Purwadhika
Digital Technology School

Full Stack AI Software Development

Advanced CSS Techniques

Job Connector Program

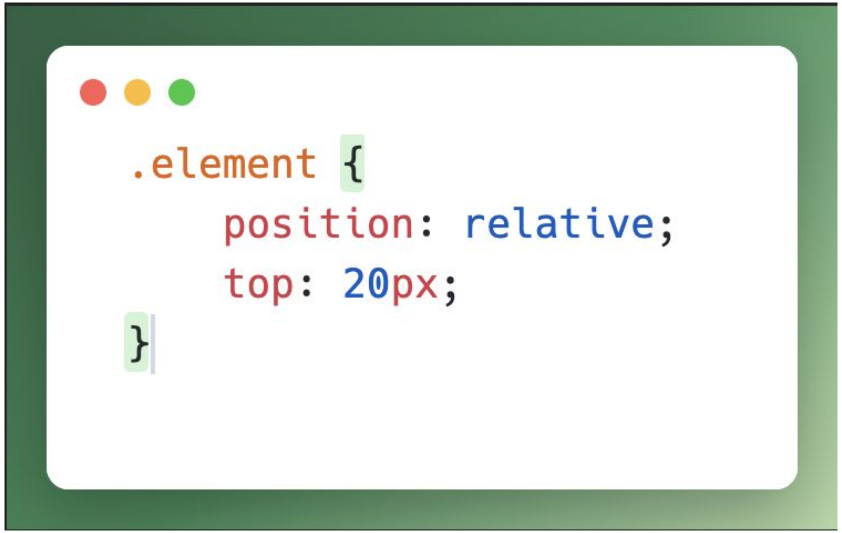
Position

The position CSS property sets how an element is positioned in a document. The **top, right, bottom, and left** properties determine the final location of positioned elements.

There are several position in css:

- Static (default)
- Relative
- Absolute
- Fixed
- Sticky

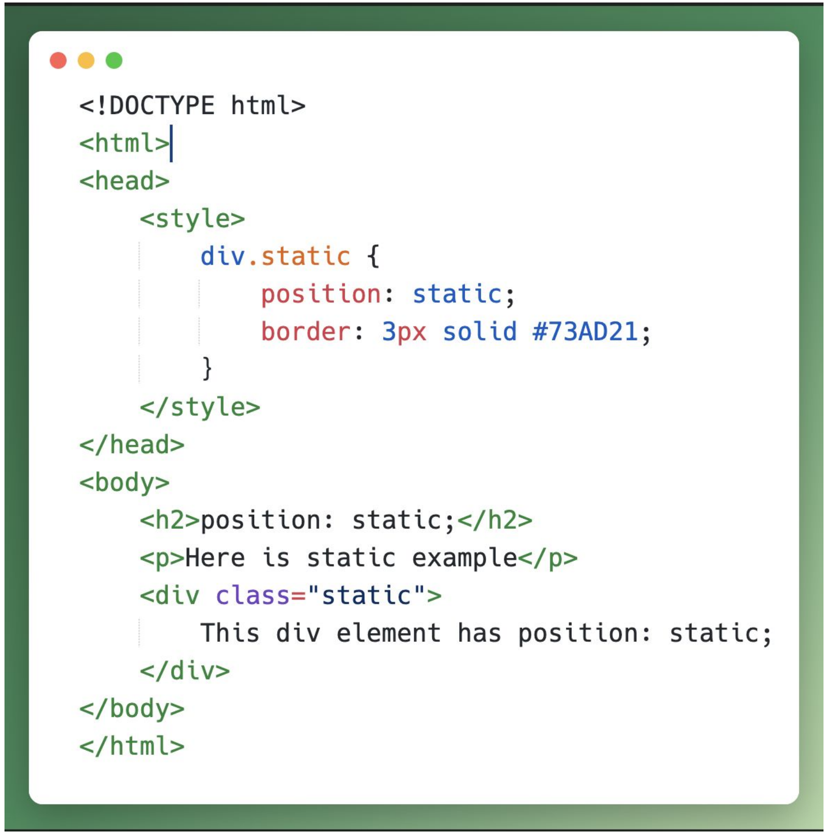
https://developer.mozilla.org/en-US/docs/Web/CSS/position



```
.element {
    position: relative;
    top: 20px;
}
```

Position - Static

An element with position: static is not positioned in any special way, it is always positioned according to the normal flow of the page.

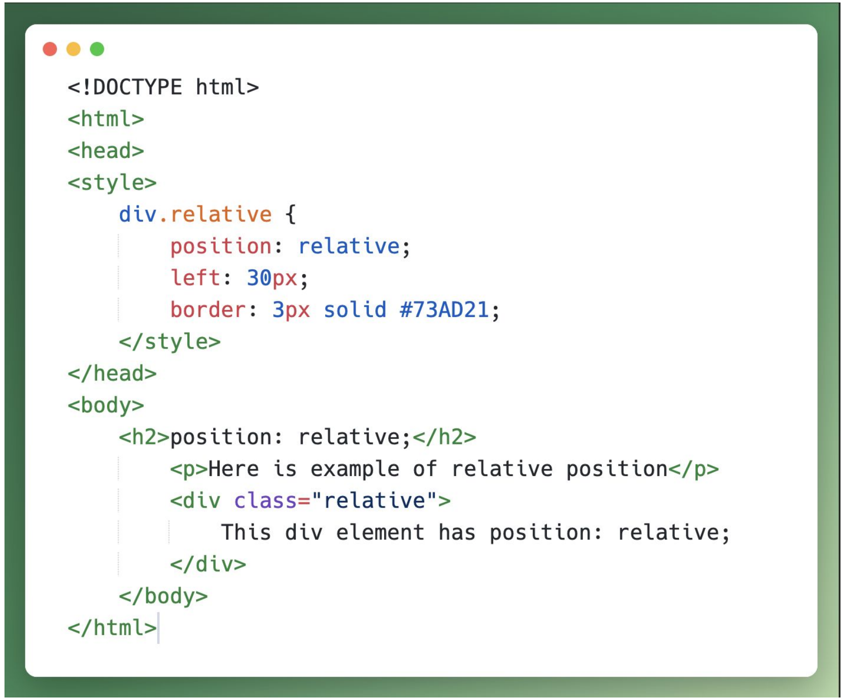


```
<!DOCTYPE html>
<html>
<head>
    <style>
        div.static {
            position: static;
            border: 3px solid #73AD21;
        }
    </style>
</head>
<body>
    <h2>position: static;</h2>
    <p>Here is static example</p>
    <div class="static">
        This div element has position: static;
    </div>
</body>
</html>
```

Position - Relative

An element with position: relative; is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Check out the example, other content will not be adjusted to fit into any gap left by the element.



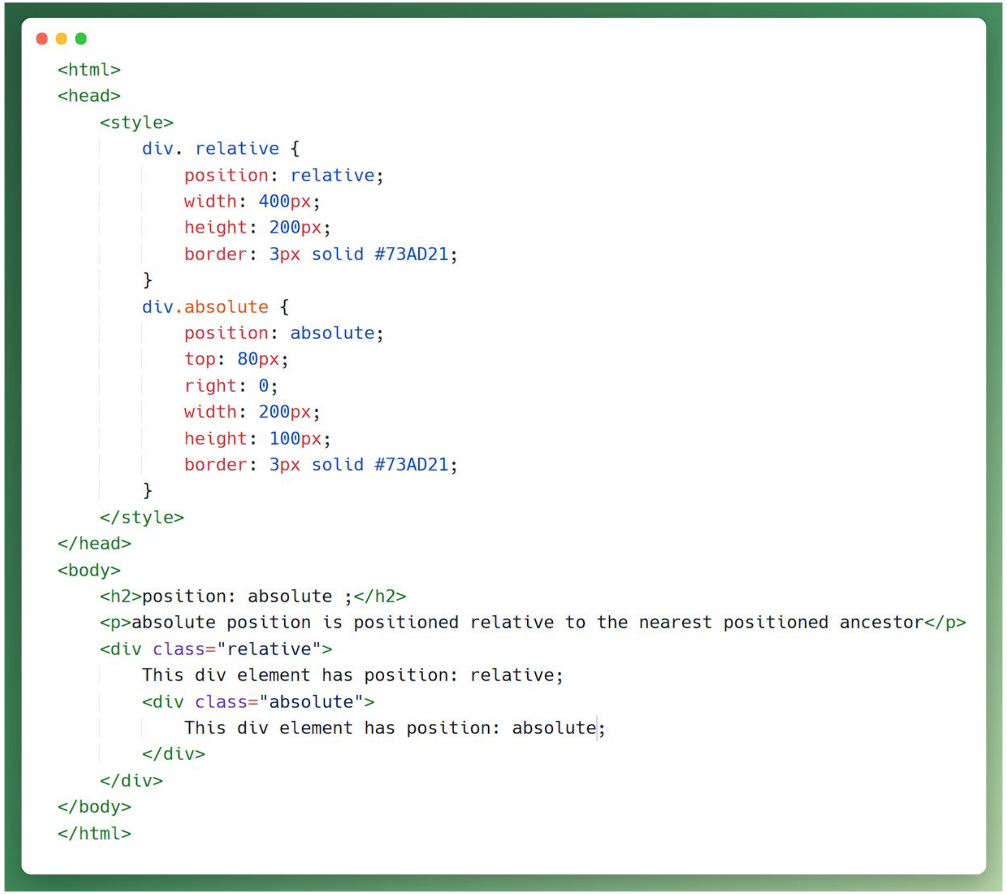
```
<!DOCTYPE html>
<html>
<head>
<style>
    div.relative {
        position: relative;
        left: 30px;
        border: 3px solid #73AD21;
    </style>
</head>
<body>
    <h2>position: relative;</h2>
        <p>Here is example of relative position</p>
        <div class="relative">
            This div element has position: relative;
        </div>
    </body>
</html>
```

Position - Absolute

An element with position: absolute is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

If an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

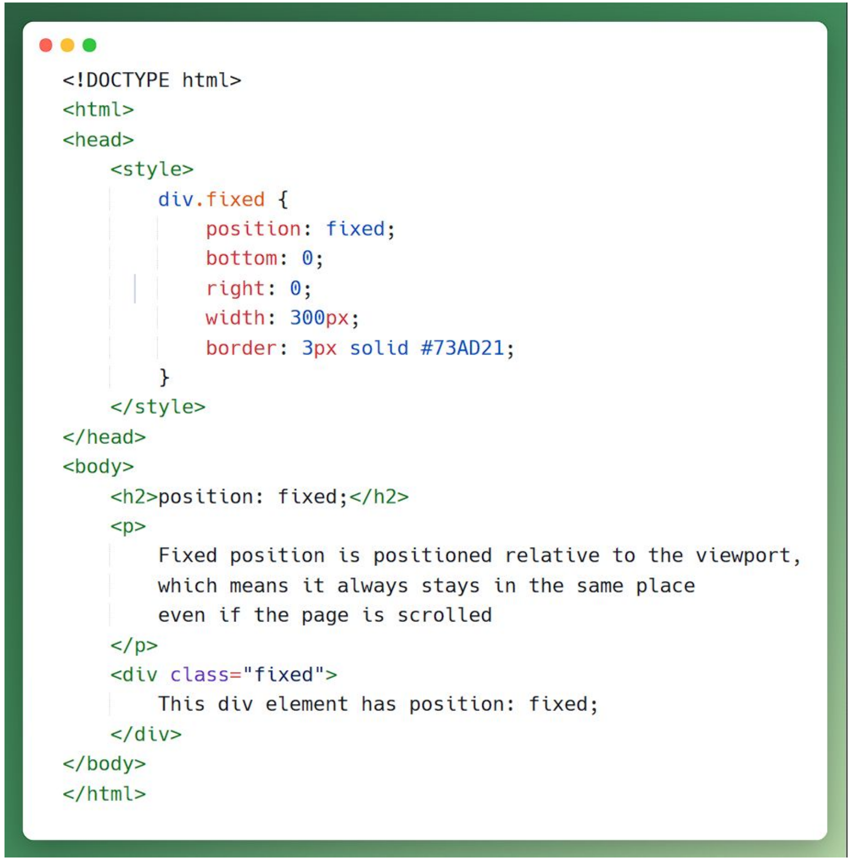


```
<html>
<head>
    <style>
        div. relative {
            position: relative;
            width: 400px;
            height: 200px;
            border: 3px solid #73AD21;
        }
        div.absolute {
            position: absolute;
            top: 80px;
            right: 0;
            width: 200px;
            height: 100px;
            border: 3px solid #73AD21;
        }
    </style>
</head>
<body>
    <h2>position: absolute ;</h2>
    <p>absolute position is positioned relative to the nearest positioned ancestor</p>
    <div class="relative">
        This div element has position: relative;
        <div class="absolute">
            This div element has position: absolute;
        </div>
    </div>
</body>
</html>
```

Position - Fixed

An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.



```
<!DOCTYPE html>
<html>
<head>
    <style>
        div.fixed {
            position: fixed;
            bottom: 0;
            right: 0;
            width: 300px;
            border: 3px solid #73AD21;
        }
    </style>
</head>
<body>
    <h2>position: fixed;</h2>
    <p>
        Fixed position is positioned relative to the viewport,
        which means it always stays in the same place
        even if the page is scrolled
    </p>
    <div class="fixed">
        This div element has position: fixed;
    </div>
</body>
</html>
```

Media Queries

Media queries are useful when you want to modify your site or app depending on a device's general type (such as print vs. screen) or specific characteristics and parameters (such as screen resolution or browser viewport width).

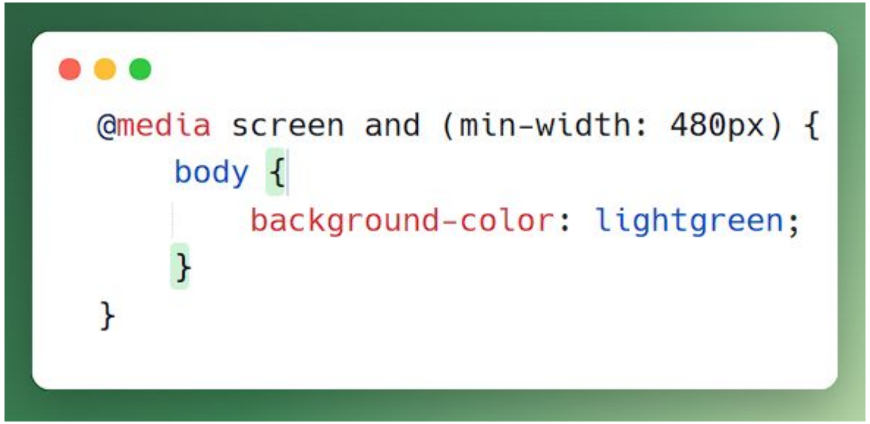
Media queries are used for the following:

- To conditionally apply styles with the CSS @media and @import at-rules.
- To target specific media for the <style>, <link>, <source>, and other HTML elements with the media= attribute.
- To test and monitor media states using the Window.matchMedia() and MediaQueryList.addListener() JavaScript methods.

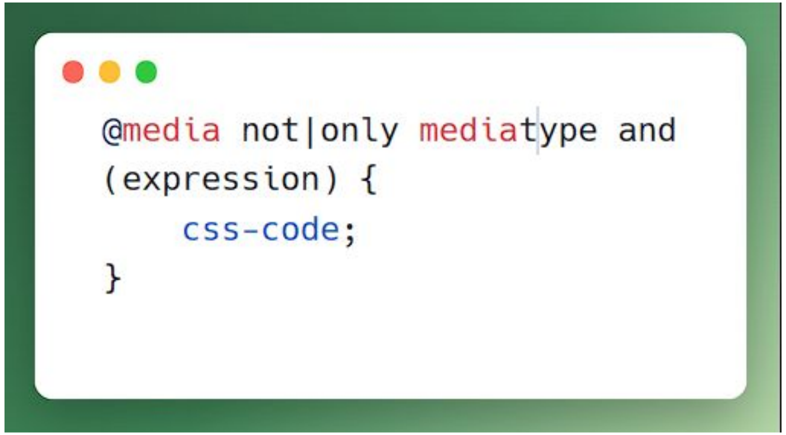
https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

Media Query Syntax

Example:



```
@media screen and (min-width: 480px) {
    body {
        background-color: lightgreen;
    }
}
```



```
@media not|only mediatype and
(expression) {
    css-code;
}
```

The following example shows a menu that will float to the left of the page if the **viewport** is 480 pixels wide or wider (if the **viewport** is less than 480 pixels, the menu will be on top of the content):

Common Media Query Breakpoints

There is no standard exactly defined, but these are some **commonly used ones**

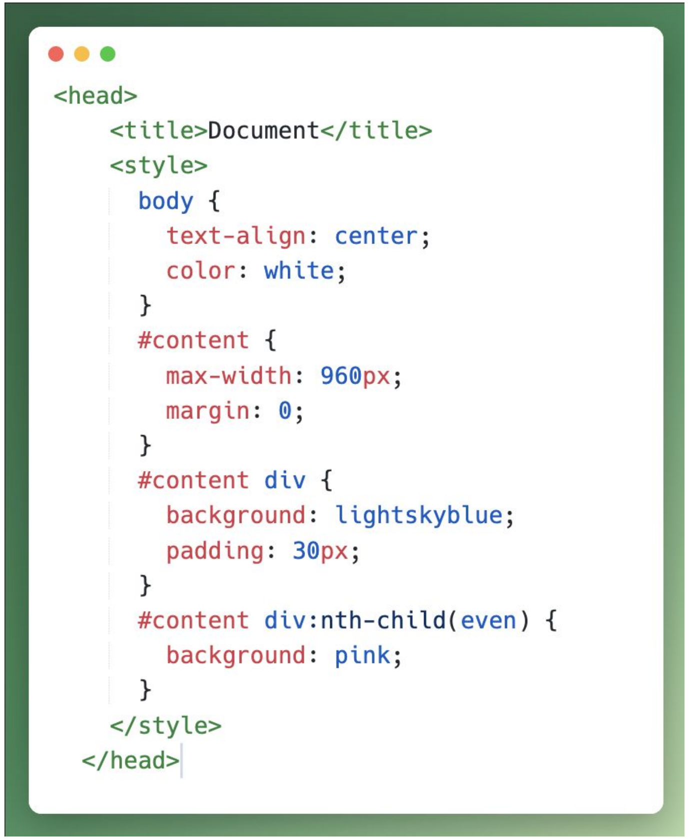
- 320px - 480px: Mobile devices
- 481px - 768px: iPads, Tablets
- 769px - 1024px: Small screens, laptops
- 1025px - 1200px: Desktops, large screens
- 1201px and more: Extra large screens, TV

Grid

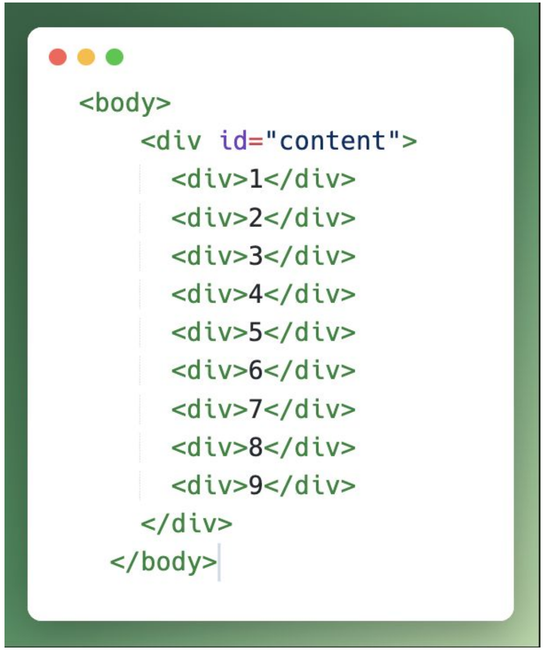
CSS Grid Layout (aka “Grid” or “CSS Grid”), is a two-dimensional grid-based layout system that, compared to any web layout system of the past, completely changes the way we design user interfaces.

Src :
https://developer.mozilla.org/en-US/docs/Web/CSS/grid

Grid Example



```
<head>
    <title>Document</title>
    <style>
      body {
        text-align: center;
        color: white;
      }
      #content {
        max-width: 960px;
        margin: 0;
      }
      #content div {
        background: lightskyblue;
        padding: 30px;
      }
      #content div:nth-child(even) {
        background: pink;
      }
    </style>
  </head>
```



```
<body>
    <div id="content">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
      <div>7</div>
      <div>8</div>
      <div>9</div>
    </div>
  </body>
```

1

2

3

4

5

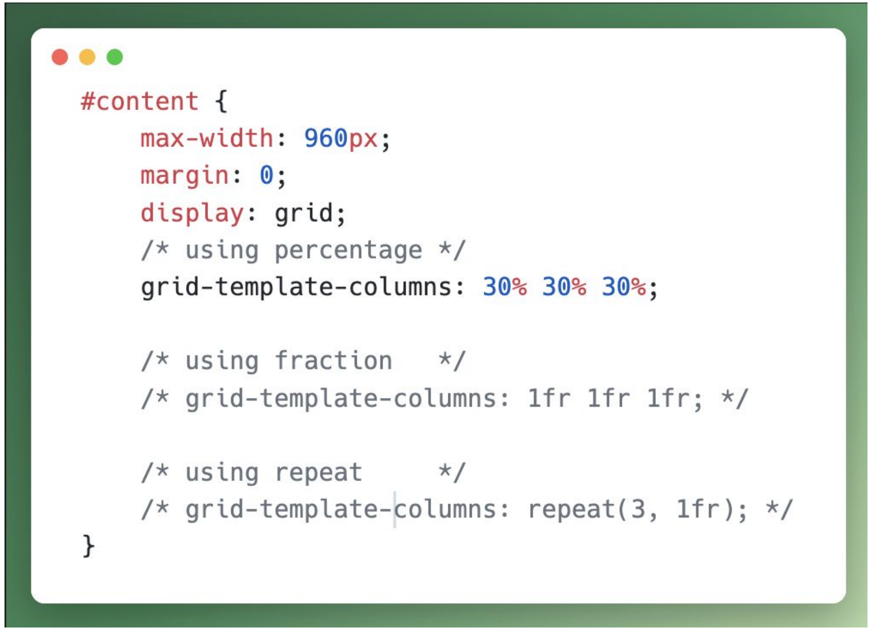
6

7

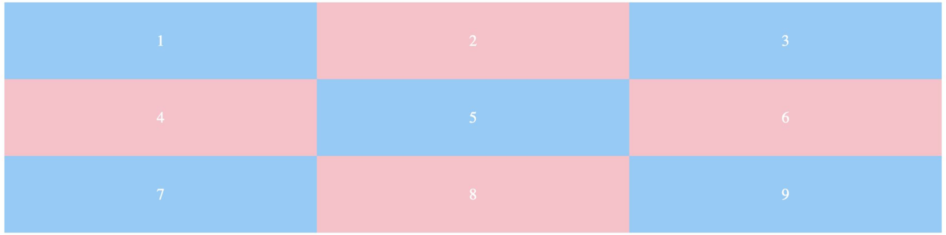
8

9

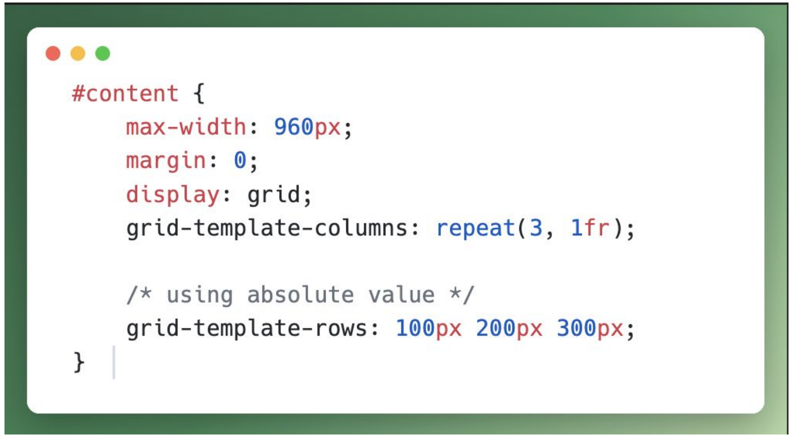
Grid Template Columns



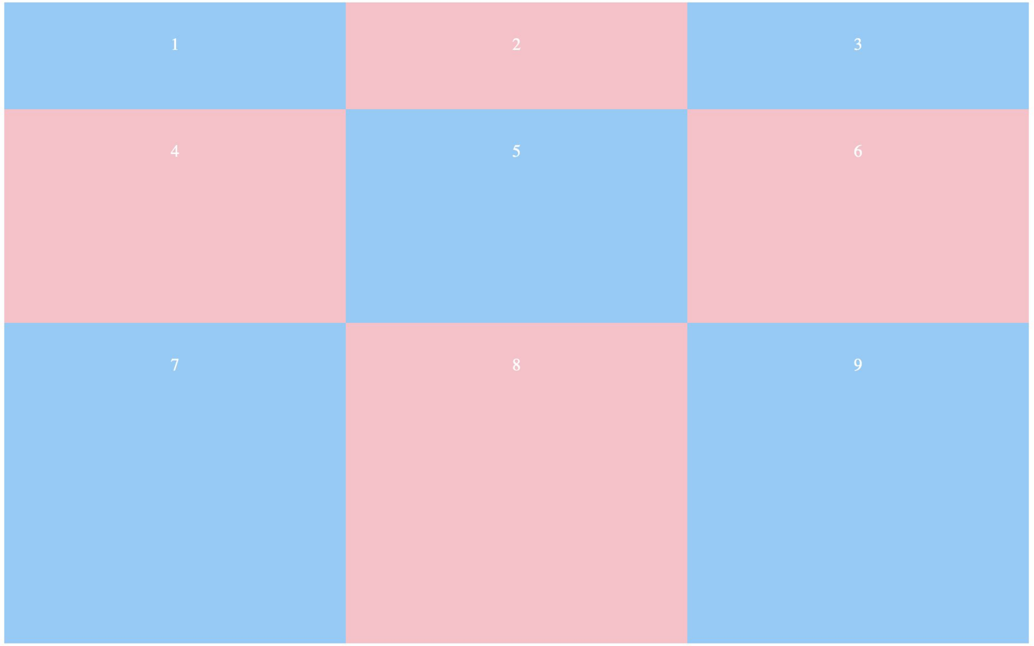
```css
#content {
    max-width: 960px;
    margin: 0;
    display: grid;
    /* using percentage */
    grid-template-columns: 30% 30% 30%;

    /* using fraction   */
    /* grid-template-columns: 1fr 1fr 1fr; */

    /* using repeat     */
    /* grid-template-columns: repeat(3, 1fr); */
}
```



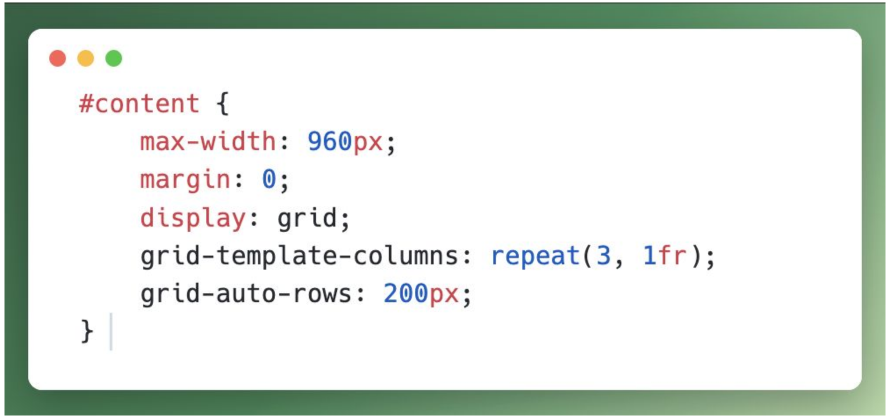
Grid Template Rows



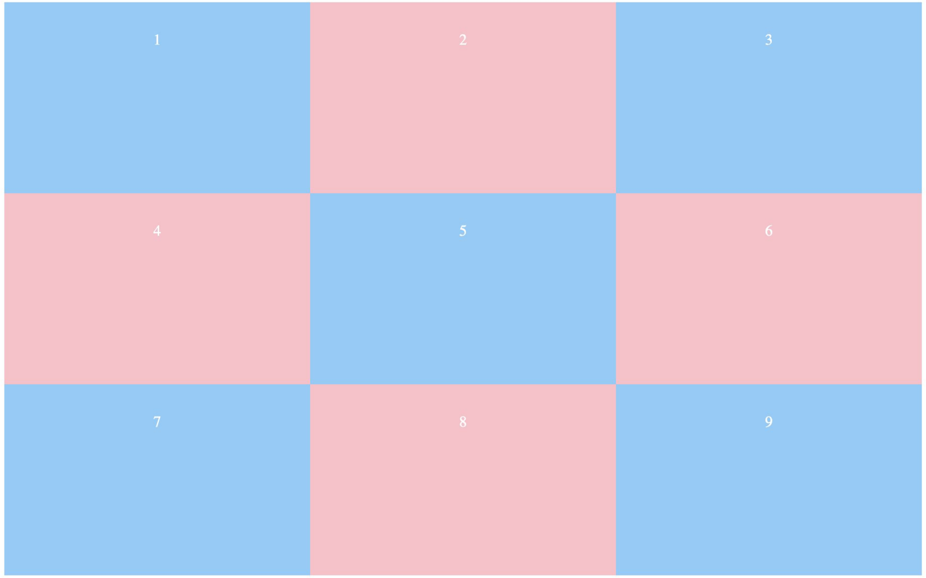
```
#content {
    max-width: 960px;
    margin: 0;
    display: grid;
    grid-template-columns: repeat(3, 1fr);

    /* using absolute value */
    grid-template-rows: 100px 200px 300px;
}
```



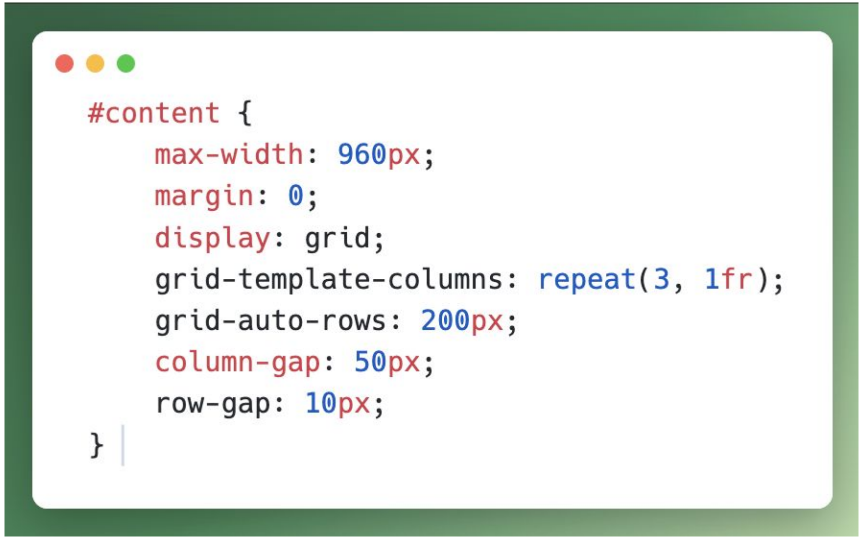
Grid Auto Rows



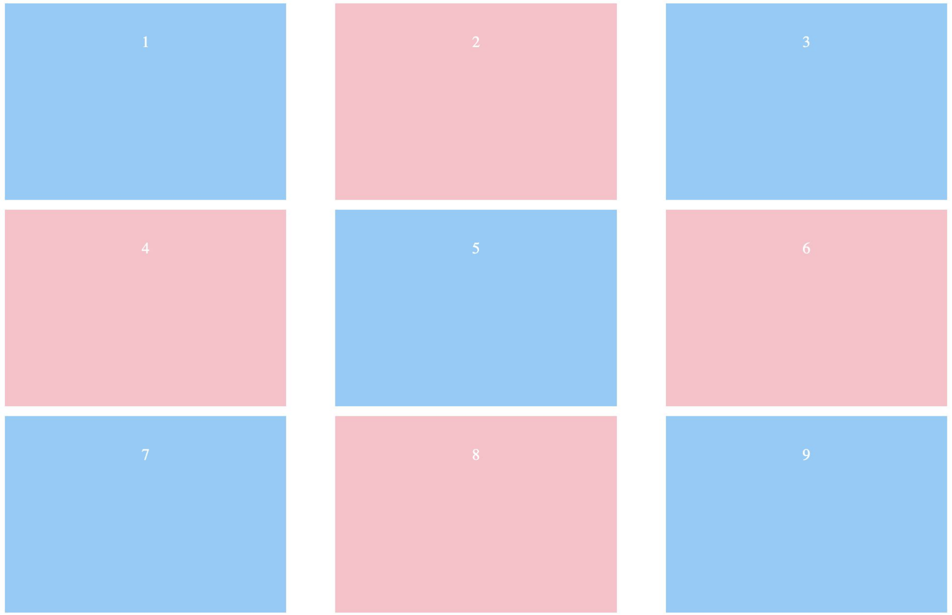
```
#content {
    max-width: 960px;
    margin: 0;
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-auto-rows: 200px;
}
```



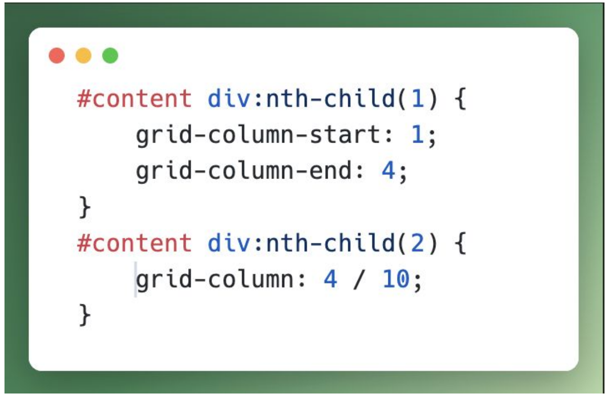
Gap



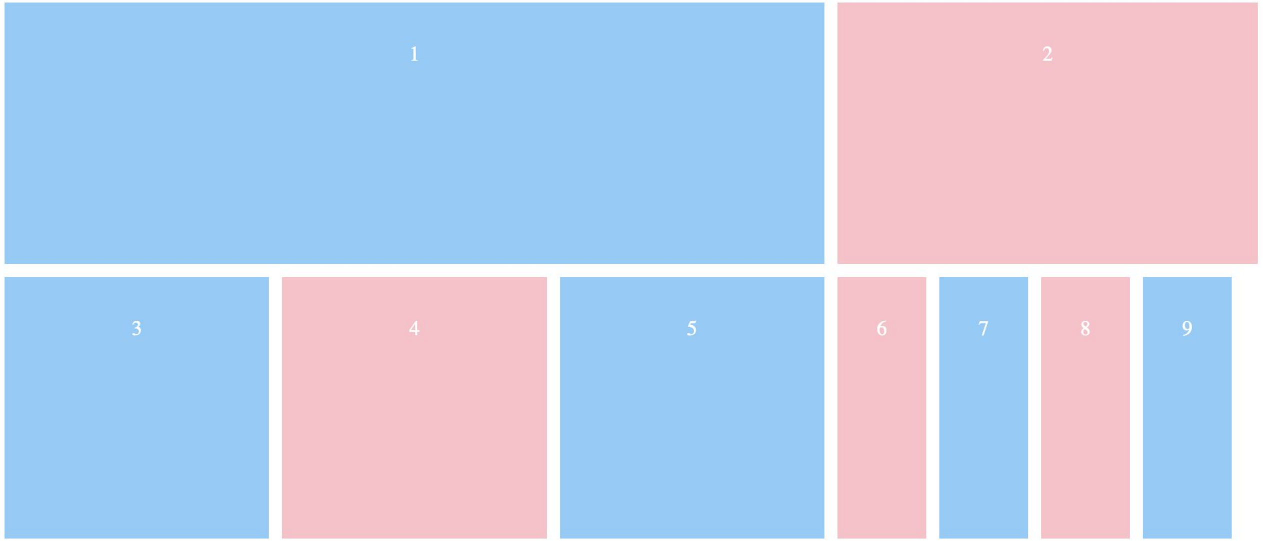
```
#content {
    max-width: 960px;
    margin: 0;
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-auto-rows: 200px;
    column-gap: 50px;
    row-gap: 10px;
}
```



Grid Column



```
#content div:nth-child(1) {
    grid-column-start: 1;
    grid-column-end: 4;
}
#content div:nth-child(2) {
    grid-column: 4 / 10;
}
```

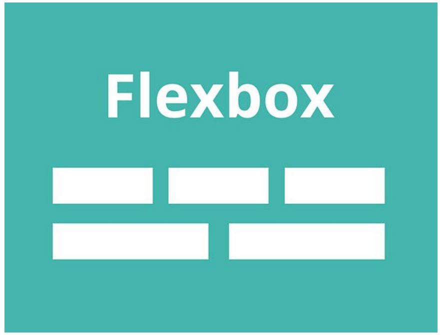


Flex

The CSS Flexbox model is a one-dimensional layout approach designed for efficiently organizing and aligning items within a container. It offers flexibility in managing space distribution and is particularly handy for crafting clean and responsive layouts, making it well-suited for both small-scale designs and mobile-friendly applications.

Features of flexbox:

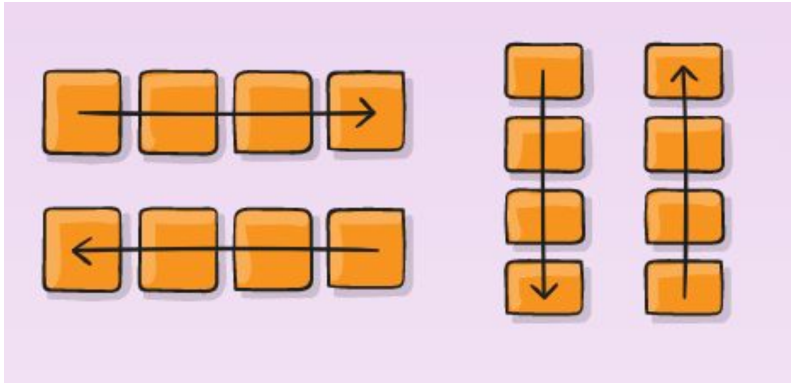
- A lot of flexibility is given.
- Arrangement & alignment of items.
- Proper spacing
- Order & Sequencing of items.



Src : https://developer.mozilla.org/en-US/docs/Web/CSS/flex

Flex Directions

This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

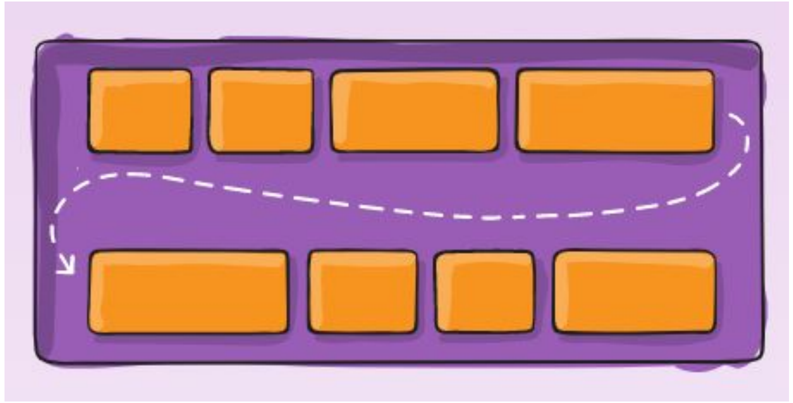


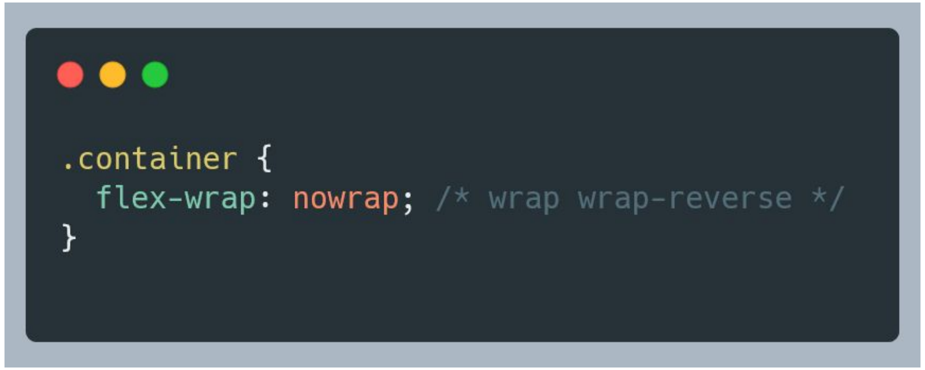
CSS Syntax

```
flex-direction: row|row-reverse|column|column-reverse|initial|inherit;
```

Flex-wrap

By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property.

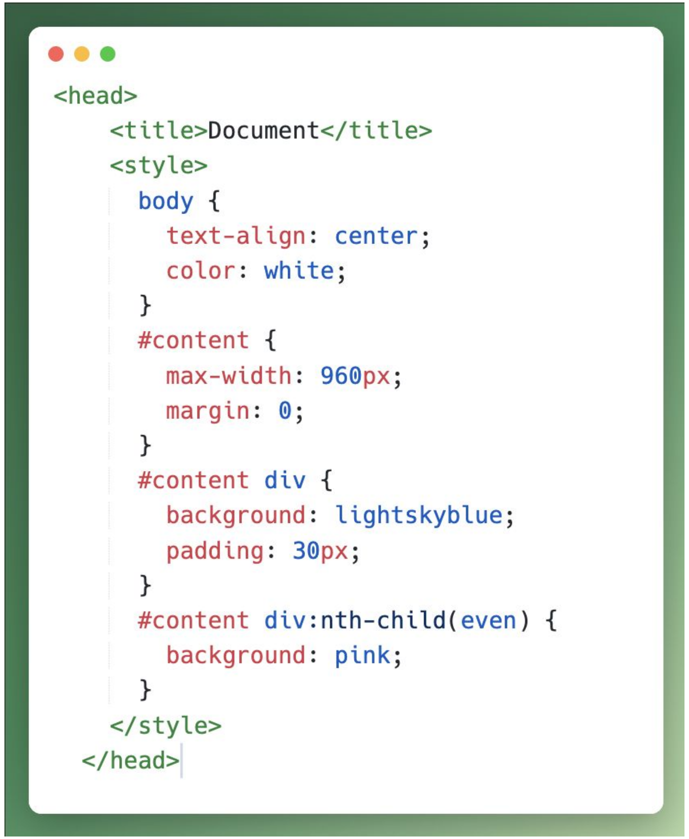




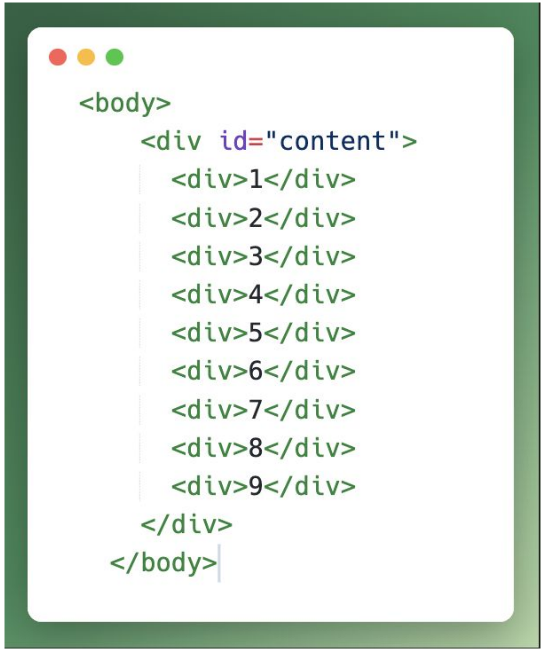
```
.container {
  flex-wrap: nowrap; /* wrap wrap-reverse */
}
```

Find out more on: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Flex Example



```
<head>
    <title>Document</title>
    <style>
      body {
        text-align: center;
        color: white;
      }
      #content {
        max-width: 960px;
        margin: 0;
      }
      #content div {
        background: lightskyblue;
        padding: 30px;
      }
      #content div:nth-child(even) {
        background: pink;
      }
    </style>
  </head>
```



```
  <body>
    <div id="content">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
      <div>7</div>
      <div>8</div>
      <div>9</div>
    </div>
  </body>
```

1

2

3

4

5

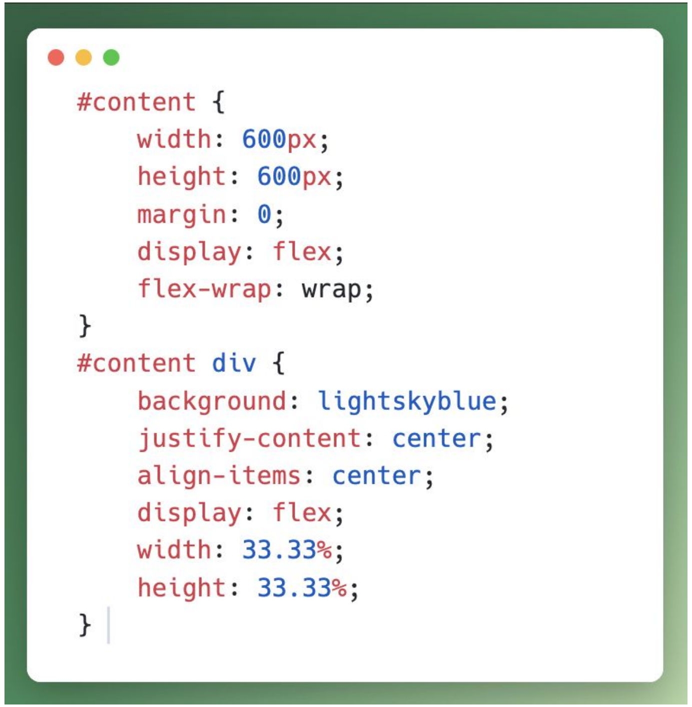
6

7

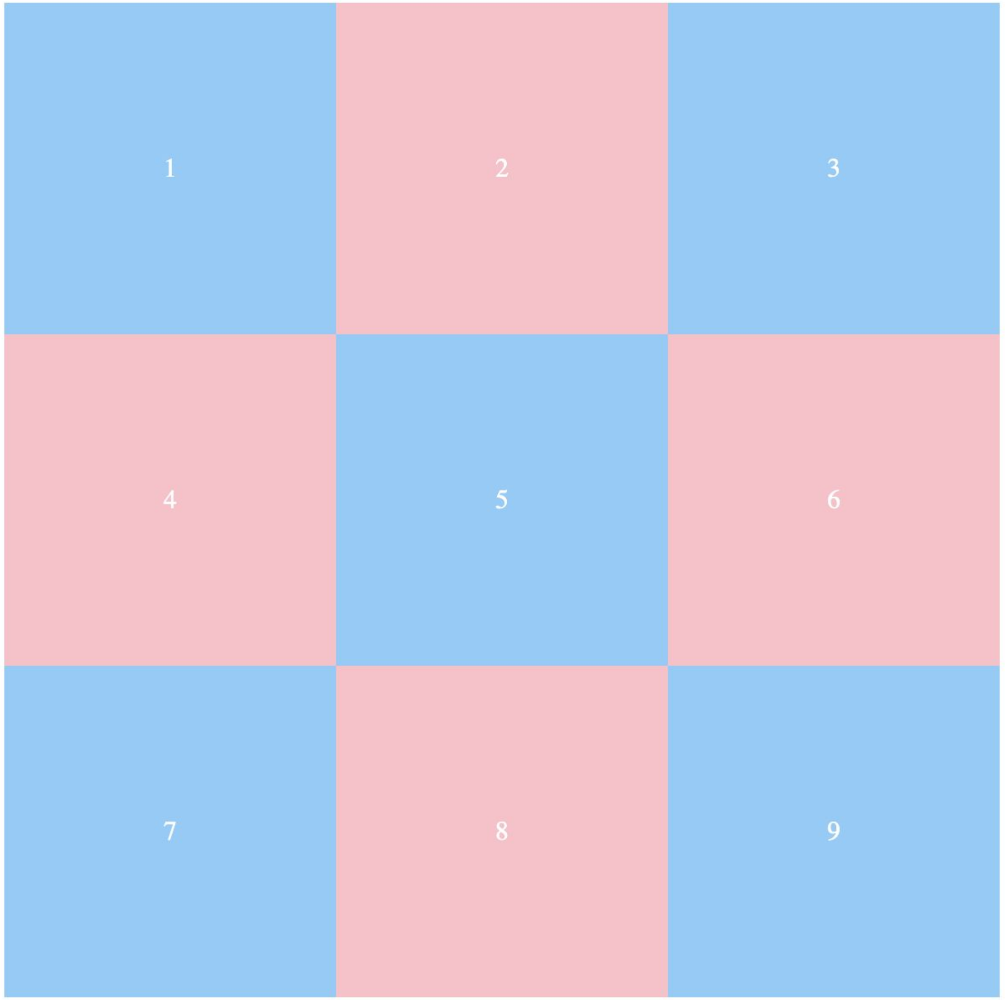
8

9

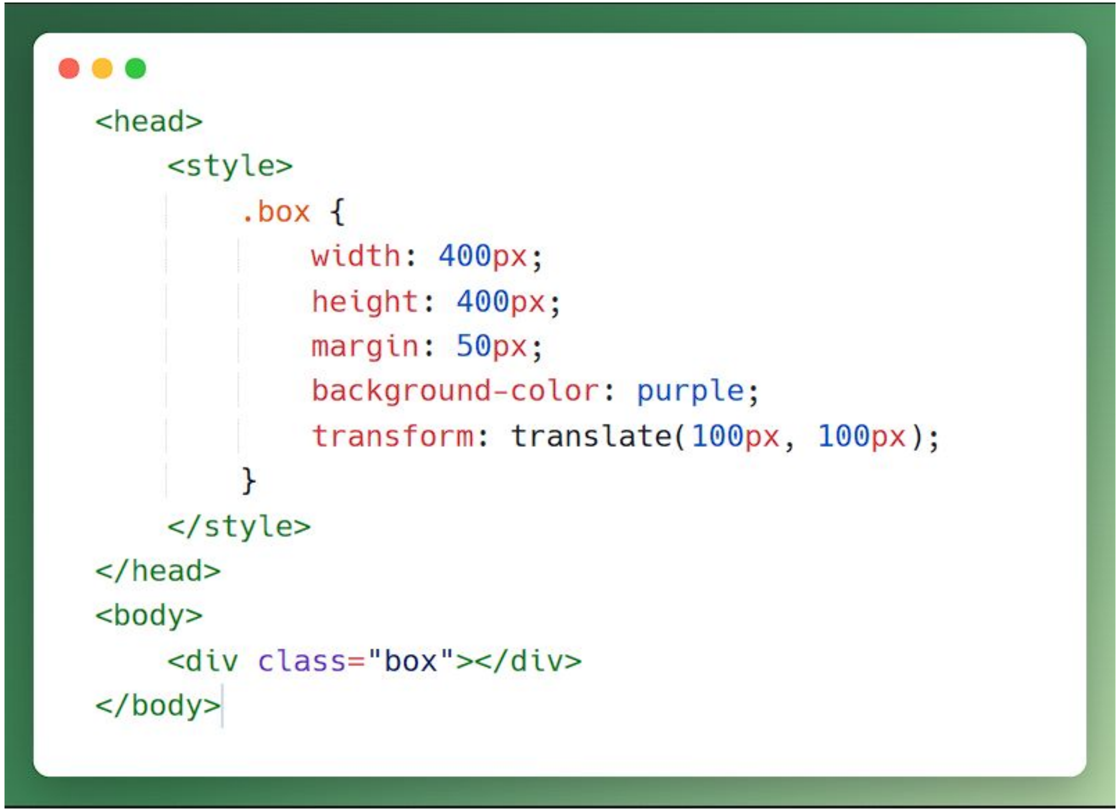
Flex Example



```
#content {
    width: 600px;
    height: 600px;
    margin: 0;
    display: flex;
    flex-wrap: wrap;
}
#content div {
    background: lightskyblue;
    justify-content: center;
    align-items: center;
    display: flex;
    width: 33.33%;
    height: 33.33%;
}
```

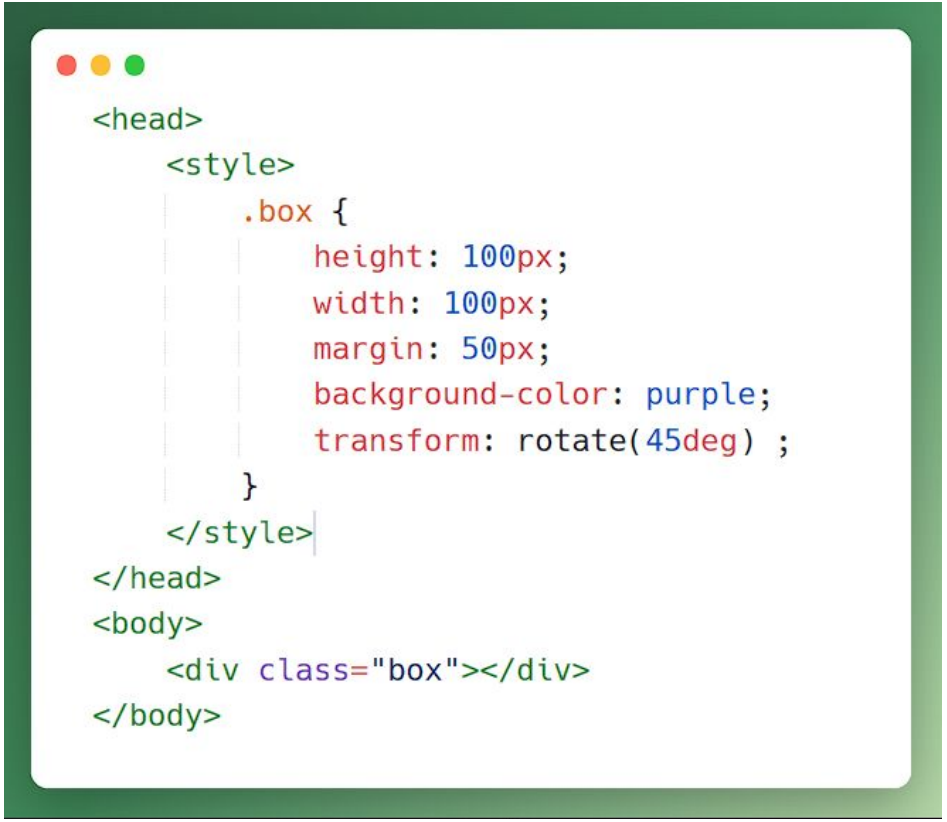


Animation - 2D Transform - Translate



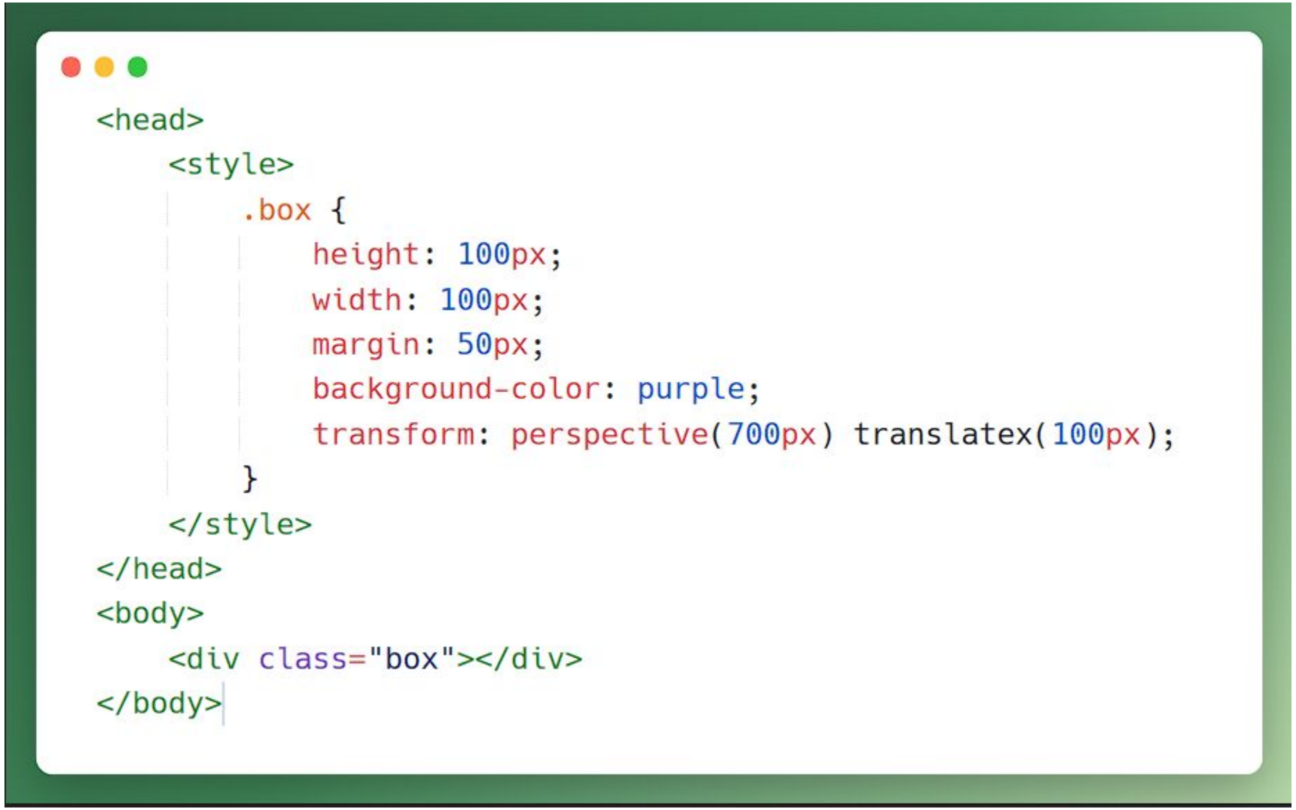
```
<head>
    <style>
        .box {
            width: 400px;
            height: 400px;
            margin: 50px;
            background-color: purple;
            transform: translate(100px, 100px);
        }
    </style>
</head>
<body>
    <div class="box"></div>
</body>
```

Animation - 2D Transform - Rotate



```
<head>
    <style>
        .box {
            height: 100px;
            width: 100px;
            margin: 50px;
            background-color: purple;
            transform: rotate(45deg) ;
        }
    </style>
</head>
<body>
    <div class="box"></div>
</body>
```

Animation - 3D Transforms - TranslateX



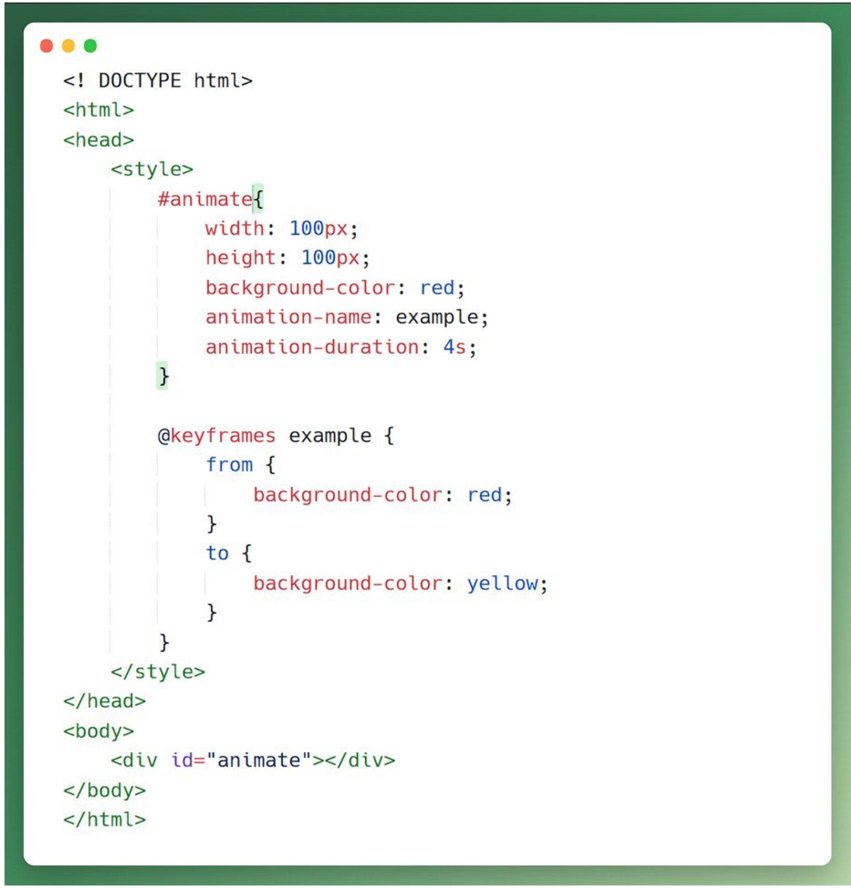
```
<head>
    <style>
        .box {
            height: 100px;
            width: 100px;
            margin: 50px;
            background-color: purple;
            transform: perspective(700px) translatex(100px);
        }
    </style>
</head>
<body>
    <div class="box"></div>
</body>
```

Animation - @keyframes rule

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

To get an animation to work, you must bind the animation to an element.

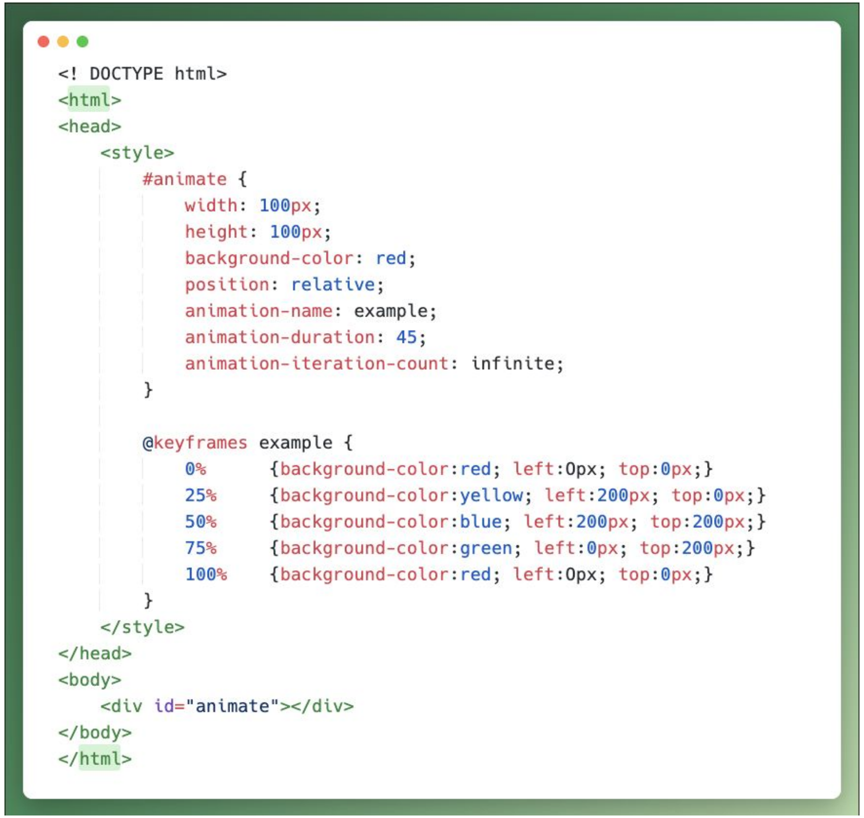
The following example binds the "example" animation to the <div> element. The animation will last for 4 seconds, and it will gradually change the background-color of the <div> element from "red" to "yellow":



```
<! DOCTYPE html>
<html>
<head>
    <style>
        #animate{
            width: 100px;
            height: 100px;
            background-color: red;
            animation-name: example;
            animation-duration: 4s;
        }

        @keyframes example {
            from {
                background-color: red;
            }
            to {
                background-color: yellow;
            }
        }
    </style>
</head>
<body>
    <div id="animate"></div>
</body>
</html>
```

Animation - @keyframes rule

The following example uses the value "infinite" to make the animation continue forever:



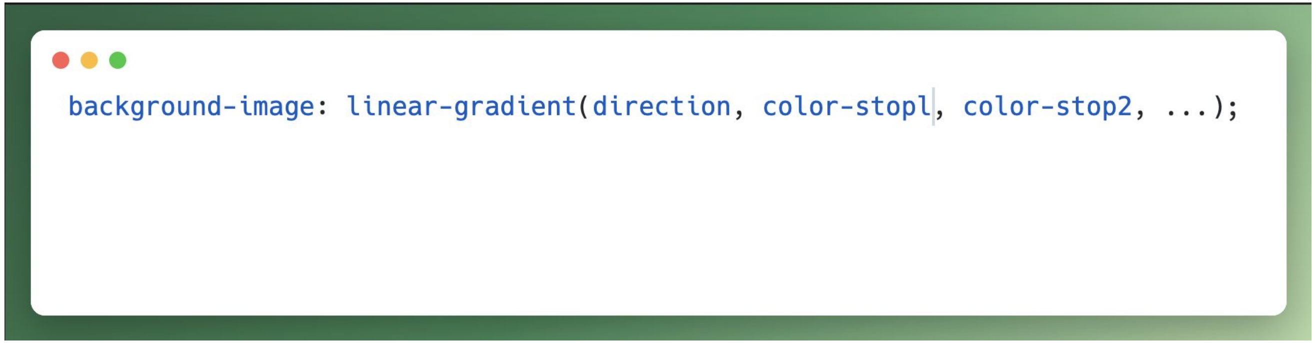
```
<! DOCTYPE html>
<html>
<head>
    <style>
        #animate {
            width: 100px;
            height: 100px;
            background-color: red;
            position: relative;
            animation-name: example;
            animation-duration: 45;
            animation-iteration-count: infinite;
        }

        @keyframes example {
            0%      {background-color:red; left:0px; top:0px;}
            25%     {background-color:yellow; left:200px; top:0px;}
            50%     {background-color:blue; left:200px; top:200px;}
            75%     {background-color:green; left:0px; top:200px;}
            100%    {background-color:red; left:0px; top:0px;}
        }
    </style>
</head>
<body>
    <div id="animate"></div>
</body>
</html>
```

Gradient

CSS gradients let you display smooth transitions between two or more specified colors.

CSS defines three types of gradients:

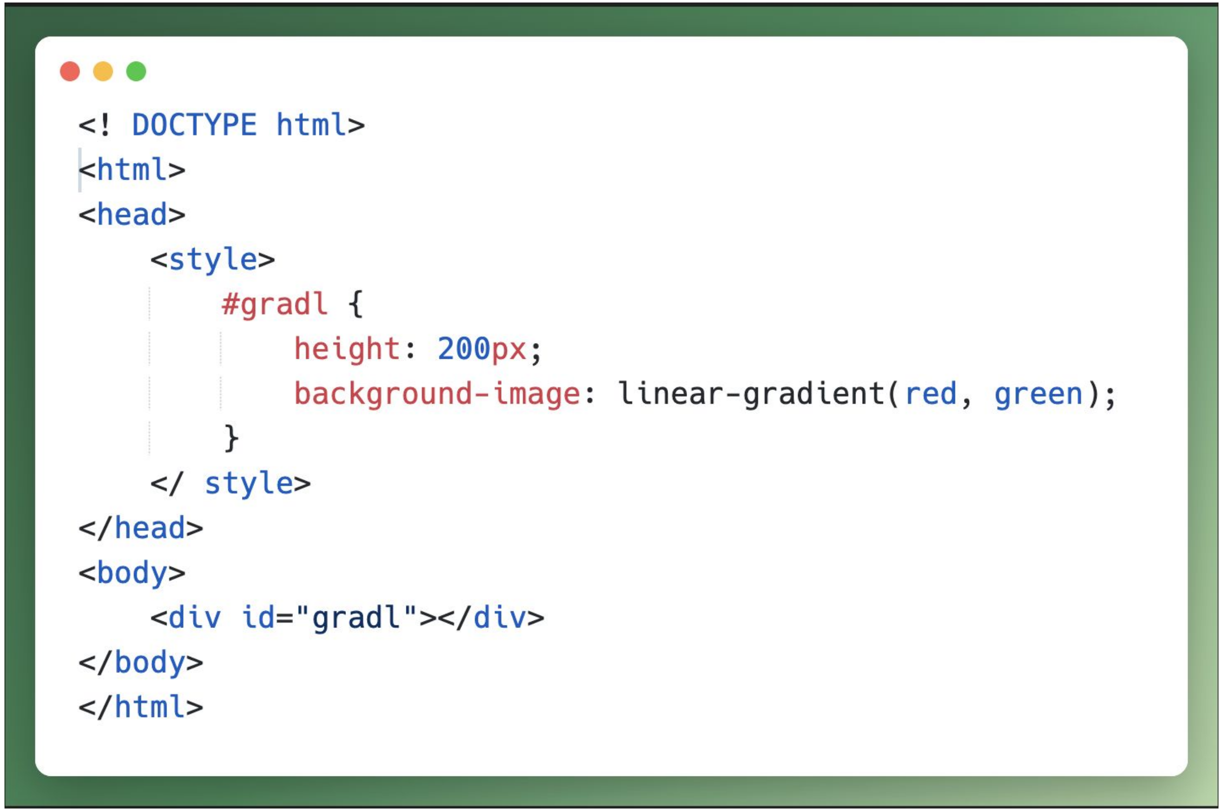
- Linear Gradients (goes down/up/left/right/diagonally)
- Radial Gradients (defined by their center)
- Conic Gradients (rotated around a center point)



```
background-image: linear-gradient(direction, color-stop1, color-stop2, ...);
```

Gradient - Example

The following example shows a linear gradient that starts at the top. It starts red, transitioning to green:



```
<! DOCTYPE html>
<html>
<head>
    <style>
        #gradl {
            height: 200px;
            background-image: linear-gradient(red, green);
        }
    </ style>
</head>
<body>
    <div id="gradl"></div>
</body>
</html>
```

Exercise

Create a component with styling like this figma design :

- https://www.figma.com/design/VLtlf6F1uCUebVNxMZuH3b/product-preview-card-component
- https://www.figma.com/design/uxJkwlwl4jQdC12K2FMcgo/testimonials-grid-section

Thank you