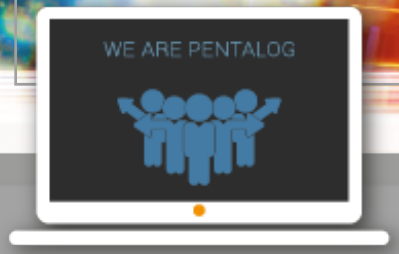
Pentalog

A FULL-STACK NEARSHORE IT GROUP SERVING COMPANIES
FROM STARTUPS TO FORTUNE 500s

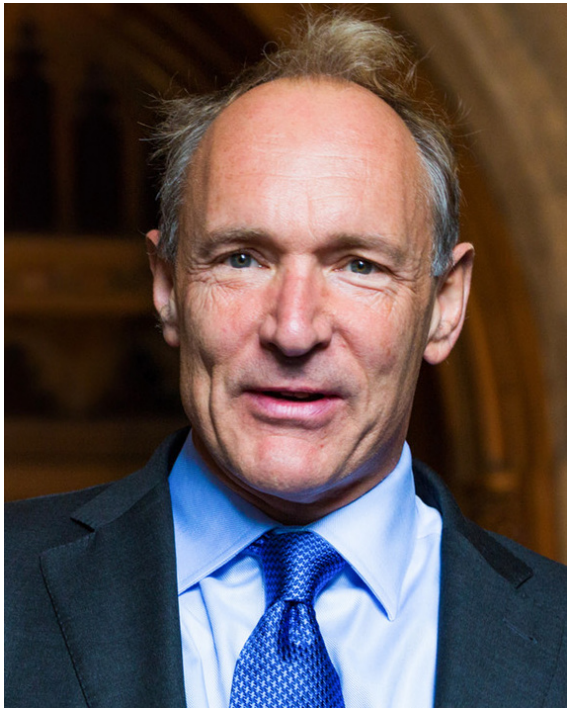
HTML5
SASS
Flex / CSS Grid

Ivan Ceban



SIR TIMOTHY JOHN BERNERS-LEE

- English engineer and computer scientist
- Inventor of the World Wide Web
- Creator of first web browser
- Implemented the first successful communication between a Hypertext Transfer Protocol (HTTP) client and server via the internet in 1989
- director of the World Wide Web Consortium (W3C)
- Was knighted by Queen Elizabeth II for his pioneering work in 2004.
- Named in *"Time"* magazine's list of the 100 Most Important People of the 20th century.
- Received the 2016 Turing Award



HTML HISTORY

HTML (HyperText Markup Language), is not a programming language, but rather a language that identifies the meaning, purpose, and structure of text within a document.

HTMLs roots go back to at least 1980, with Tim Berners-Lee's project ENQUIRE. And actually, the concept of hypertext goes back even further than that. The concept first appeared in the early 1940s, and was named and demonstrated in the 1960s.

In 1989, Lee proposed a new hypertext system based on the ideas of ENQUIRE. This became the first version of what we now call HTML.

Since then, the language has been in constant development. The specification is managed by the World Wide Web Consortium (Berners-Lee is still the director), and the Web Hypertext Application Technology Working Group. (So, if you don't like HTML5, these are the people to blame.)

HTML VERSIONING

HTML 1.0: This was the barebones version of HTML and the very first release of the language in 1990.

HTML 2.0: This version was introduced in 1995. It gradually evolved, allowing extra capabilities including form-based file upload, tables, client-side image maps and internationalization.

HTML 3.2: In an attempt to ensure development of standards for the World Wide Web, the World Wide Web Consortium (W3C) was founded by Tim Berners-Lee in 1994. By 1997, they published HTML 3.2.

HTML 4.0: Later in 1997, the W3C released HTML 4.0 — a version that adopted many browser-specific element types and attributes.
HTML 4.0 was later reissued with minor edits in 1998.

HTML 4.01: In December 1999, HTML 4.01 was released.

XHTML: The specifications were introduced in 2000 and it was recommended to be used as the joint-standard with HTML 4.01. It incorporated XML to ensure code is properly written and to ensure interoperability between programming languages.

HTML5: The W3C published HTML5 as a recommendation in October 2014 and later released HTML 5.1 in November 2016.

HTML5

HTML5 is the latest specification of the HTML language, and represented a major break with previous markup practices. The purpose of the profound changes to the language was to standardize the many new ways in which developers were using it, as well as to encourage a single set of best practices with regards to web development.

Most of the individual changes are a result of larger objectives in the design of the language. These objectives primarily include:

- Encouraging semantic (meaningful) markup
- Separating design from content
- Promoting accessibility and design responsiveness
- Reducing the overlap between HTML, CSS, and JavaScript
- Supporting rich media experiences while eliminating the need for plugins such as Flash or Java

Getting a handle on HTML5 isn't just about learning which CSS features replace old HTML features. If you want to get an intuitive sense of HTML5, it is best to understand how these objectives affected the development of the language.

WHY SHOULD I USE HTML5?

The most straight-forward answer to that question is simply that it is the current, “right” version of the language.

- Easier to write
- Easier to maintain
- Easier to redesign
- Better for Search Engine Optimization
- Better for the blind and visually impaired
- Better for content aggregators and feed readers
- Better for users on mobile devices
- Better for users on slower internet connections
- Fewer chances of design breaks
- Easier to add media
- Easier to create interactive applications
- Deprecated features will likely stop being supported at some point, breaking your page

CSS PREPROCESSORS

A **CSS preprocessor** is a program that lets you generate CSS from the preprocessor's own unique syntax. There are many CSS preprocessors to choose from, however most CSS preprocessors will add some features that don't exist in pure CSS, such as mixin, nesting selector, inheritance selector, and so on. These features make the CSS structure more readable and easier to maintain.

To use a CSS preprocessor, you must install a CSS compiler on your web server.

Here are a few of the most popular CSS preprocessors:

- Sass
- Less
- Stylus
- PostCSS

10 REASONS TO USE PREPROCESSORS

1. $variables

```
$color-green: #91ea42;
.text-green {
  color: $color-green;
}
.button-green {
  background: $color-green;
}
```

2. @imports

```
@import 'views/_home';
@import 'views/_about';
@import 'views/_gallery';
```

3. @mixins

```
// mixin
@mixin transform($property: scale3d(2, 1.5, 0.5)) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}
// include mixin
div {
  @include transform();
}
```

4. @extend

```
.button {
  display: inline-block;
  border-radius: 3px;
  padding: 16px;
}
.button-green {
  @extend .button;
  color: #ffffff;
  background: #91ea42;
}
```

5. Math!

```
$nav-height: 60px;
body {
  padding-top: $nav-height + 40px;
}
.one-third {
  width: (100% / 3);
}
```

10 REASONS TO USE PREPROCESSORS

6. Loops

```
$facebook: #3b5998;
$twitter: #4099ff;
$colors: (
  "facebook": $facebook,
  "twitter": $twitter
);
@each $key, $color in $colors {
  .color-#{$key} {
    background-color: $color;
  }
  .text-#{$key} {
    color: $color;
  }
}
```

7. Nesting

```
nav {
  ul {
    list-style: none;
    li {
      display: inline-block;
    }
  }
  a {
    display: block;
    text-decoration: none;
    padding: 0.5rem;
  }
}
```

8. Pre-built Functions

```
$color: #91ea42;
$amount: 50%;
div {
  background-color: lighten($color, $amount);
}
```

9. Frameworks & Libraries

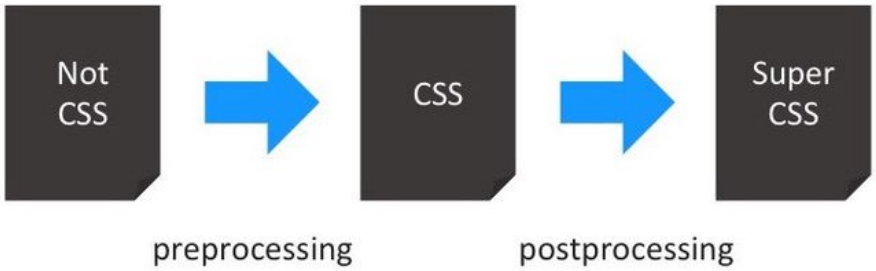
Bootstrap and Foundation (ones of the most popular frameworks) are built on SASS

10. Code Optimization

- Fewer repetitions.
- Easier debugging.
- Easier to read and make sense of someone else's code.

CSS POST-PROCESSORS

Post-processing happens after you already produced the plain CSS, and want to **extend it further** through automation.



Autoprefixer, is one of the most popular **local post-processing tools**. It adds all necessary browser prefixes to plain css. You don't need to write prefixes, Autoprefixer does it for you.

You write this: => (Run Autoprefixer) => And get this:

```
.example {
  animation-name: slidein;
}
```

```
.example {
  -webkit-animation-name: slidein;
  -o-animation-name: slidein;
  -ms-animation-name: slidein;
  -moz-animation-name: slidein;
  animation-name: slidein;
}
```

TABLE LAYOUT

The main html technology of 2 dimensional layout on websites, before CSS2 appeared.

```
<!-- Start of header container -->
<table width="100%" cellspacing="0" cellpadding="0" border="0" align="center">
    <tr>
        <td valign="top">
            <!-- Start of header -->
            <table width="600" cellspacing="0" cellpadding="20" border="0" align="c
                <tr>
                    <td style="padding-bottom: 10px">
                        <p style="font-size: 60px; font-family: Tahoma,Geneva,sans-
                        PureText email
                        </p>
                    </td>
                </tr>
            </table>
            <!-- End of header -->
        </td>
    </tr>
</table>
<!-- End of header container-->
<!-- Start of intro copy -->
<table width="600" cellspacing="0" cellpadding="0" border="0" align="center">
    <tr>
        <td valign="top" style="padding-top: 25px; padding-right: 20px;padding-bott
            <p style="font-size: 20px; font-family: Tahoma,Geneva,sans-serif; line-
            Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orc
            </p>
        </td>
    </tr>
</table>
<!-- End of intro copy -->
<!-- Start of main container -->
<table width="600" border="0" cellspacing="0" cellpadding="0" align="center">
    <tr>
        <!-- Start of Sidebar -->
        <td width="200" valign="top">
            <table width="200" border="0" cellspacing="0" cellpadding="0" align="le
                <tr>
                    <td valign="top" style="padding-top: 0;padding-right: 20px;padd
                        <h2 style="font-size: 18px; font-family: Tahoma,Geneva,sans
                        <ul style="font-size: 13px; font-family: Tahoma,Geneva,sans
                            <!--[if gte mso 9]>
                            <ul style="margin-left: 23px;list-style: disc;">
                            <![endif]-->
                            <li>Donec non tortor in feugiat</li>
                            <li>Cras aliquam massa ullamcorper</li>
                            <li>Nullam porttitor ligula</li>
```

Issues:

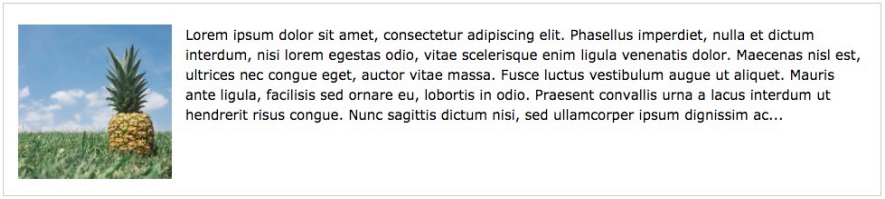
- No possibility to reorder elements
- Not responsive
- Positioning (absolute, relative, fixed) is impossible
- Too much code lines => mess
- Difficult to maintain
- Unexpected behavior
- It is outdated more then 10 years

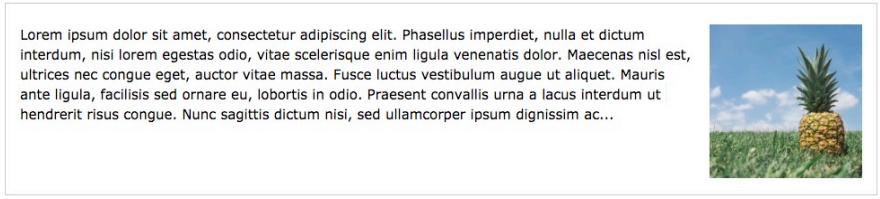
The biggest pain:

It is still required in emails layout

FLOAT

The CSS float property specifies how an element should float.





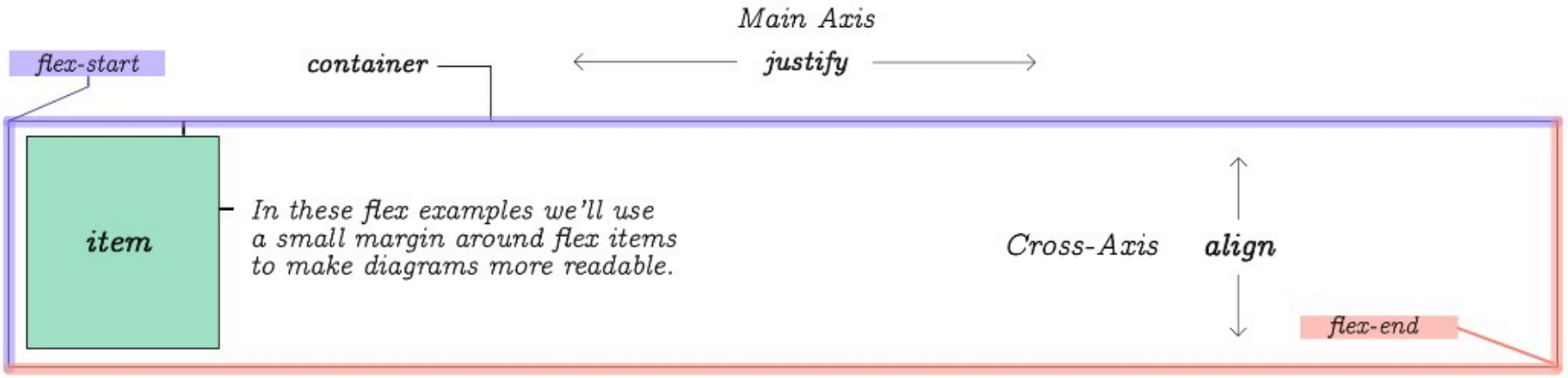

```
.box {
  float: left;
  width: 33.33%; /* three boxes (use 25% for four, and 50% for two, etc) */
  padding: 50px; /* if you want space between the images */
}
```

Issues:

- Equal height
- Dynamic space between elements
- Vertical alignment
- Responsiveness

FLEX

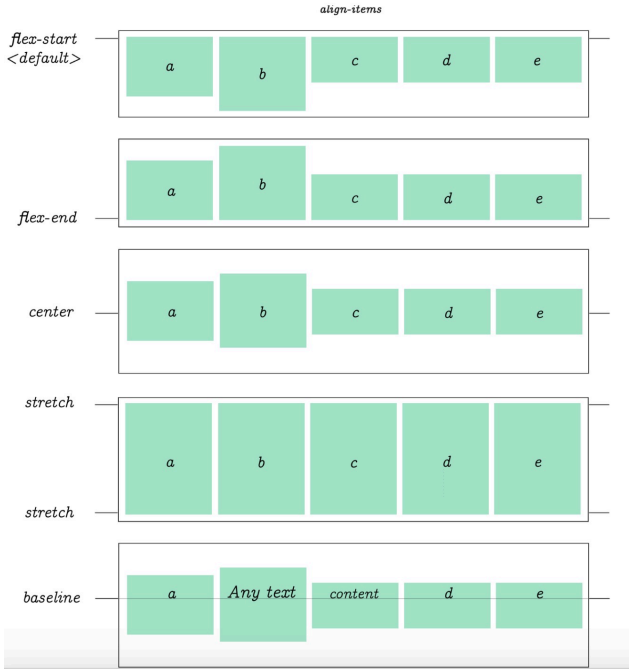
Flex is a set of rules for automatically stretching multiple columns and rows of content across parent container.
Unlike many other CSS properties, in Flex you have a main container and items nested within it. Some CSS flex properties are used only on the parent. Others only on the items.

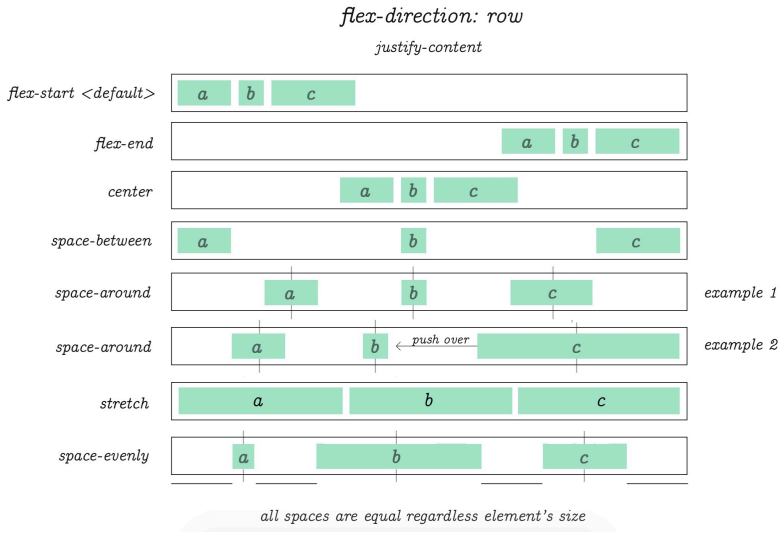


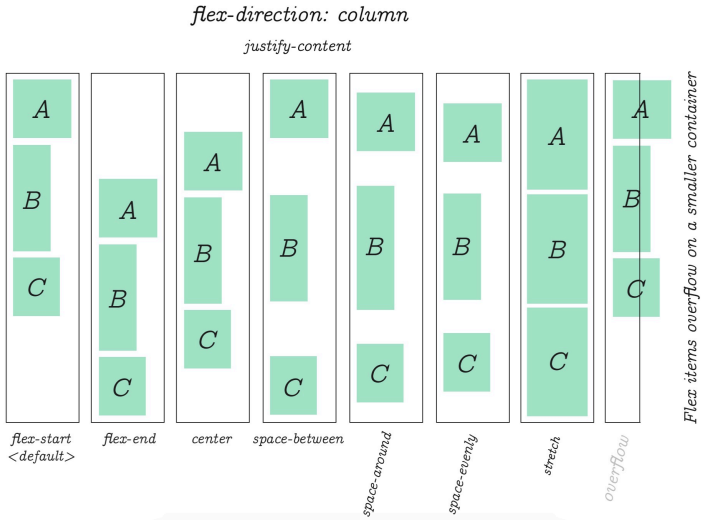
You can think of a flex element as a parent container with **display:flex**. Elements placed inside this container are called items. Each container has a **flex-start** and **flex-end** points as shown on this diagram.

FLEXBOX

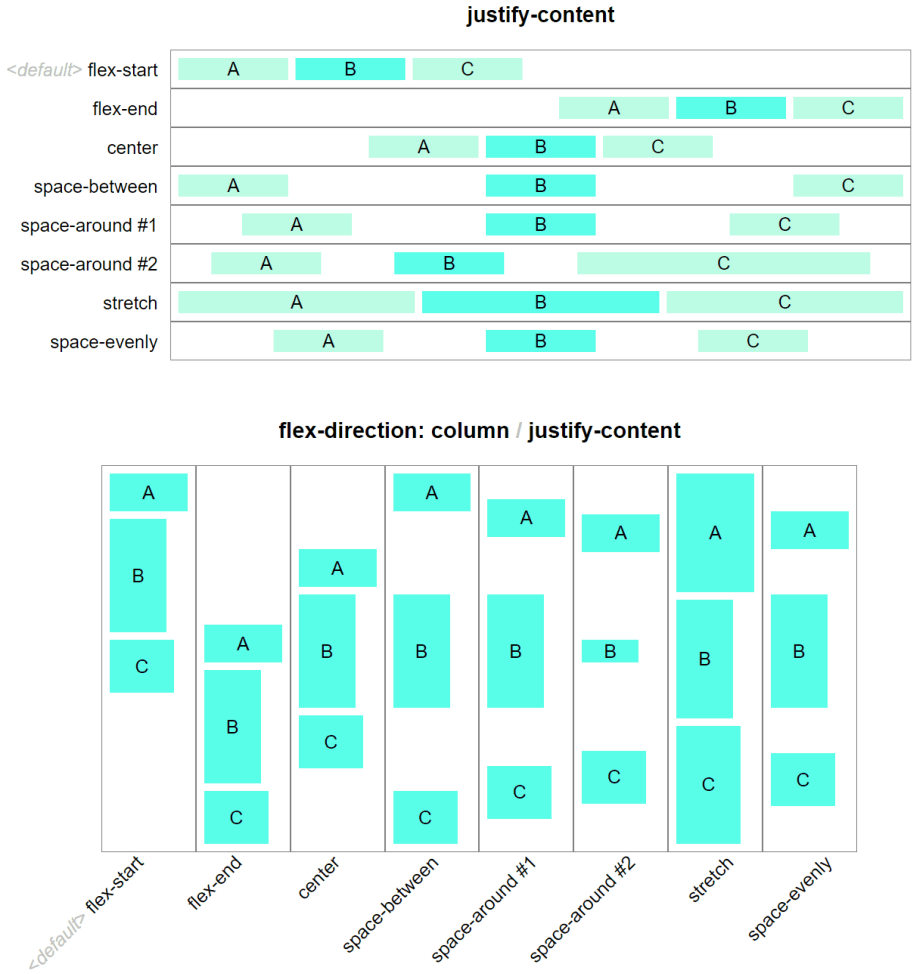
While the list of items is provided in a linear way, Flex requires you to be mindful of rows and columns. For this reason, it has two coordinate axis. The horizontal axis is referred to as Main-Axis and the vertical is the Cross-Axis.
To control the behavior of content's width and gaps between that stretch horizontally across the Main-Axis you will use ***justify*** properties. To control vertical behavior of items you will use ***align*** properties.







FLEXBOX

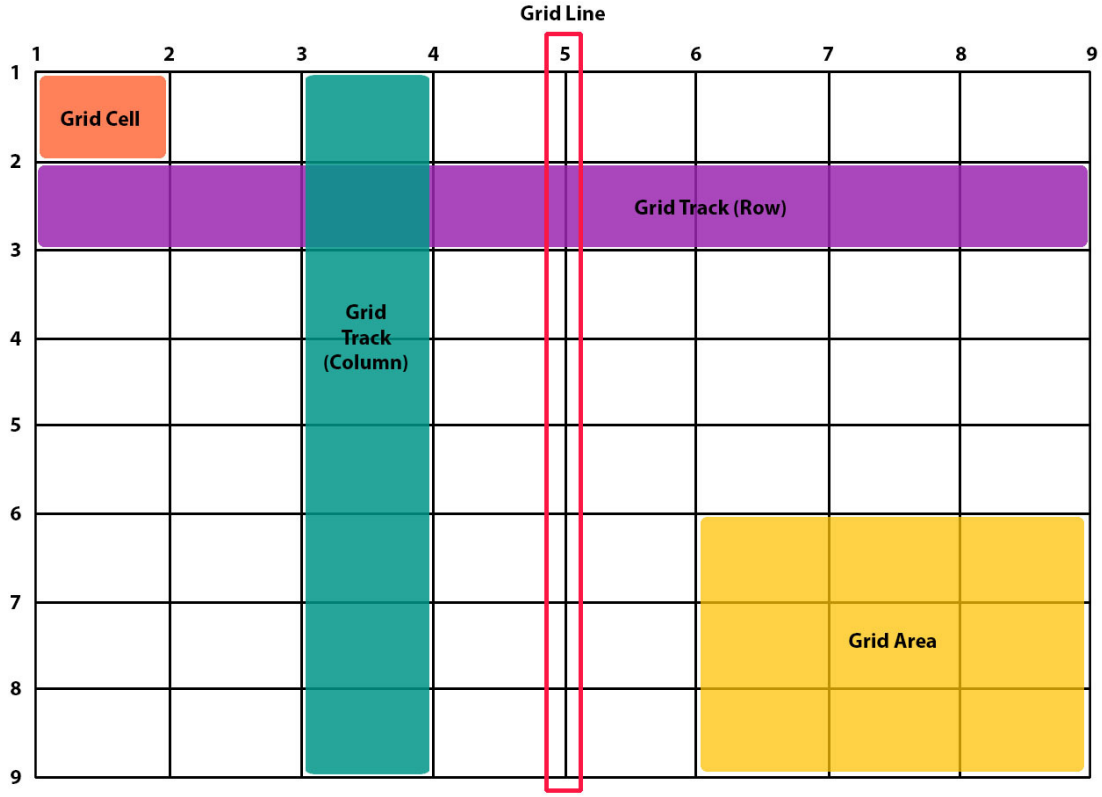


justify-content: flex-start | flex-end | center | space-between | space-around | stretch | space-evenly. In this example we're using only 3 items per row.

There is no limit on the number of items you wish to use in flex. These diagrams only demonstrate the behavior of items when one of the listed values is applied to **justify-content** property.

The same **justify-content** property is used to align items when **flex-direction** is column

CSS GRID



Grid container - the element that contains the elements of the grid.

Grid lines - the horizontal and vertical lines which divide the grid.

Grid column - the space between two adjacent vertical grid lines.

Grid row - the space between two adjacent horizontal grid lines.

Grid area - is any area of space within the grid container that is bound by four grid lines.

Grid cell - the intersection between a grid column and a grid row. It is a space bound by exactly four grid lines, which makes it the smallest available space within the grid.

Grid - the sum of all these parts

COLUMNS & ROWS

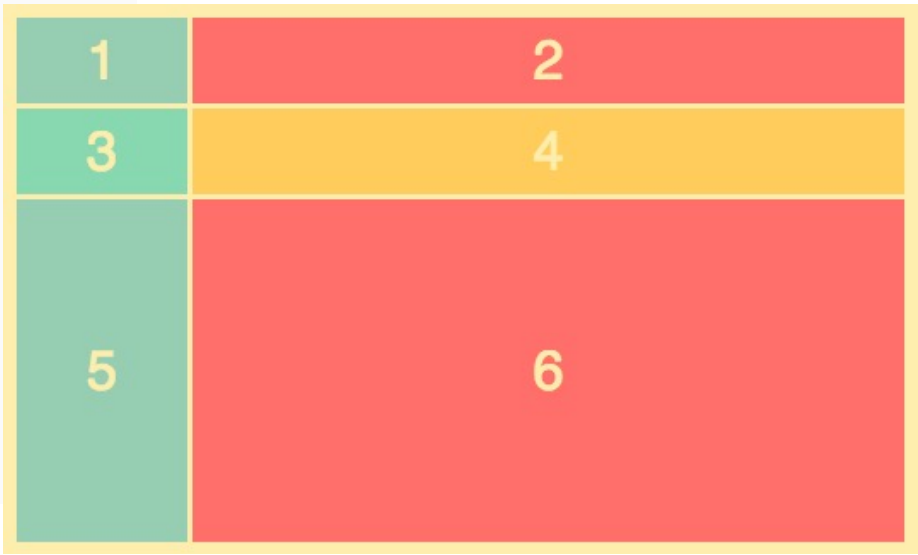
```html
<html>
    <head>
        <style>
            .container {
                display: grid;
                grid-template-columns: 100px auto;
                grid-template-rows: 50px 50px 200px;
                grid-gap: 3px;
            }
        </style>
    </head>
    <body>
        <div class="container">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
            <div>6</div>
        </div>
    </body>
</html>
```

grid-template-columns: **12px 12px 12px;**
grid-template-rows: **12px 12px 12px;**

grid-template-columns: **repeat(3, 12px);**
grid-template-rows: **repeat(3, auto);**

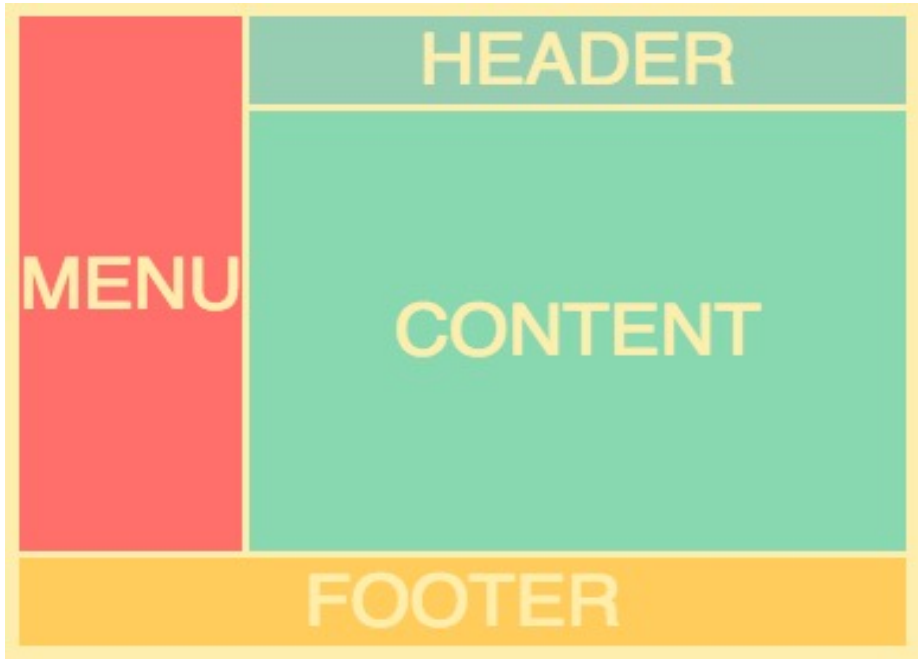
grid-template-columns: **8px auto 8px;**
grid-template-rows: **8px auto 12px;**

grid-template-columns: **22% 22% auto;**
grid-template-rows: **22% auto 22%;**



ITEMS POSITIONING

```html
<html>
    <head>
        <style>
          .container {
              display: grid;
              grid-gap: 3px;
              grid-template-columns: repeat(12, 1fr);
              grid-template-rows: 40px 200px 40px;
          }
          .header {
              grid-column: 2 / -1;
          }
          .menu {
              grid-row: 1 / 3;
          }
          .content {
              grid-column: 2 / -1;
          }
          .footer {
              grid-column: 1 / -1;
          }
        </style>
    </head>
    <body>
        <div class="container">
            <div class="header">HEADER</div>
            <div class="menu">MENU</div>
            <div class="content">CONTENT</div>
            <div class="footer">FOOTER</div>
        </div>
    </body>
</html>
```



grid-column-start: **1**;
grid-column-end: **3**;

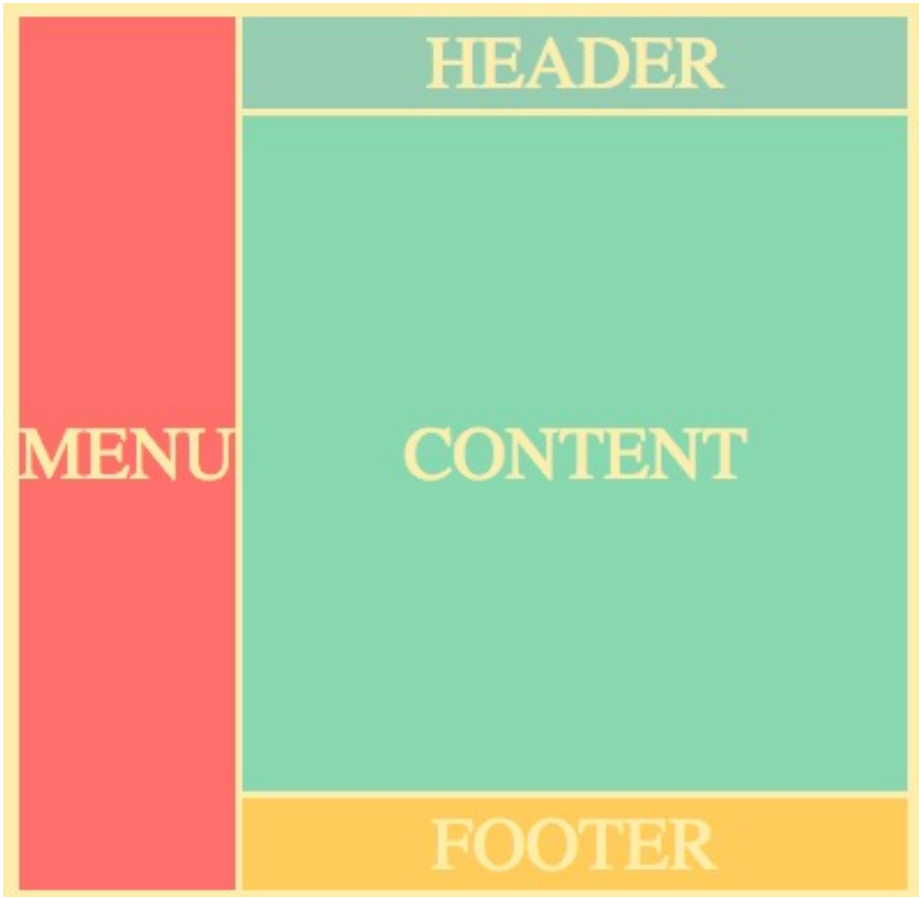
grid-column: **1 / 3**;

grid-column: **1 / span 2**;

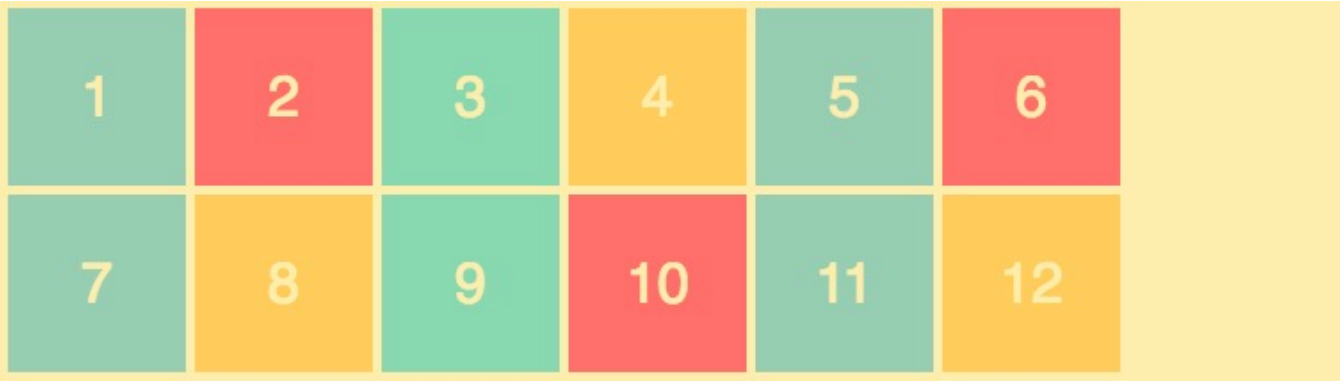
<number> | <name> | span <number> | span <name> | auto

TEMPLATE AREAS

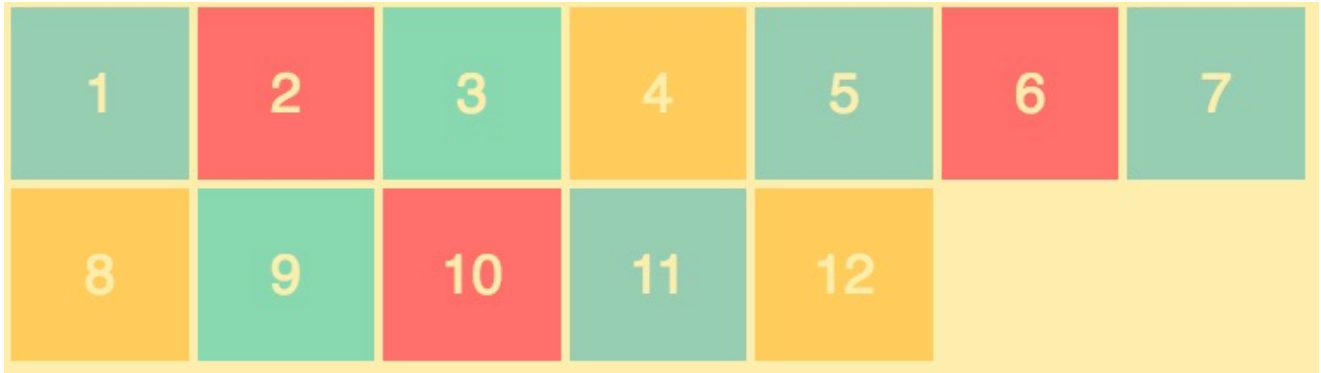
```
.container {
    height: 100%;
    display: grid;
    grid-gap: 3px;
    grid-template-columns: repeat(12, 1fr);
    grid-template-rows: 40px auto 40px;
    grid-template-areas:
        "m h h h h h h h h h h h"
        "m c c c c c c c c c c c"
        "m f f f f f f f f f f f";
}
.header {
    grid-area: h;
}
.menu {
    grid-area: m;
}
.content {
    grid-area: c;
}
.footer {
    grid-area: f;
}
```



AUTO-FIT & MINMAX

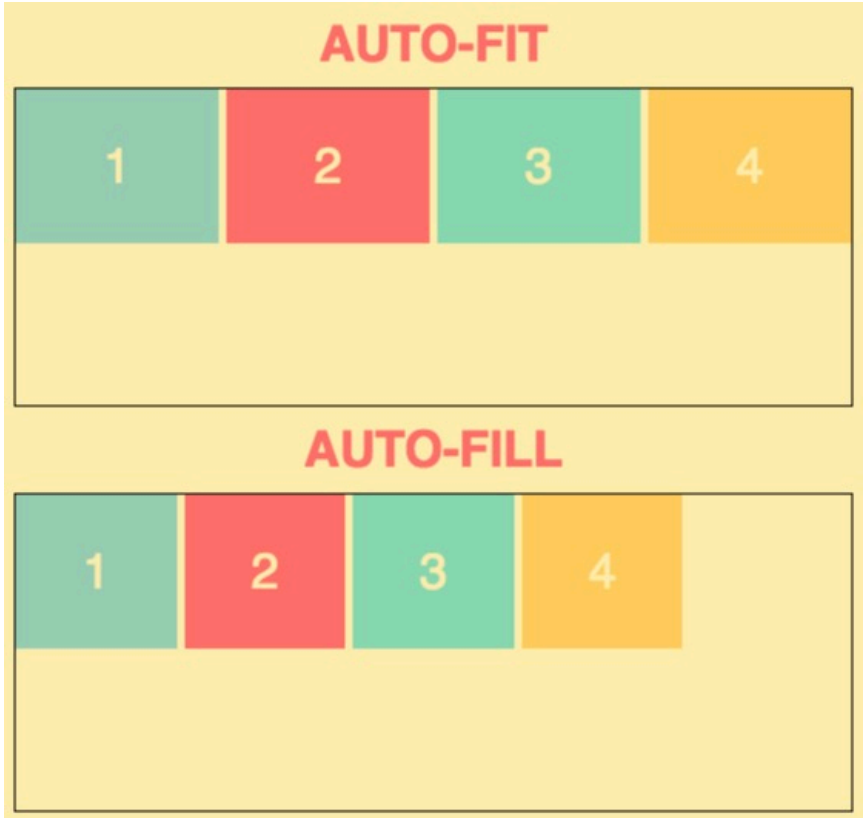



```
.container {
    display: grid;
    grid-gap: 5px;
    grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
    grid-template-rows: repeat(2, 100px);
}
```



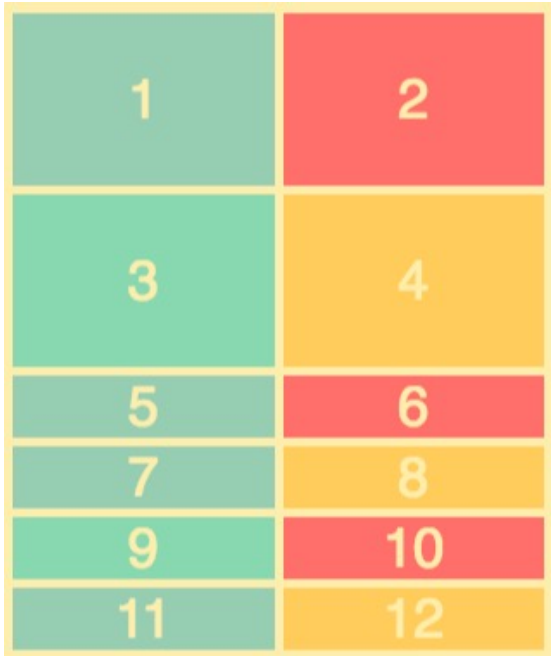
AUTO-FIT VS AUTO-FILL

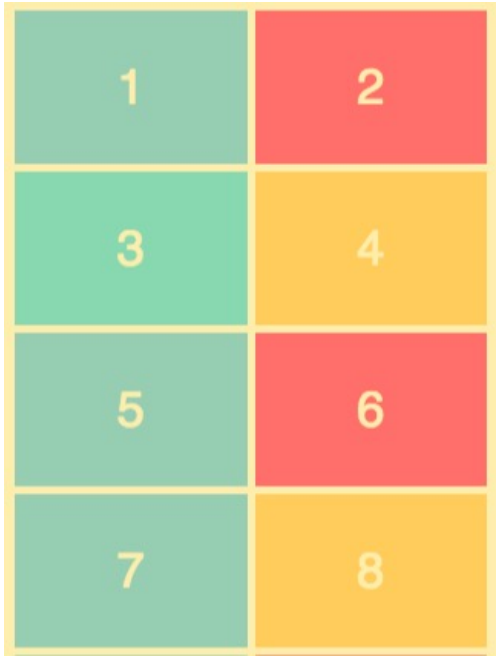
```css
.container {
    border: 1px solid black;
    display: grid;
    grid-gap: 5px;
    grid-template-columns:
        repeat(auto-fit, minmax(100px, 1fr));
    grid-template-rows: 100px 100px;
}

.container2 {
    border: 1px solid black;
    display: grid;
    grid-gap: 5px;
    grid-template-columns:
        repeat(auto-fill, minmax(100px, 1fr));
    grid-template-rows: 100px 100px;
}
```



IMPLICIT ROWS

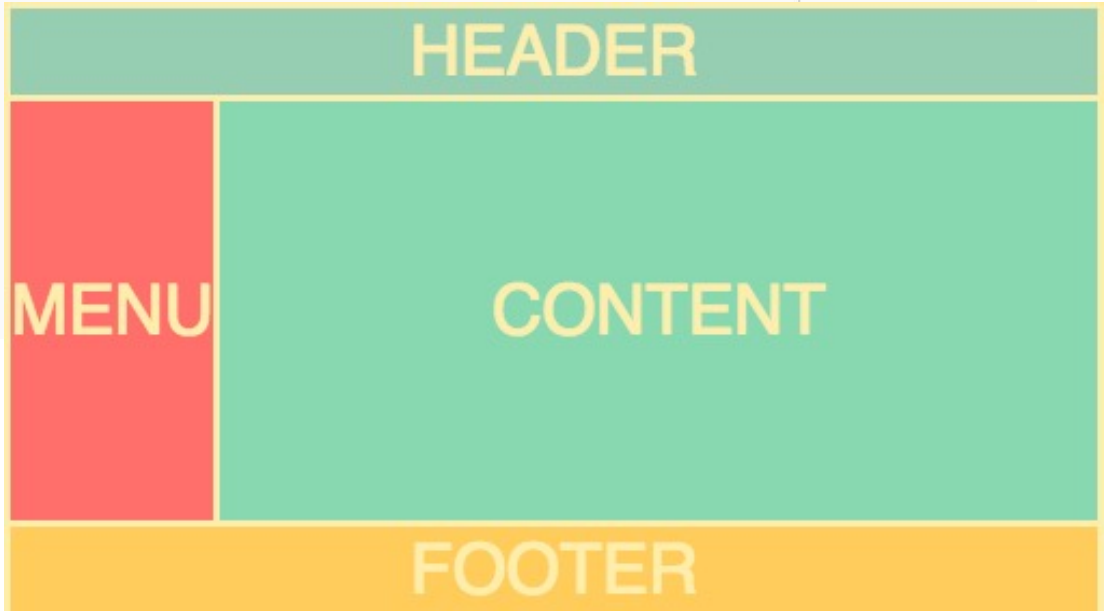
```
.container {
    display: grid;
    grid-gap: 5px;
    grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
    /* grid-template-rows: 100px 100px; */
    grid-auto-rows: 100px;
}
```





NAMED LINES

```
.container {
    height: 100%;
    display: grid;
    grid-gap: 3px;
    grid-template-columns: [main-start] 1fr [content-start] 5fr [content-end main-end];
    grid-template-rows: [main-start] 40px [content-start] auto [content-end] 40px [main-end];
}
.header {
    grid-column: main-start / main-end;
}
.menu {}
.content {
    grid-area: content;
}
.footer {
    grid-column: main;
}
```

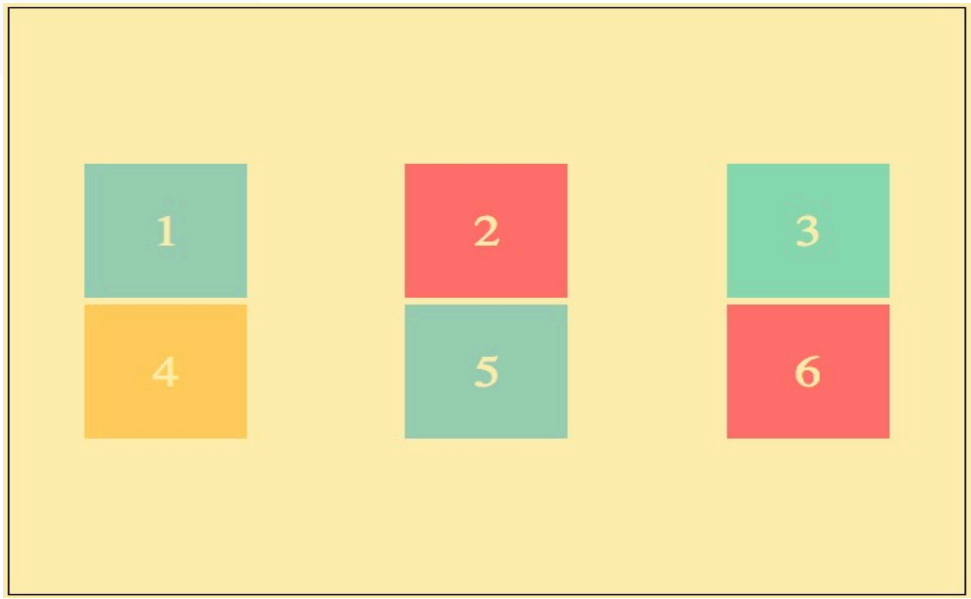


JUSTIFY & ALIGN CONTENT

```
1  .container {
2      border: 1px solid black;
3      height: 100%;
4      display: grid;
5      grid-gap: 5px;
6      grid-template-columns: repeat(3, 100px);
7      grid-template-rows: repeat(2, 100px);
8      justify-content: space-around;
9      align-content: center;
10 }
```

justify-content:
start
end
center
stretch
space-around
space-between
space-evenly;

align-content:
start
end
center
stretch
space-around
space-between
space-evenly;



JUSTIFY & ALIGN ITEMS

```
.container {
    height: 100%;
    display: grid;
    grid-gap: 3px;
    grid-template-columns: repeat(12, 1fr);
    grid-template-rows: 40px auto 40px;
    justify-items: center;
    /* align-items: center; */
}
.header {
    grid-column: 1 / -1;
}
.menu {
    grid-column: 1 / 3;
}
.content {
    grid-column: 3 / -1;
    /* justify-self: center; */
    align-self: end;
}
.footer {
    grid-column: 1 / -1;
}
```

justify-self / align-self:
start
end
center
stretch (default);

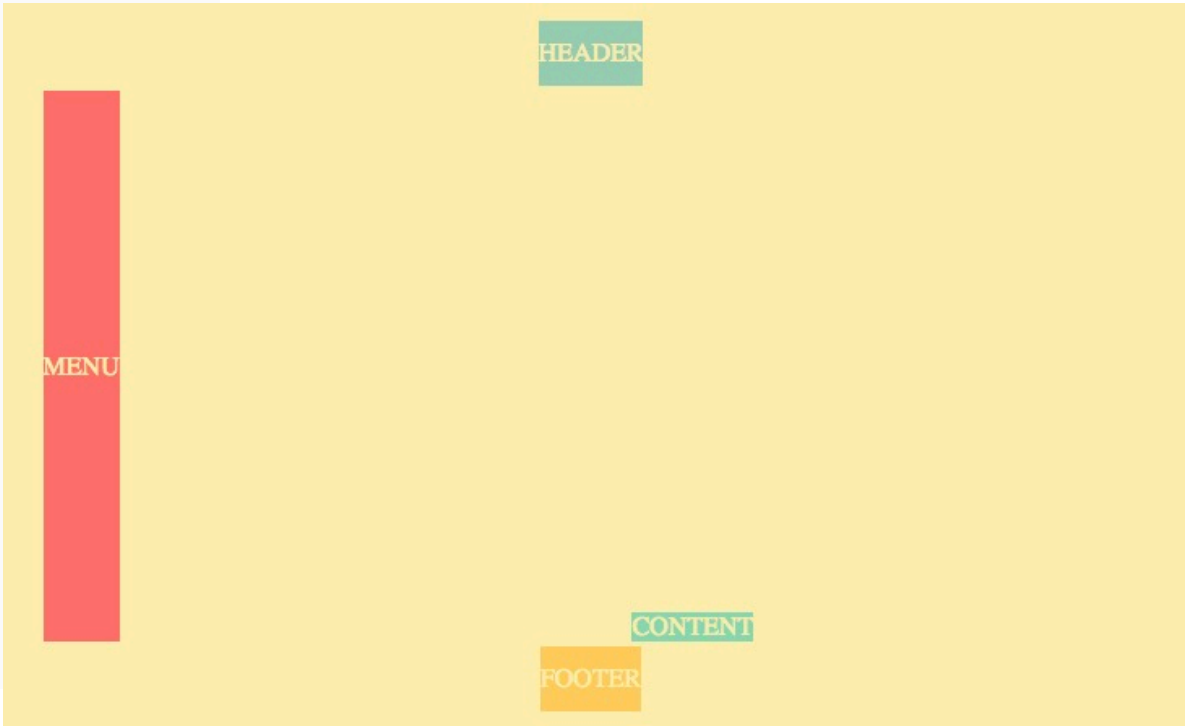


IMAGE GRID

```
.container {
    display: grid;
    grid-gap: 5px;
    grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
    grid-auto-rows: 75px;
    grid-auto-flow: dense;
}
.horizontal {
    grid-column: span 2;
}
.vertical {
    grid-row: span 2;
}
.big {
    grid-column: span 2;
    grid-row: span 2;
}
```

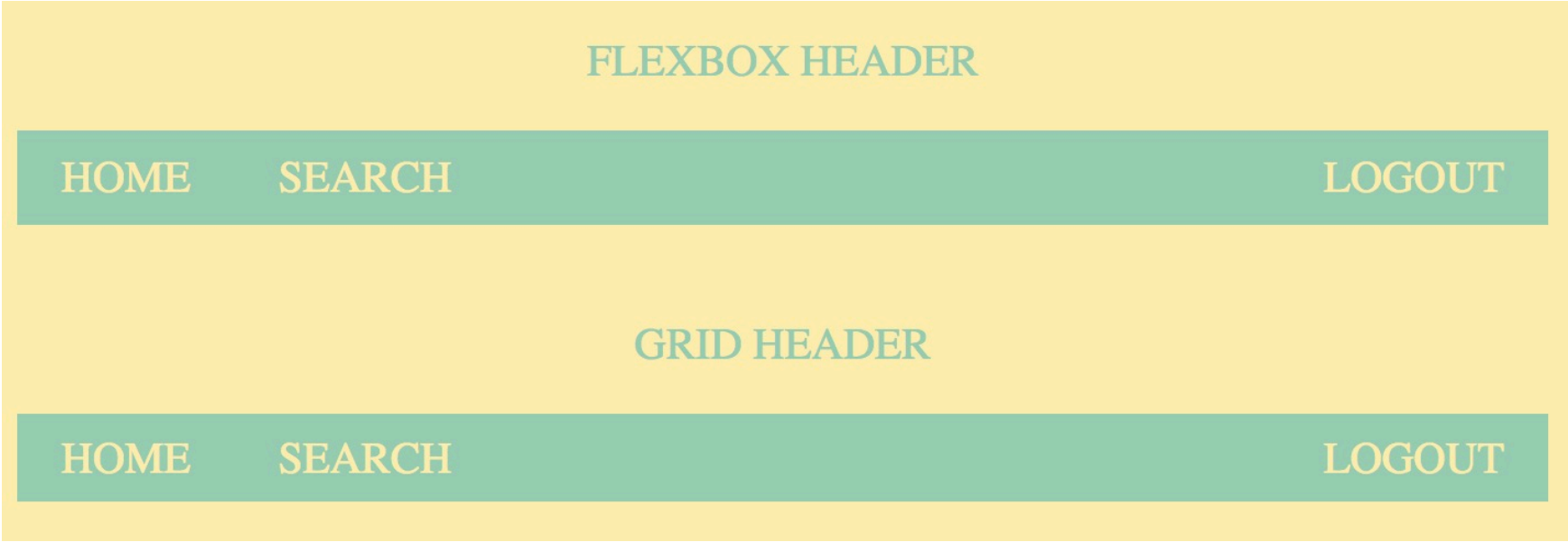

Flexbox
CSS Grid

FLEXBOX VS GRID

```css
.flexbox-header {
    display: flex;
}
.flexbox-header > div:nth-child(3) {
    margin-left: auto;
}
```

```css
.grid-header {
  display: grid;
  grid-template-columns: repeat(12, 1fr);
  grid-template-rows: 40px;
}
.grid-header div:last-child {
  grid-column: 12 / -1;
}
```

4 lines of CSS code (FLEX) VS 8 lines of CSS code (GRID)



FLEXBOX VS GRID

```css
.container,
.wrapper {
  flex-direction: column;
  align-items: stretch;
}
.main {
  justify-content: stretch;
  align-items: stretch;
  margin-bottom: 3px;
}
.wrapper {
  flex: 1;
}
.header {
  height: 40px;
  margin-bottom: 3px;
}
.menu {
  margin-right: 3px;
}
.content {
  height: 200px;
}
.footer {
  height: 40px;
}
```

```html
<html>
    <head>
        <title>Positioning items</title>
        <link rel="stylesheet" href="index.css">
        <link rel="stylesheet" href="basic.css">
    </head>
    <body>
        <div class="container">
          <div class="main">
            <div class="menu">MENU</div>
            <div class="wrapper">
              <div class="header">HEADER</div>
              <div class="content">CONTENT</div>
            </div>
          </div>
          <div class="footer">FOOTER</div>
        </div>
    </body>
</html>
```

26 lines of CSS code (FLEX)

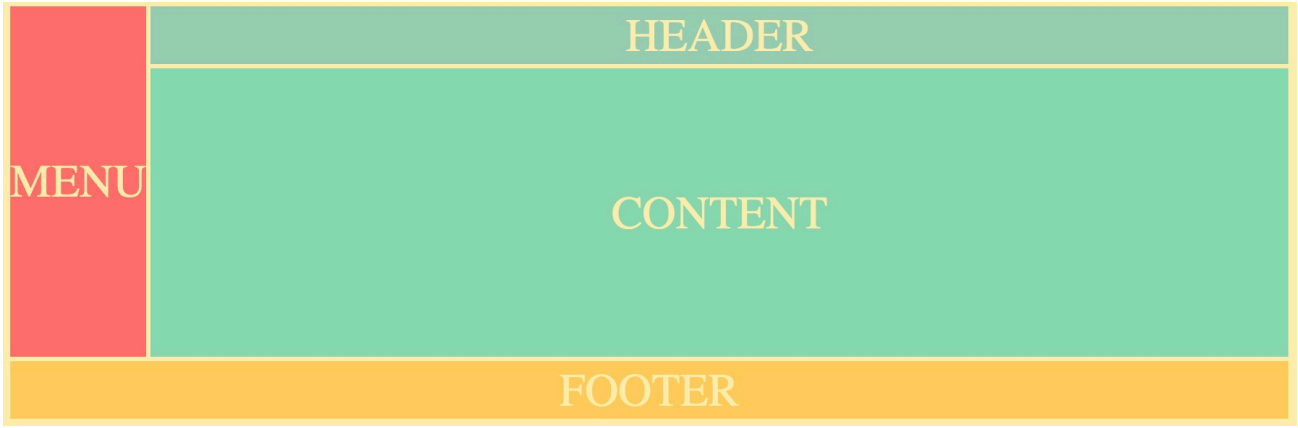
VS

17 lines of CSS code (GRID)

10 lines of HTML code (FLEX)

VS

5 lines of HTML code (GRID)



FLEXBOX VS GRID

FlexBox support: **98.24%**

CSS Flexible Box Layout Module

- CR

Usage — % of all users

Global: 94.89% + 3.34% = 98.24%

unprefixed: 94.52% + 2.29% = 96.82%

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 | Apply filters | Show all | ?

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser
					10-11.5			
		2-21	4-20	3.1-6	12.1	3.2-6.1		
6-9		22-27	21-28	6.1-8	15-16	7-8.4		2.1-4.3
10	12-17	28-65	29-72	9-12	17-57	9-11.4		4.4-4.4.4
11	18	66	73	12.1	58	12.1	all	67
		67-68	74-76	TP		12.2		

Grid support: **91.98%**

CSS Grid Layout (level 1)

- CR

Usage — % of all users

Global: 89.42% + 2.56% = 91.98%

unprefixed: 89.42%

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.

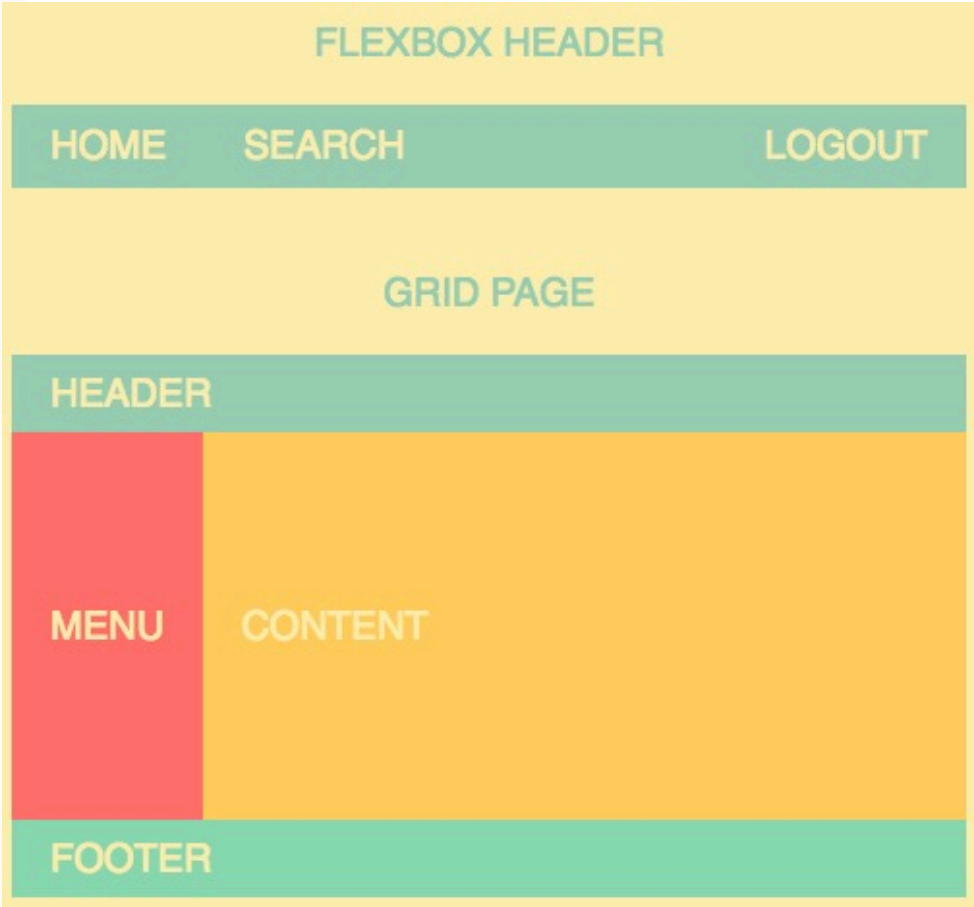
Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser
		2-39	4-28					
		40-51	29-56		10-27			
6-9	12-15	52-53	57	3.1-10	28-43	3.2-10.2		
10	16-17	54-65	58-72	10.1-12	44-57	10.3-11.4		2.1-4.4.4
11	18	66	73	12.1	58	12.1	all	67
		67-68	74-76	TP		12.2		

FLEXBOX VS GRID

```css
.flexbox-header {
    display: flex;
}
.flexbox-header > div:nth-child(3) {
    margin-left: auto;
}

.grid-page {
    display: grid;
    grid-template-columns: repeat(12, 1fr);
    grid-template-rows: 40px 200px 40px;
}
.header {
    grid-column: 1 / -1;
}
.menu {
    grid-column: 1 / 2;
}
.content {
    grid-column: 2 / -1;
}
.footer {
    grid-column: 1 / -1;
}
```

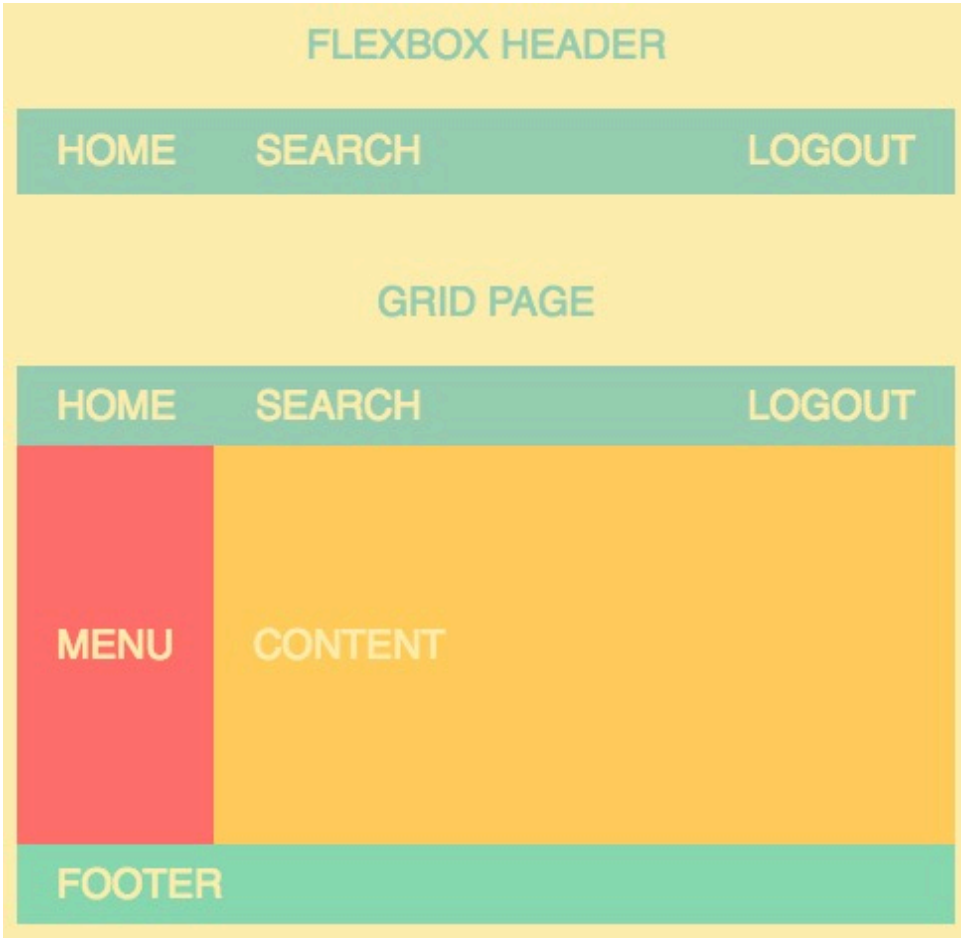
Flexbox: one dimensional layouts



Grid: two dimensional layouts

FLEXBOX & GRID

```
.grid-page {
    display: grid;
    grid-template-columns: repeat(12, 1fr);
    grid-template-rows: 40px 200px 40px;
}
.header {
    grid-column: 1 / -1;
    display: flex;
}
.header > div:nth-child(3) {
    margin-left: auto;
}
.menu {
    grid-column: 1 / 2;
}
.content {
    grid-column: 2 / -1;
}
.footer {
    grid-column: 1 / -1;
}
```




Questions?

USEFUL LINKS

https://html.com/html5/ - HTML5

https://raygun.com/blog/10-reasons-css-preprocessor/ - CSS Preprocessors

https://www.hongkiat.com/blog/css-post-processors-tips-resources/ - CSS Post-Processors

https://medium.com/@js_tut/the-complete-css-flex-box-tutorial-d17971950bdc - Flex tutorial

https://flexboxfroggy.com/ - Flex learning game

CSS Grid:
https://scrimba.com/g/gR8PTE

https://css-tricks.com/snippets/css/complete-guide-grid/

http://grid.malven.co/

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout

https://www.udemy.com/advanced-css-and-sass/ - CSS courses