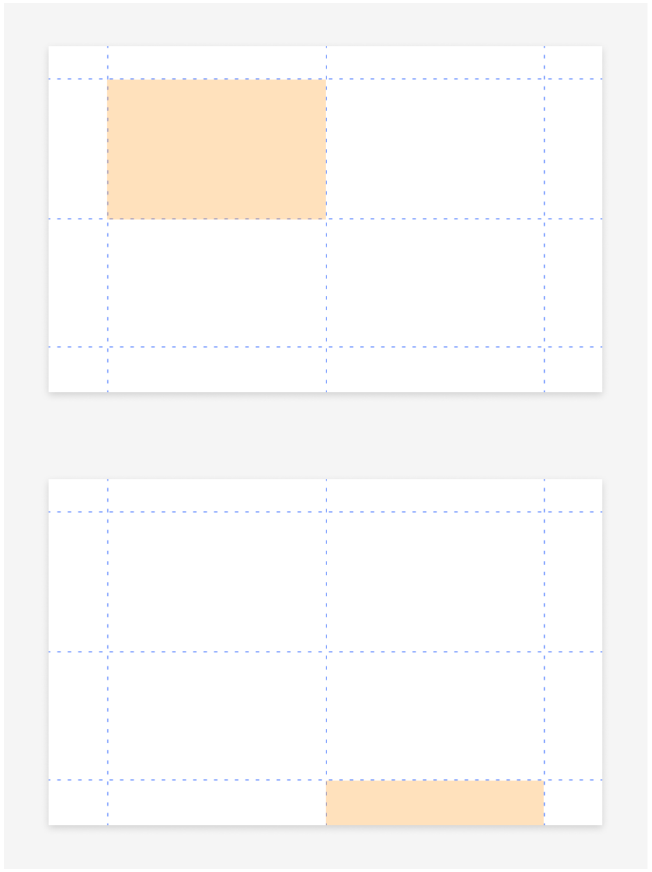
-A Complete Guide to Grid | CSS Tricks

Grid Basics – Grid Lines



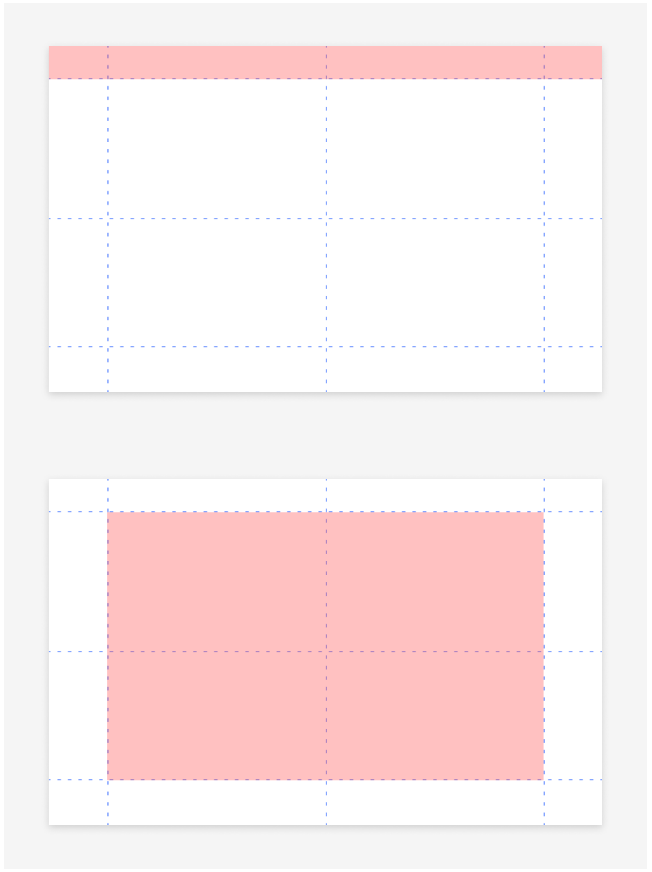
Grid lines are the lines before and after each part of the grid, as defined by you! (we'll get there).

Grid Basics – Grid Cells



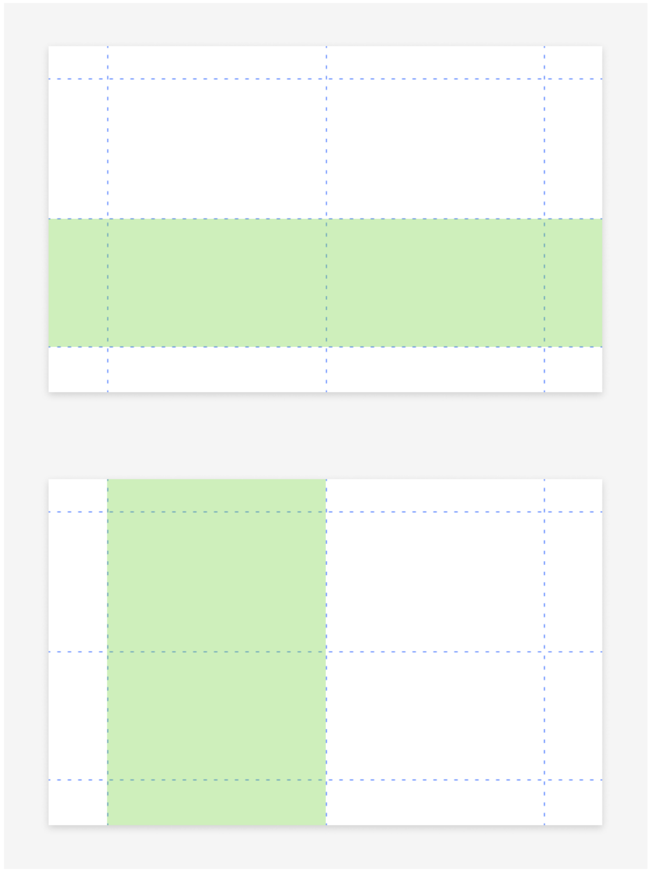
Grid cells are the smallest unit of the grid, taking up the space between 4 grid lines, without crossing any extra grid lines.

Grid Basics – Grid Areas



Grid areas are any areas which are surrounded by 4 grid lines.

Grid Basics – Grid Tracks



Grid tracks are the rows and columns created by the grid.

Creating the Grid Container

Use the `display` CSS property to create a Grid container, using one of the following two values:

`display: grid`

Creates grid container with 100% width.

`display: inline-grid`

Creates grid container with width based on children.

Creating Rows and Columns

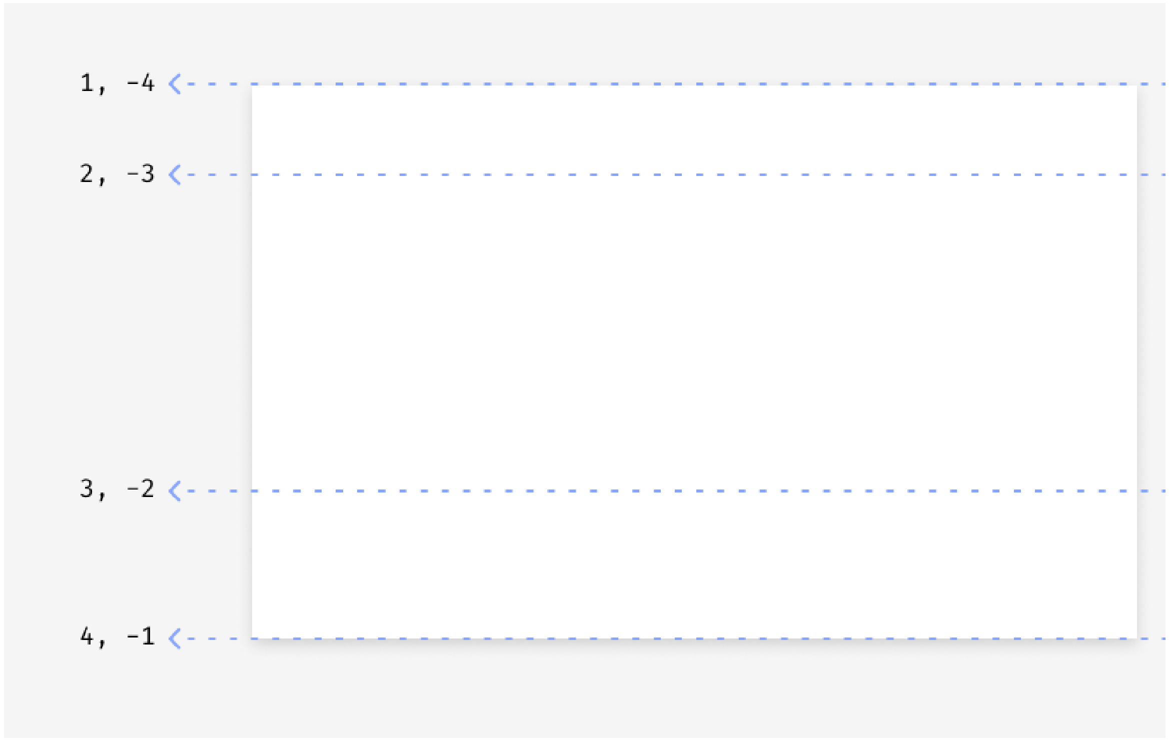
`grid-template-rows` allows you to define:

- The height of each grid track (row)
- The name of each horizontal grid line (optional)

Creating Rows and Columns

Examples:

```
grid-template-rows: 48px auto 100px;
```

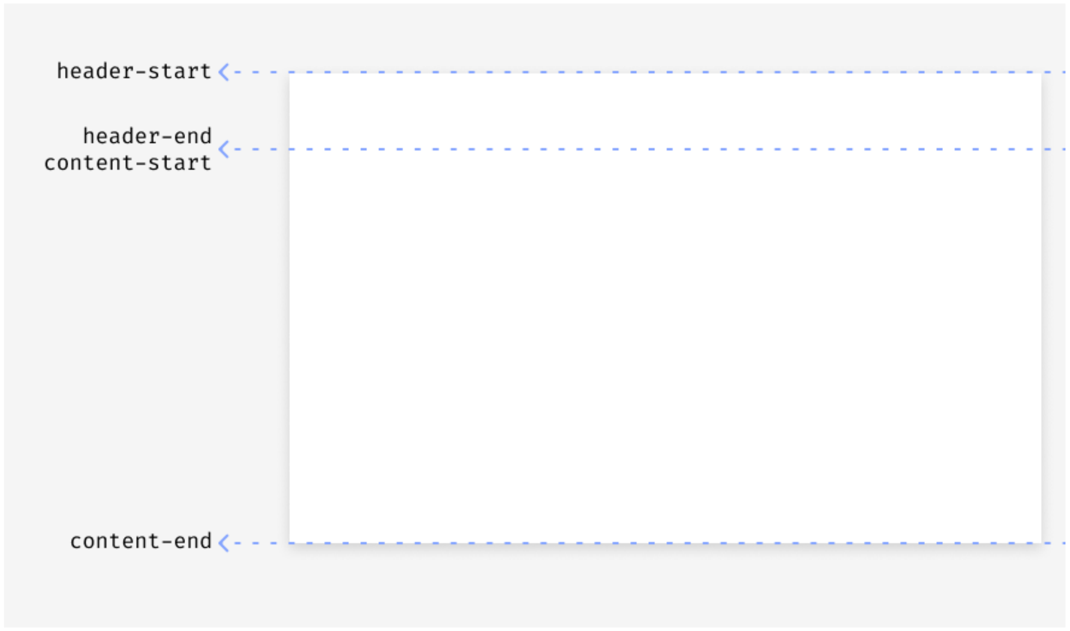


Creating Rows and Columns

Examples:

```
grid-template-rows: [header-start] 48px [header-end content-start] auto [content-end]
```

Grid lines can have multiple names. In this example, "header-end" and "content-start" are the same line. The numbers still work as well.



Creating Rows and Columns

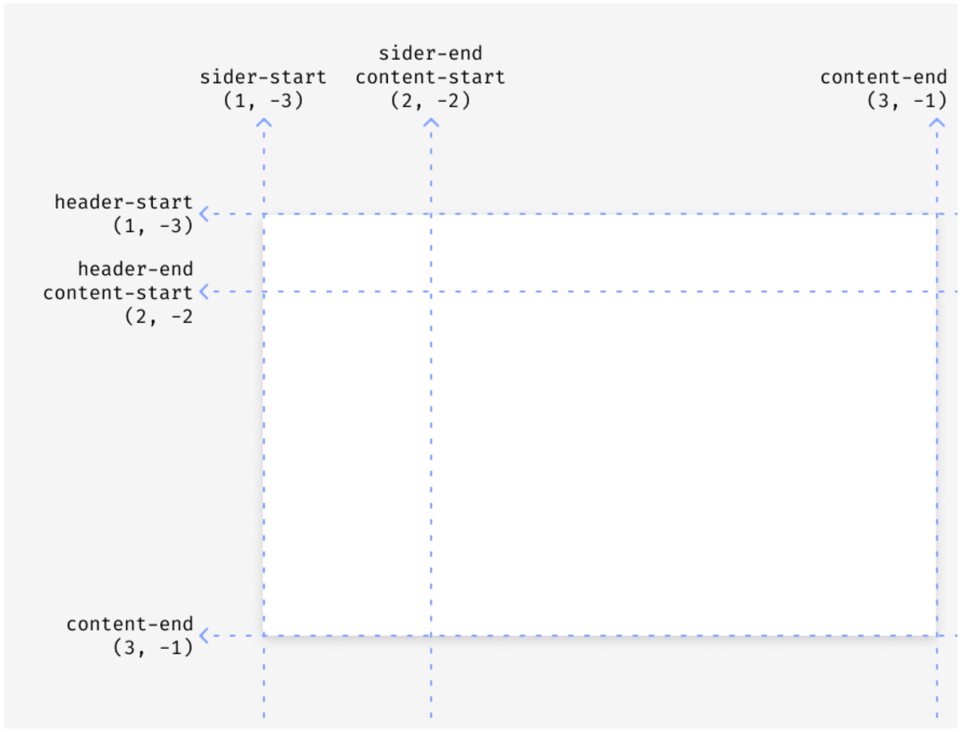
`grid-template-columns` allows you to define:

- The width of each grid track (column)
- The name of each vertical grid line (optional)

Creating Rows and Columns

Example:

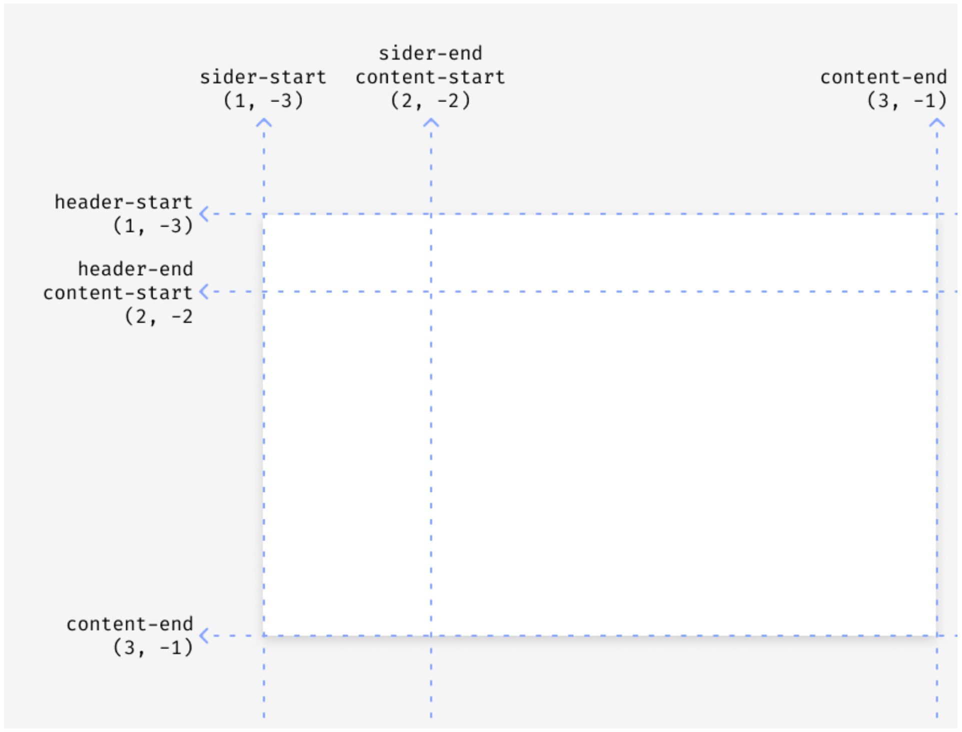
```
grid-template-rows: [header-start] 48px [header-end content-start] auto [content-end]
grid-template-columns: [sider-start] 200px [sider-end content-start] auto [content-end]
```



Creating Rows and Columns

Example:

```
grid-template: [header-start] 48px [header-end content-start] auto [content-end]
               / [sider-start] 200px [sider-end content-start] auto [content-end]
```



Defining Grid Areas

Now that we have our grid lines, grid tracks, and grid cells set up, we just need to define the grid areas (which house all of our content).

This can be done either on the individual item level, or by using the `grid-template-areas` property on the **grid container**.

Defining Grid Areas – Individual Items

To place an individual item into a specific **grid area**, you must define the row-start, row-end, column-start, and column-end.

Method 1:

```
header {
  grid-row-start: header-start;
  grid-row-end: header-end;
  grid-column-start: sider-start;
  grid-column-end: content-end;
}
```

Defining Grid Areas – Individual Items

To place an individual item into a specific **grid area**, you must define the row-start, row-end, column-start, and column-end.

Method 2:

```
header {
  grid-row: header-start / header-end;
  grid-column: sider-start / content-end;
}
```

Defining Grid Areas – Individual Items

To place an individual item into a specific **grid area**, you must define the row-start, row-end, column-start, and column-end.

Method 3:

```
header {
  grid-area: header-start / sider-start  / header-end / content-end;
         /*  row-start    / column-start /  row-end   / column-end   */
}
```

Defining Grid Areas – Grid Container

To set all of the **grid areas** in a single **grid container**, you must first name the grid areas on the individual items.

Add to children of grid container:

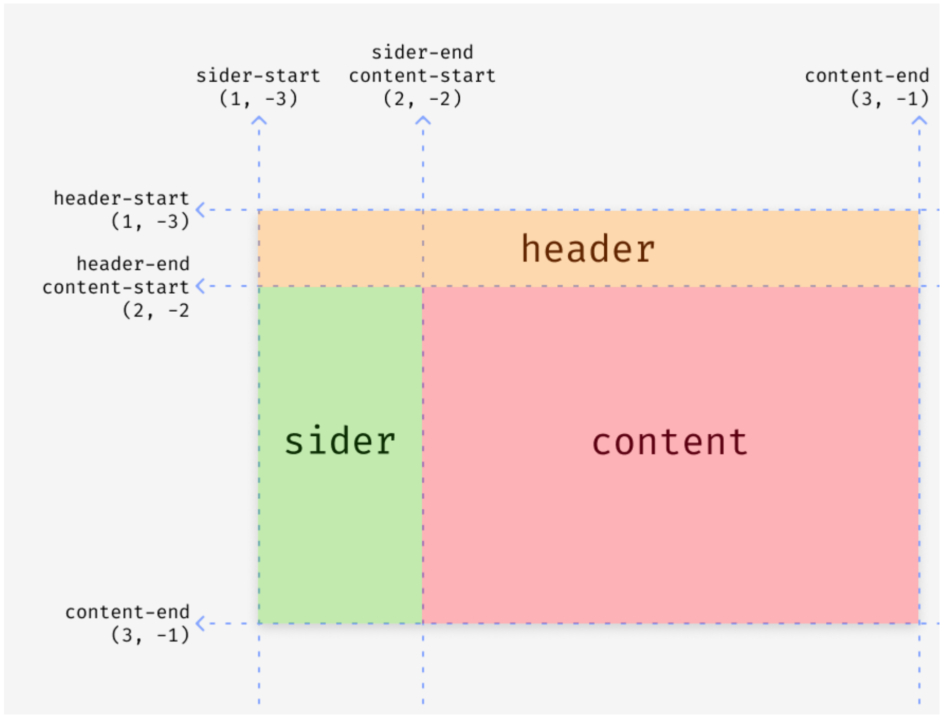
```
header { grid-area: header; }
main   { grid-area: content; }
aside  { grid-area: sider; }
```

Note: This uses the same `grid-area` property as in the previous example, however in this case it simply gives a name to the grid area. The grid lines and cells that are covered are defined on the grid container (parent).

Defining Grid Areas – Grid Container

Next, use those names to assign the items to grid areas using the `grid-template-areas` property on the **grid container**.

```
.container {
  grid-template-areas:
    “header   header”
    “sider    content”;
}
```



Defining Grid Areas – Grid Container

We can also include our grid-template-row and grid-template-column options in one handy syntax:

```
.container {
  grid-template:
    [header-start]  “header   header” 48px [header-end] /* Row 1 */
    [content-start] “sider    content” auto [content-end] /* Row 2 */
    / [sider-start] 200px [sider-end content-start] auto [content-end];
 /* / grid-template-column values */
}
```

Defining Grid Areas – Grid Container

We can also include our grid-template-row and grid-template-column options in one handy syntax:

```
.container {
  grid-template:
    [header-start]  “header   header” 48px [header-end] /* Row 1 */
    [content-start] “sider    content” auto [content-end] /* Row 2 */
    / [sider-start] 200px [sider-end content-start] auto [content-end];
  /* / grid-template-column values */
}
```

Defining Grid Areas – Grid Container

We can also include our grid-template-row and grid-template-column options in one handy syntax:

```
.container {
  grid-template:
    [header-start]  “header   header” 48px [header-end] /* Row 1 */
    [content-start] “sider    content” auto [content-end] /* Row 2 */
    / [sider-start] 200px [sider-end content-start] auto [content-end];
 /* / grid-template-column values */
}
```

Defining Grid Areas – Grid Container

We can also include our grid-template-row and grid-template-column options in one handy syntax:

```
.container {
  grid-template:
    [header-start]  “header   header” 48px [header-end] /* Row 1 */
    [content-start] “sider    content” auto [content-end] /* Row 2 */
    / [sider-start] 200px [sider-end content-start] auto [content-end];
 /* / grid-template-column values */
}
```

Defining Grid Areas – Grid Container

We can also include our grid-template-row and grid-template-column options in one handy syntax:

```
.container {
  grid-template:
    [header-start]  “header   header” 48px [header-end] /* Row 1 */
    [content-start] “sider    content” auto [content-end] /* Row 2 */
    / [sider-start] 200px [sider-end content-start] auto [content-end];
 /* / grid-template-column values */
}
```

Defining Grid Areas – Grid Container

We can also include our grid-template-row and grid-template-column options in one handy syntax:

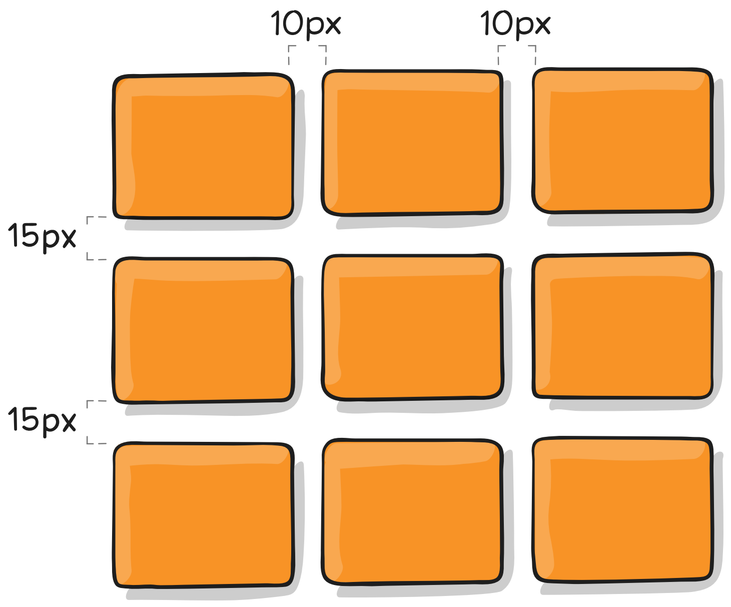
```
.container {
  grid-template:
    [header-start]  “header    header” 48px [header-end] /* Row 1 */
    [content-start] “sider     content” auto [content-end] /* Row 2 */
    / [sider-start] 200px [sider-end content-start] auto [content-end];
 /* / grid-template-column values */
}
```

Defining Grid Areas – Grid Container

We can also include our grid-template-row and grid-template-column options in one handy syntax:

```
.container {
  grid-template:
    [header-start]  “header   header” 48px [header-end] /* Row 1 */
    [content-start] “sider    content” auto [content-end] /* Row 2 */
    / [sider-start] 200px [sider-end content-start] auto [content-end];
 /* / grid-template-column values */
}
```

Spacing Between Grid Areas



To add a specific amount of space between all rows or all columns, use the `row-gap` and `column-gap` properties on the parent **grid container**.

```
.container {
  row-gap: 15px;
  column-gap: 10px;
}
```

Other Grid Properties

Grid Container

`justify-items`

`align-items`

`place-items`

`justify-content`

`align-content`

`place-content`

`grid-auto-columns`

`grid-auto-rows`

Grid Container (cont)

`grid-auto-flow`

`grid`

[Get all the details here!]()

One or the other?

When to use Flexbox vs. Grid

One or the other?

The truth is, there isn't an exact formula for when to use Flexbox versus CSS Grid.

People have differing opinions, but in the end, either module can accomplish just about anything you need, in one way or another. So try both and figure out for yourself which you prefer!

With that said, here are some themes you'll find if you spend some time looking into this question...

Large, complex layouts? **Grid**

- CSS Grid is great for laying out complex screens with many different elements.
- It provides the simplest way to explicitly define how the elements will be arranged on the screen.

Focusing on content? **Flexbox**

- Flexbox is great for controlling the flow of different pieces of content, such as images, text, buttons, and even larger UI elements in groupings.

- Flexbox also provides very straight-forward ways of aligning content, both vertically and horizontally, using "justify" and "align" properties.

But honestly, the answer is **Both**

Don't limit yourself!

Using CSS Grid to create layouts and Flexbox to control the content and smaller UI elements is a great strategy.

Both tools work very nicely with each other, so the more you can leverage them together, the better.

Resources

Websites you need to become a CSS ninja.

Resources

Flexbox

- Complete Guide to Flexbox (CSS-Tricks)
- CSS Flexbox (W3)

CSS Grid

- Complete Guide to Grid (CSS-Tricks)
- CSS Grid (W3)

Other

- Flexbox vs. CSS Grid: Which Should You Use and When?
- Modern Web Layout with Flexbox and CSS Grid (Pluralsight Class)