CSS

Introduction

THIS DOCUMENT COVERS

- Introduction

Basics

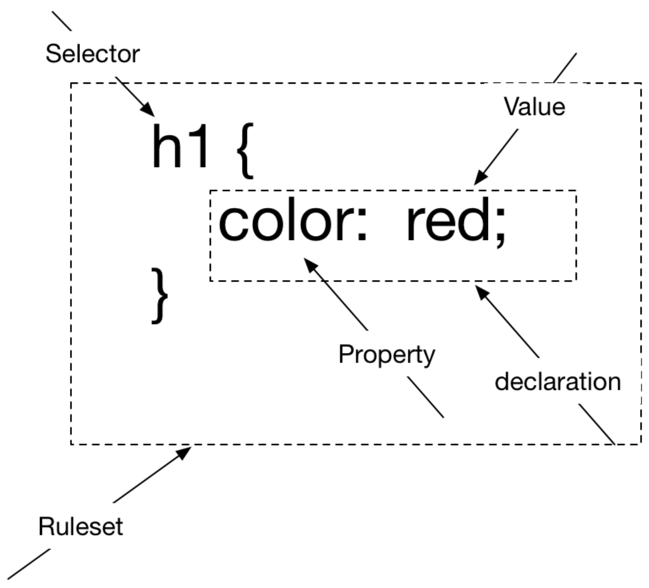
Adding to Document

CSS can be specified in three ways as this fragment shows

```
<head>
    <meta charset="utf-8">
    <title>Hello World</title>

    <!-- 1. External Style Sheet-->
    <link rel="stylesheet" href="../mystyle.css">

    <!-- 2.  Embedded Style sheet -->
    <style>
        h1 { text-align: center; }
    </style>
</head>

<body>
    <!-- 3. Inline Style-->
    <h1>Suomi</h1>
    <p  style="color: blue;">Terve Maailmalle</p>
</body>
```

Resolving Conflicts

The following shows the nomenclature of a CSS ruleset



The browser must resolve a value for every property of every element on the page. When multiple rulesets target the same property, the browser must resolve the conflict. In order to resolve conflicts, the browser looks at

- Origin
- Specificity
- Source Order

Origin

Any styles provided by the browser are known as **user-agent** style. Such styles provide basic default styling for headings, paragraphs and lists. Any styles provided by the developer are known as **author** styles. Author styles always take precedence over user-agent styles.

Specificity

A selector can contain any number of id, class and tag names. The number of each determine a selectors specificity. Consider the following

```
.nav a {
     color: white;
 }
```

It has specificity of `0,1,1` because it has no id names, 1 class name and 1 tag names. Comparing the specificity of two selectors tells us which one wins,

> ### INLINE STYLES
>
> Any style specified inline as a style attribute on an HTML element takes precedence over any other selector

SOURCE ORDER

If origin and specificity do not provide a winner, the style defined later or included later wins

Inheritance

If a property does not receive a value from the cascade it can, in some cases, inherit a value from ancestor elements in the DOM hierarchy. Not all properties are inheritable. Mainly it applies to properties that affect text such as font, colour, etc.

Units of Measure

Absolute

On most systems a pixel defines a logical pixel rather than a physical pixel. Each logical pixel is typically set such that it is approximately 1/96 of an inch. The following table shows other units of measurement

CSS Term	Meaning	Description/Example
px	Logical Pixel	1/96inch
pt	Point	1/72 inch
pc	Pica	12pt
mm	Millimetre	
cm	Centimetre	
in	Inch	25.4mm

Relative

EMS

1em is set to equal the font size of the current element for all properties other than font size. For font size 1em is set to the font size of the parent element. This can lead to unexpected results if we use the same em measure for font-size and another measure. Consider the following code.

Listing 1 HTML

```
<body>
  Body Text
  <div>
    Div Text
  </div>
  </div>
</body>
```

Listing 2 CSS

```
body {
  font-size:24px;
}

body div {
  font-size:0.75em;
```

```
    margin:0.75em;
  }
```

The font size of the div is 0.75x24px=18px. The margin is then 0.75x18px=13.5px

Em is good for padding, margin and element sizing. It can lead to strange behaviour where we have nested elements each inheriting font-size from their parent and multiplying it by a constant factor

```
  <div>
    Level One
    <div>
      Level Two
      <div>
        Level Three
        <div>
          Level four
        </div>
      </div>
    </div>
  </div>

  body {
    font-size:24px;
  }

  body div {
    font-size:0.75em;
    margin:0.75em;
  }
```

Level One
Level Two
Level Three
Level four

REM

The rem measure prevents this problem from the previous section. It stands for Root em and so 1rem is equal to the font-size of the root element in the DOM.

Guidelines

“When in doubt, use rems for font size, pixels for borders, and ems for most other properties.

Viewport Relative

The viewport is the area of the browser window not including the toolbar, address bar etc.

CSS Term	**Meaning**

vh	1/100 of viewport height
vw	1/100 of viewport width
vmin	1/100 of whatever is smaller viewport width or height
vmax	1/100 of whatever is larger viewport width or height

50vh is 50% of the height of the viewport

Unitless

Line height is typically specified as a unitless value. This causes it to be always relative to the local font-size of the element

Questions – Basics

Why don’t we advise using javascript to apply inline styles?

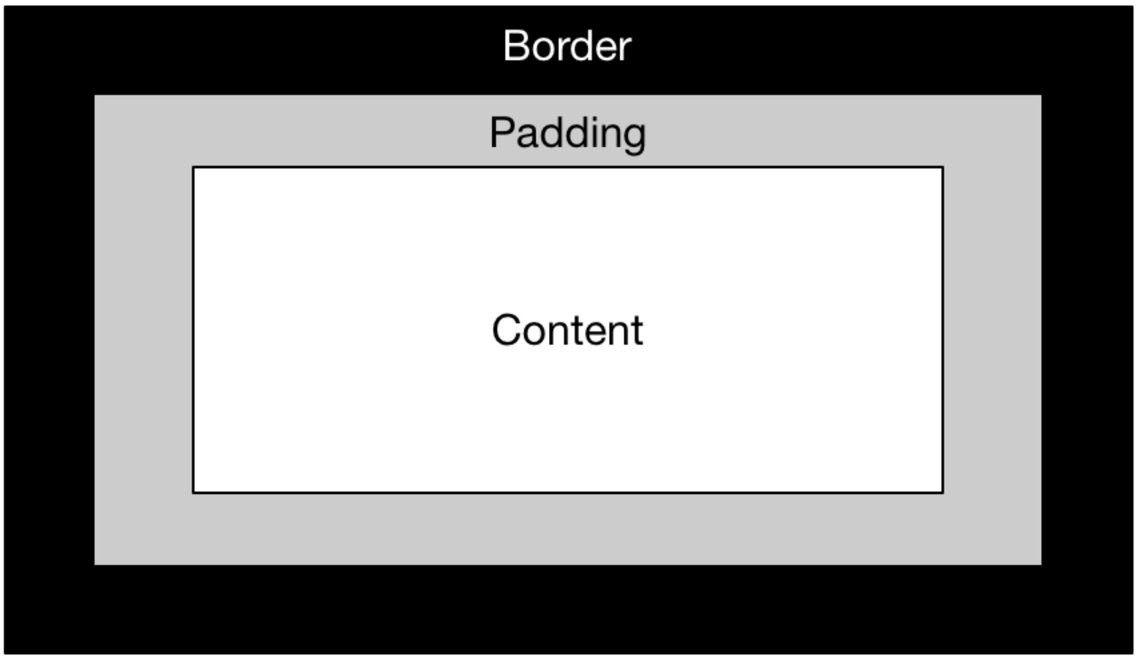
Users only chance to override is with !important.

What is better?

Use JavaScript to add and remove classes specified in a stylesheet you provide.

Box Model

For the purpose of layout and sizing every element of an HTML document can be thought of as a box. From a rendering perspective the box consists of content, padding and border.

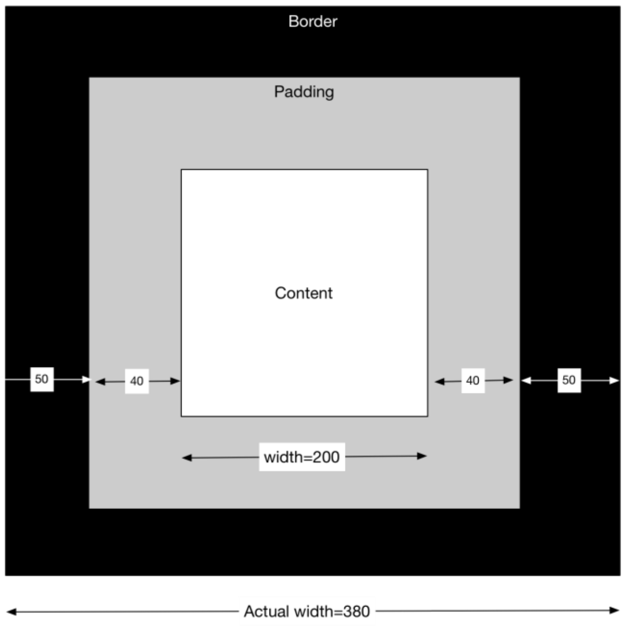


The actual rendered width is then the sum of the content width, padding width and border width. Similarly, the actual rendered height is the sum of the content height, padding height and border height.

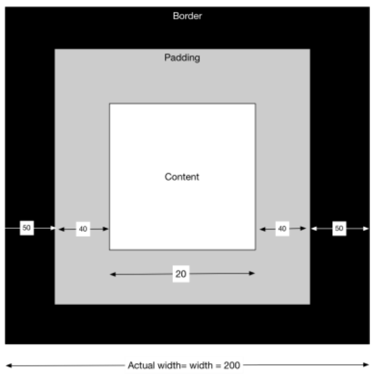
Box-sizing

The meaning of the CSS attributes width and height depend on the value of the CSS `box-sizing` attribute. With the default value of `content-box` the width and height only refer to the width and height of the content. The actual width and height are then given by taking (content) width/height and adding padding and border. Consider the following CSS/html and observer the result

```
          .outer {
              display: flex;
              box-sizing: content-box;
              background-color: lightgray;
              padding: 40px;
              margin: 30px;
              width: 200px;
              height: 200px;
              border: 50px solid black;
          }

          .inner {
              background-color: white;
              flex-grow: 1;
          }
      </style>
  </head>
  <body>
      <div id="outer" class="outer">
          <div class="inner">
          </div>
      </div>
</body>
```



Now if we change our box-sizing to be `border-box` we get a different behaviour. The width and height refer to the actual width and height of 200px. The margins and border are inside this box and take space away from the content area.



Margins

Margins are added around the box and do not count towards the render size. One peculiar feature of web development is that if a bottom margin of one component touches the top margin of another component the margins collapse. The effect of this is that the gap between the elements borders is set to the larger of the two margins rather than the sum. This applies even if the element on the top or bottom is inside another component. As long as two vertical margins touch margin collapsing will take place.

One solution is to put either the top or bottom element inside a container with a small amount of padding or border. This prevents the margins touching and collapsing.

Inline and Block Boxes

Every element has a `display` property that defines how its box is treated. The type of the element determines the default value. Div elements have a default value of `block` whereas span elements have a default value of `inline`. The default values can be changed by simply setting the value of the display property via CSS.

Inline tags don’t create a line break before and after them whereas box tags do. While one can add margin and padding to the left or right of inline elements one cannot add margin or padding above and below them. If you really want an inline element to obey top and bottom margins use the value `inline-block`.

Float

Float is designed to pull an element to the side of its container allowing the rest of the content to wrap around it.

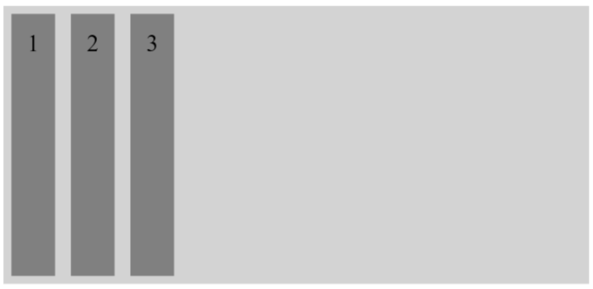
Flexbox

Row Left to right

Defaults

Consider the following basic setup. We have explicitly specified some default values.

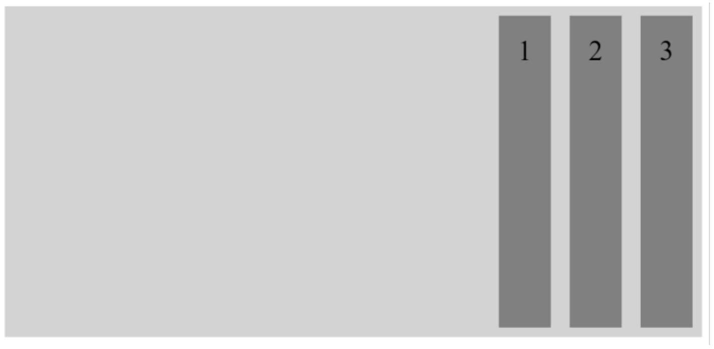
```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to the left hand sice */
  justify-content: flex-start;

  /* Align content vertically to stetch top to bottom */
  align-items: stretch;
}

.flex-container > div {
  background-color: grey;
  margin: 10px;
  padding: 20px;
  font-size: 30px;
}
```



ALIGN HORIZONTALLY TO RIGHT

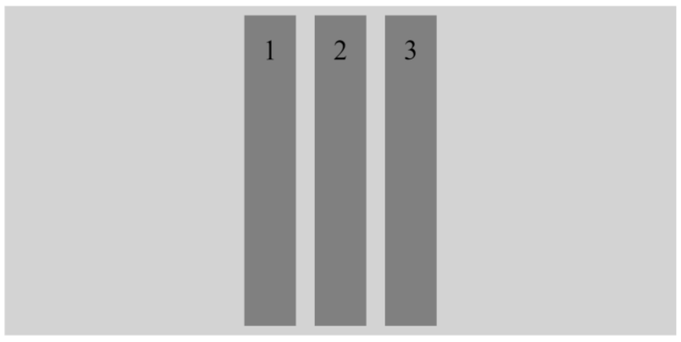
We modify the justify-content to alight the content to the right-hand side.

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to the right hand sice */
  justify-content: flex-end;

  /* Align content vertically to stetch top to bottom */
  align-items: stretch;
}
```



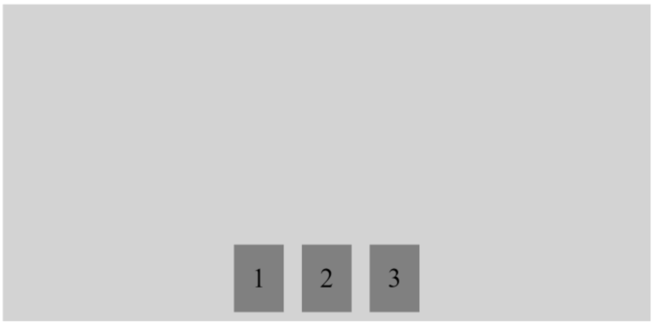
Align Horizontally to Center

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;


  /* Align content horizontally to center */
  justify-content: center;

  /* Align content vertically to stetch top to bottom */
  align-items: stretch;
}
```



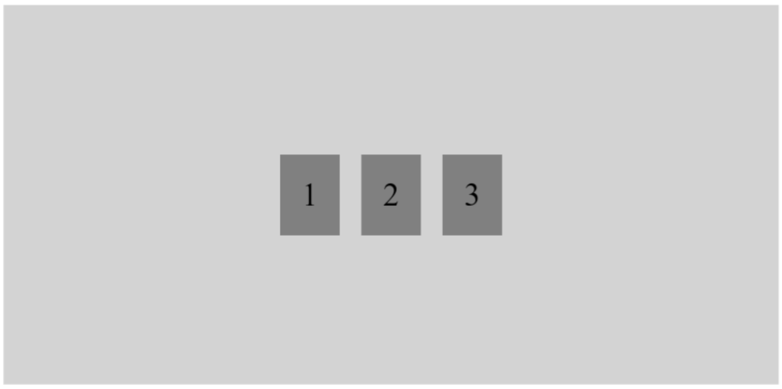
Align Vertically to Bottom

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to center */
  justify-content: center;

  /* Align content vertically to top */
  align-items: flex-end;
}
```



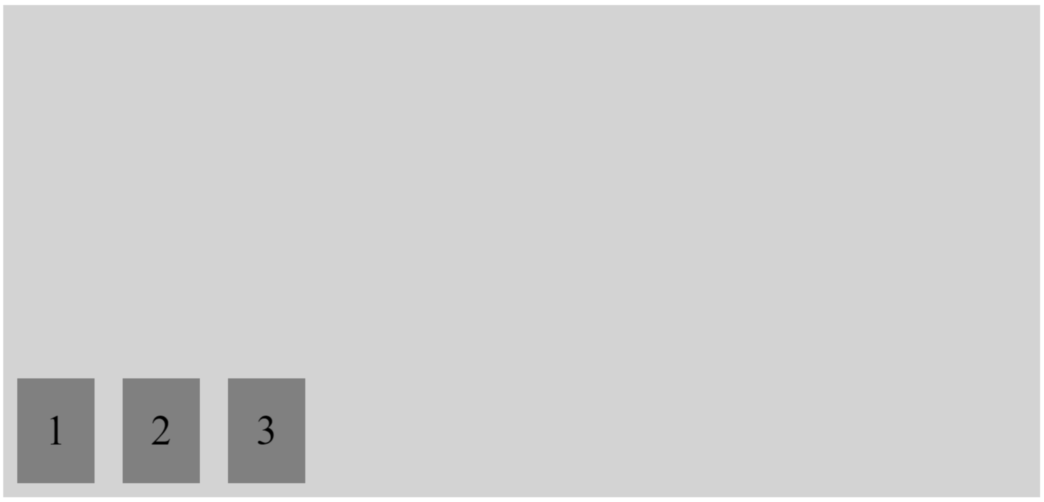
Align Vertically to Center

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to center */
  justify-content: center;

  /* Align content vertically to stetch top to bottom */
  align-items: stretch;
}
```



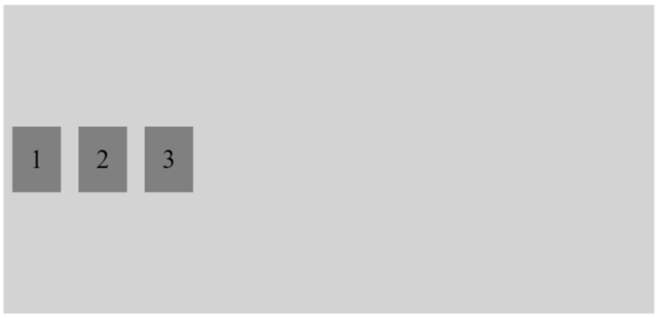
ALIGN VERTICALLY TO BOTTOM

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to left */
  justify-content: flex-start;

  /* Align content vertically to bottom */
  align-items: flex-end;
}
```



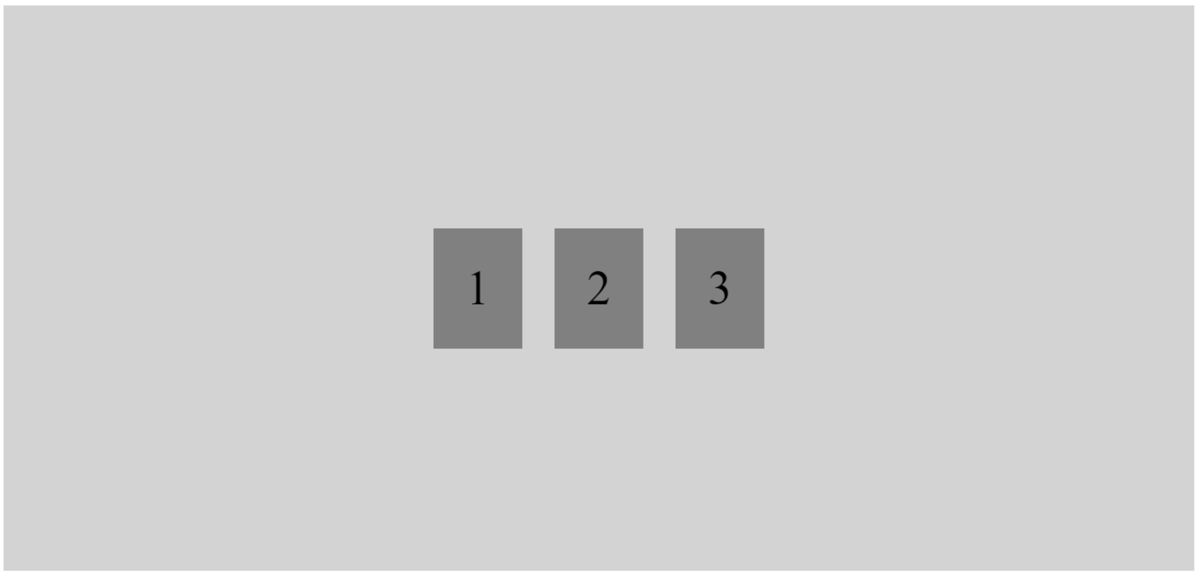
ALIGN VERTICALLY TO CENTRE

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to center */
  justify-content: flex-start;

  /* Align content vertically to center */
  align-items: center;
}
```



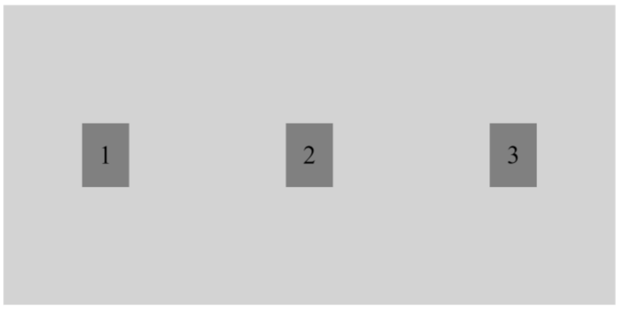
Center horizontally and vertically

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to center */
  justify-content: center;

  /* Align content vertically to stetch top to bottom */
  align-items: center;
}
```



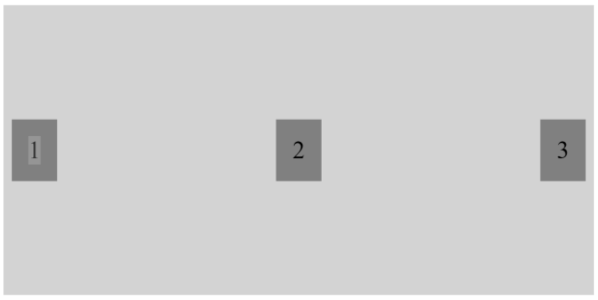
Horizonal Space Around

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to center */
  justify-content: space-around;

  /* Align content vertically to stetch top to bottom */
  align-items: center;
}
```



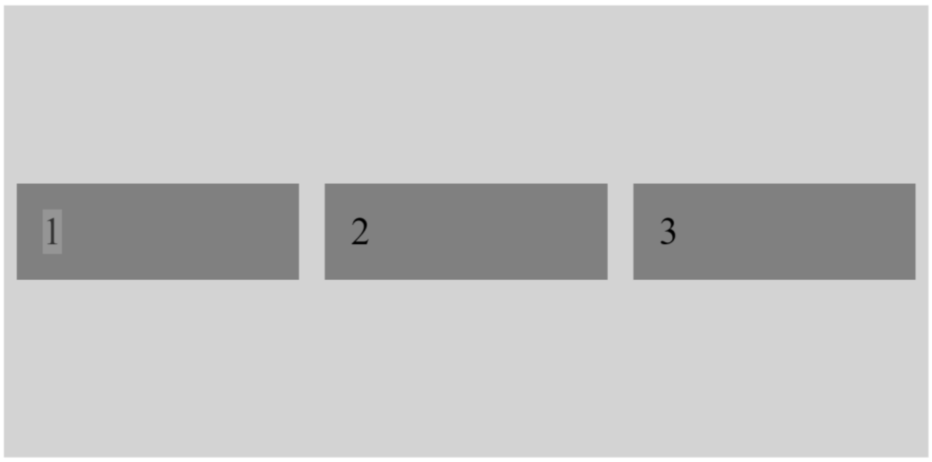
Horizonal Space Between

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;

  /* Lay out items from left to right */
  flex-direction: row;

  /* Align content horizontally to center */
  justify-content: space-between;

  /* Align content vertically to stetch top to bottom */
  align-items: center;
}
```



Equal Width Columns

```
.flex-container {
  display: flex;
  background-color: lightgray;
  height: 350px;
  flex-direction: row;
  justify-content: flex-start;
  align-items: center;
}
.flex-container > div {
  background-color: grey;
  margin: 10px;
  padding: 20px;
  font-size: 30px;
  flex-basis: 33%;
}
```

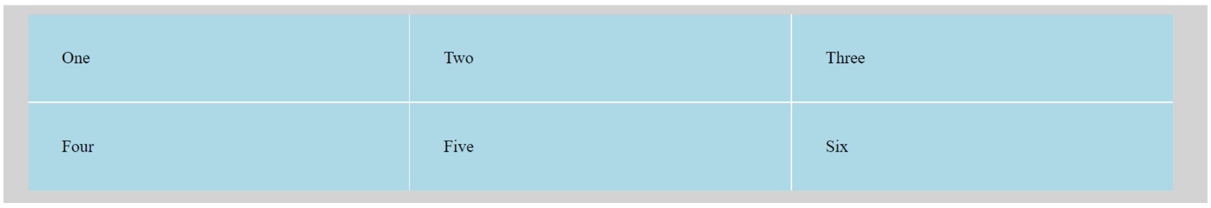


Grid

Equal Sized Cells

```
<body>
  <div class=container>
    <div class="cell">One</div>
    <div class="cell">Two</div>
    <div class="cell">Three</div>
    <div class="cell">Four</div>
    <div class="cell">Five</div>
    <div class="cell">Six</div>
  </div>
</body>
```

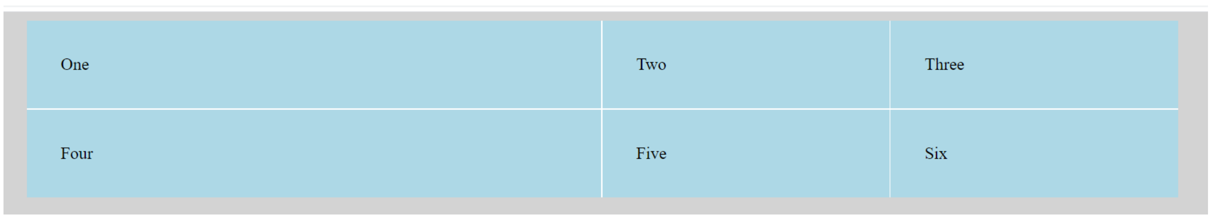
```
body {
    background-color: lightgray;
}

.container {
    max-width: 1080px;
    margin: 0 auto;
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr;
    row-gap: 1px;
    column-gap: 1px;
    background-color: white;
}

.cell {
    background-color: lightblue;
    padding: 2em;
}
```



Unequal Columns

We can modify the CSS from the previous example

```
  .container {
      max-width: 1080px;
      margin: 0 auto;
      display: grid;
      grid-template-columns: 2fr 1fr 1fr;
      grid-template-rows: 1fr 1fr;
      row-gap: 1px;
      column-gap: 1px;
      background-color: white;
  }

  .cell {
      background-color: lightblue;
      padding: 2em;
  }
```



One	Two	Three
Four	Five	Six

Web Layout

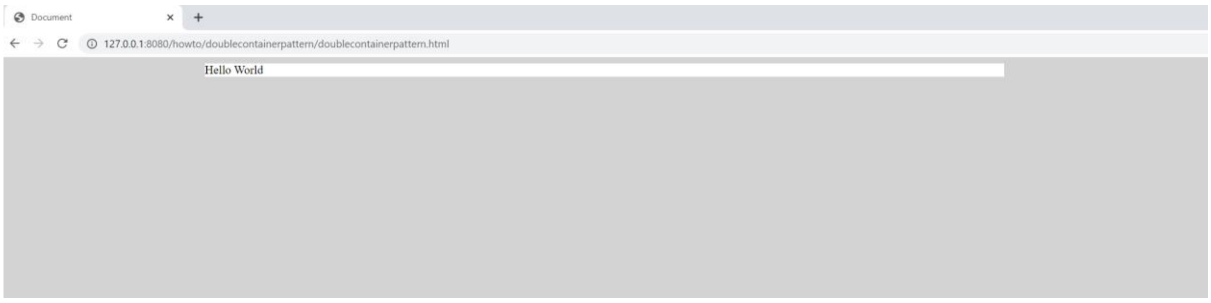
Double Container Pattern

The first thing we do in our CSS is set it up such that by default all elements use the border-box model such that the width includes the border and padding.

```
/* Make all elements use the border-box */
:root { box-sizing: border-box;}
*,::before,::after {box-sizing: inherit;}
```

Now we use the double container pattern to centre our content in a central column no wider than 1080px. The body forms the outer container which naturally take 100% of space. The inner container is the div marked “container” which has max width of 1080px. Setting the left and right margins to auto means they fill any extra space.

```
/* The Outer container in the double container pattern */
body
{
   background-color: lightgray;
}


.container
{
   max-width: 1080px;
   margin: 0 auto;
   background-color: white;
}
```


Document
127.0.0.1:8080/howto/doublecontainerpattern/doublecontainerpattern.html

Hello World

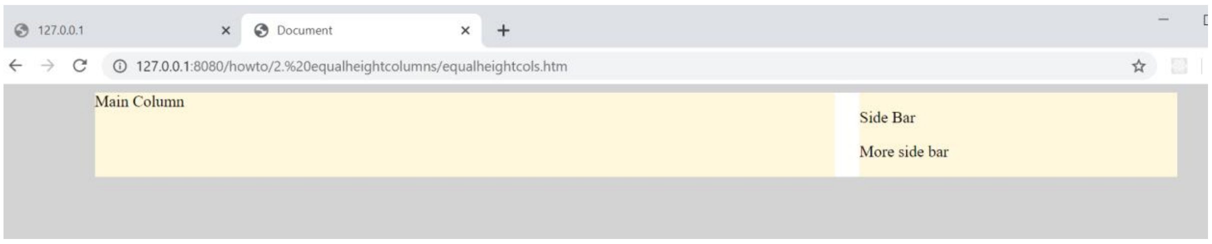
Equal height columns

Adding

```
<body>
    <div class="container">
        <main class="main">
            Main Column
        </main>
        <aside class="sidebar">
            <p>Side Bar</p>
            <p>More side bar</p>
        </aside>
    </div>
</body>

.container {
    max-width: 1080px;
    margin: 0 auto;
    background-color: white;
    display: flex; ❶
}

.main {
    width: 70%; ❷
    background-color: cornsilk;
 }

.sidebar {
    width: 30%; ❸
    margin-left: 1.5em;❹
    background-color: cornsilk;
}
```

We make use of a flex container to hold our two columns. ❶ We set the main column to take 70% of its parent container's width ❷and the sidebar to take 30% of its container's width ❸. Finally, we add a left margin to the sidebar to create a gutter between the columns. ❹ Note that the sum of the column widths plus the sidebar's left margin is more than the total 100%. The flexbox deals with this so the content fits in the width of the container.

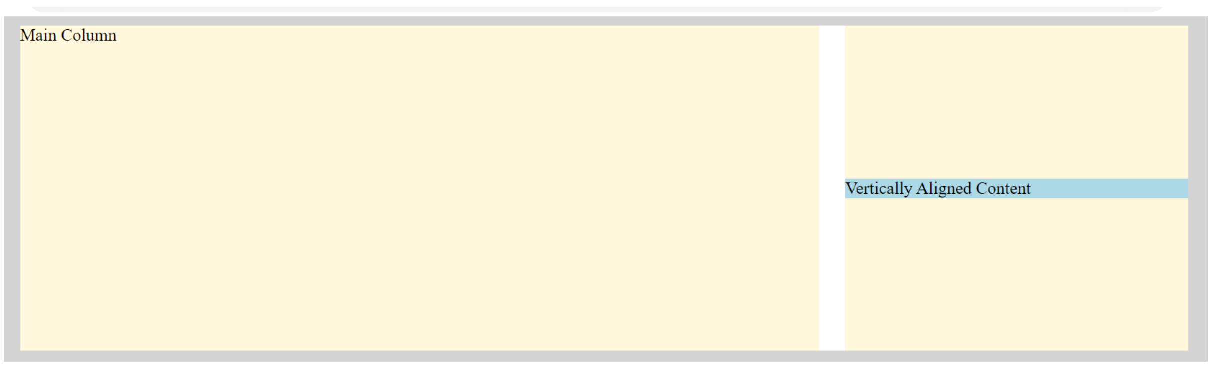
Note that the sidebar's content needs more vertical space than the main column and that the main column is set to the width of the sidebar due to flexbox.


127.0.0.1
Document
127.0.0.1:8080/howto/2.%20equalheightcolumns/equalheightcols.htm
Main Column
Side Bar
More side bar

Vertical Centring

```
<body>
    <div class="container">
        <main class="main">
            Main Column
        </main>
        <aside class="sidebar">
            <div class="sidebar-content">
            </div>
        </aside>
    </div>
</body>

.container {
    max-width: 1080px;
    margin: 0 auto;
    background-color: white;
    display: flex;
}

.main {
    width: 70%;
    background-color: cornsilk;
    min-height: 300px;
 }

.sidebar {
    display: flex;
    width: 30%;
    margin-left: 1.5em;
    background-color: cornsilk;
    align-items: center;
}

.sidebar-content {
    background-color: lightblue;
    flex-basis: 100%;
}
```

Using a flexbox as the display on the sidebar and setting its align-items to `center` gives us vertical alignment. We set the vertically aligned content’s flex-basis to 100% so it stretched across the entire width of the parent container.



Boxes with Flexbox

The following shows how to create three equal columns with equal margin on left and right side

```
<body>
    <div class="container">
        <main class="main">
            <div class="col">
               One
            </div>
            <div class="col">
                Two
            </div>
            <div class="col">
                Three
            </div>
        </main>
    </div>
</body>
```