

# CSS Specificity, Properties, and Other things

# CSS Specificity

order/hierarchy of CSS rules

# Style Sheet Sources

The cascade of the rules start from the ***farthest*** to the ***closest*** to the element.

1. User-agent (browser) default style sheets
2. External style sheets
3. Internal/Embedded style sheets
4. Inline style sheets

# CSS Selectors

**element**

**.class**

**#id**

**Combinators**

**Attribute Selectors**

**Pseudo-classes & Pseudo-elements**

# CSS Specificity

The styling declared in CSS cascade depending on the **source** (*where it was declared*) and the **selector** used.

Good reading: [CSS Specificity Wars](#) (Star Wars themed explanation)

# Specificity **Hierarchy** Groups

1. **Inline styles.** Anything placed in the style attribute of the element has precedence over anything.
2. **IDs.** Unique identifier for an element (#div)
3. **Classes, attributes, and pseudo-classes.** Includes .class, [attributes], :hover, :focus, etc.
4. **Elements (and pseudo-elements).** For example :before, :after, :first-line

# Specificity Hierarchy

For every selector, you give the following points.

Selector	Points
<b>style=""</b>	1000
<b>#id</b>	100
<b>.class</b>	10
<b>element</b>	1

# CSS Specificity

	inline	#id	.class	element
*	0	0	0	0
li	0	0	0	1
li::first-line	0	0	0	2
ul li	0	0	0	2
ul ol+li	0	0	0	3



# CSS Specificity

`h1 + *[rel=up]`

`ul ol li.red`

`li.red.level`

`style=""`

`div p`

inline	#id	.class	element
0	0	1	1
0	0	1	3
0	0	2	1
1	0	0	0
0	0	0	2

# CSS Specificity

`div p.special`

`#sith`

`body #id .sith p`

`#p #p1 #p2 p.cool`

`p:not(.cool)`

:not is a pseudo class  
with no points!

inline	#id	.class	element
0	0	1	2
0	1	0	0
0	1	1	2
0	2	1	1
0	0	0	1

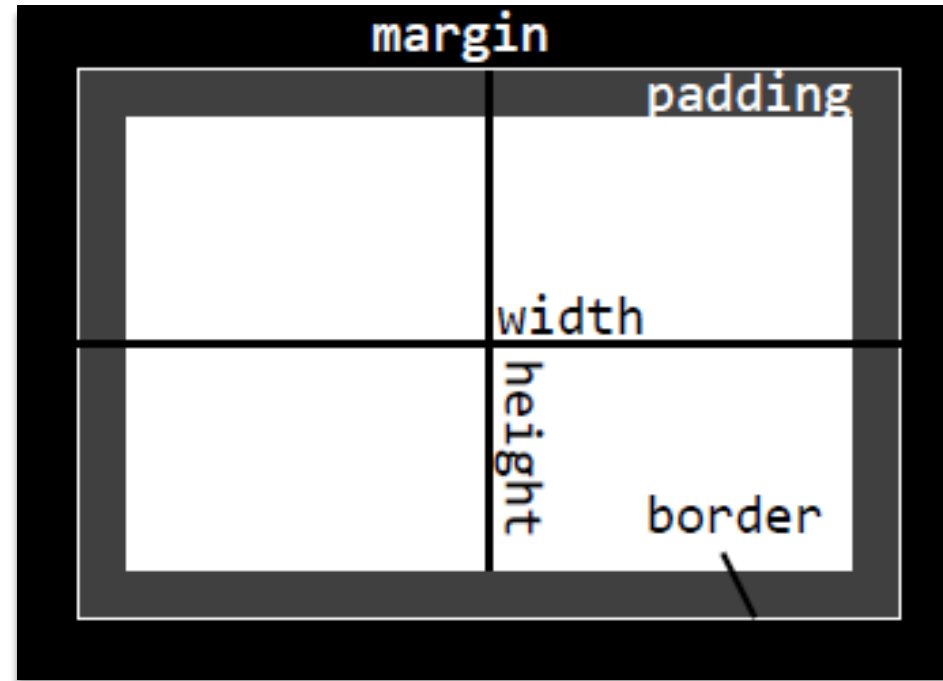
# CSS Properties

going through a list of common properties

# Layout Properties

# Basic Box Model

- height
- width
- float
- clear
- border
- padding
- margin



The CSS basic box model

All elements are represented as a rectangular box following the basic box model.

# height and width

height & width and related properties can be assigned either of the following values:

- auto (behavior depends on the property)
- absolute values
  - px (relative to viewing device's pixel density),
  - cm, mm, in, pt (1/72 of in), pc (12 pt)
- relative values
  - em (relative to font-size of the element; 2em means 2x font size)
  - ch (relative to the width of "0")
  - rem (relative to font-size of the root element)
  - vw, vh, vmin, vmax (relative to viewport dimensions)
  - % (relative to the size of the containing block)
- inherit – Inherits the property from the parent element

# min- and max-width

## min-width

- Sets the **minimum width** of an element.
- Will be applied if the content is smaller than the minimum width, else it has no effect

## max-width

- Sets the **maximum width** of an element.
- Will be applied if the content is larger than the maximum width, else it has no effect

# min- and max-height

## min-height

- Sets the **minimum height** of an element.
- Will be applied if the content is smaller than the minimum height, else it has no effect

## max-height

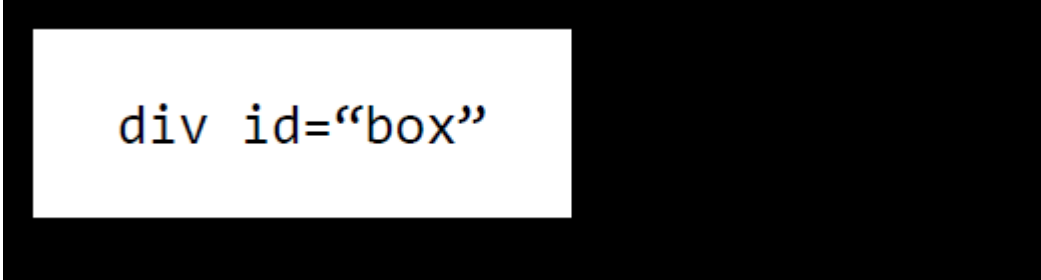
- Sets the **maximum height** of an element.
- Will be applied if the content is larger than the maximum height, else it has no effect



# float

**float** (can be left/right/none/inherit)

Makes elements float to the right or left of the screen, positioned where they are in the HTML.



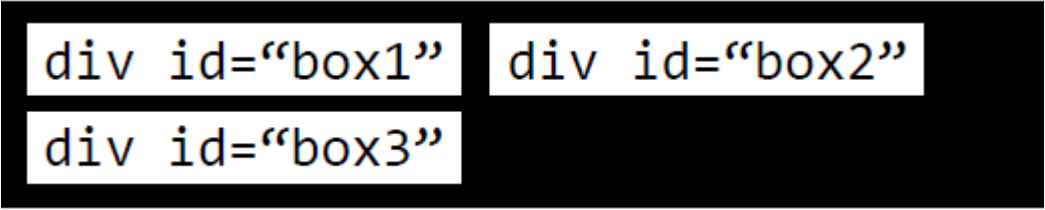
```
div id="box"
```

```
#box {  
  float: left;  
  margin-right: 10px;  
}
```

# clear

## clear

Specifies what elements can float beside the cleared element and on which side.



The diagram shows three white rectangular boxes with black text, arranged in a 2x2 grid. The top row contains 'div id="box1"' and 'div id="box2"'. The bottom row contains 'div id="box3"' and an empty space. All boxes are set against a solid black background.

```
#box1 { float: left; }
```

```
#box2 { float: left; }
```

```
#box3 { clear: both; }
```

# clear

## **clear**

- `none` – default. Allows floating elements on both sides
- `left` – no floating elements allowed on the left side
- `right` – no floating elements allowed on the right side
- `both` – no floating elements allowed on either side
- `inherit` – value will be inherited from parent

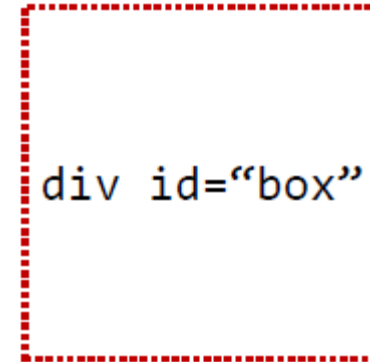
# border

## border

Specifies the style, width, and color of an element's border.

Also has a shorthand property, `border`

```
#box {  
    border-width: 1px;  
    border-style: dotted;  
    border-color: red;  
}
```



`border-top`, `border-bottom`, `border-left`, `border-right` can be used to style each border line manually.

**border documentation:**

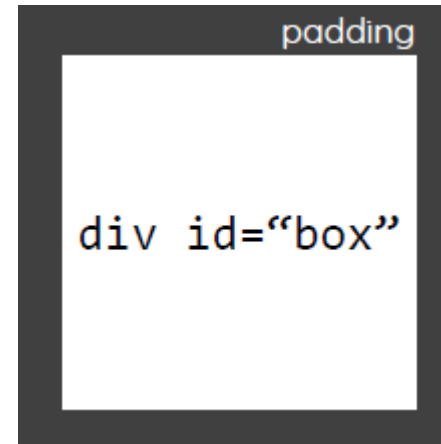
<https://developer.mozilla.org/en-US/docs/Web/CSS/border>

# padding

## padding

Generates space around an element's content, inside of any defined borders.

```
#box {  
    padding: 5px;  
}
```



padding-top, padding-left, padding-right, padding-bottom can be used to adjust each padding manually.

# padding

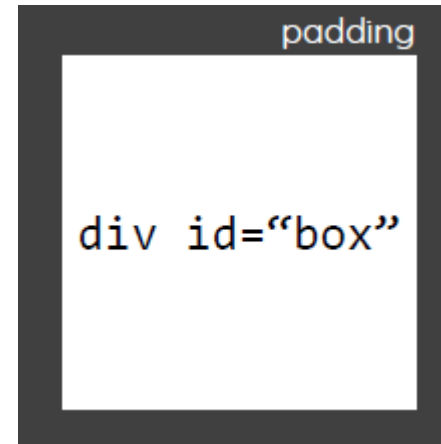
## padding

(top & bottom, right & left)

```
#box {  
    padding: 5px 5px;  
}
```

(top, right & left, bottom)

```
#box {  
    padding: 5px 5px 5px;  
}
```

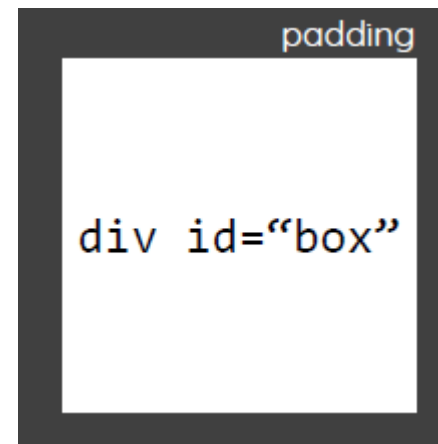


# padding

## padding

(top, right, bottom, left)

```
#box {  
    padding: 5px 5px 5px 5px;  
}
```

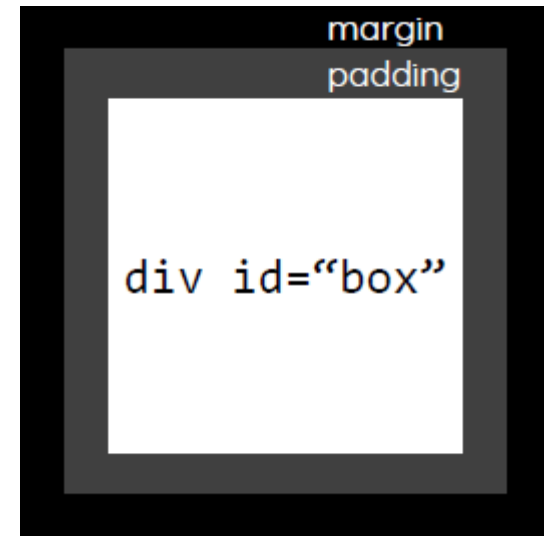


# margin

## margin

Create space around elements, outside of any defined borders

```
#box {  
    margin: 5px;  
}
```



margin-top, margin-left, margin-right, margin-bottom can be used to adjust each margin manually.



# display

## display

Sets whether an element is to be treated as a block or an inline element, and the layout used for its children as well.

default for most elements

`display: block;`

`<span>` default

`display: inline;`

Can also be set to `flex` and `grid` for the element to adhere to the **flexbox** and **grid** model, respectively.

**display** documentation:

<https://developer.mozilla.org/en-US/docs/Web/CSS/display>

# Position

## **position**

Sets how an element is positioned in a document. Possible values are:

- static
- relative
- absolute
- fixed
- sticky

Related properties:

**top**, **left**, **right**, and **bottom** – determines the final location of the element. May behave differently based on the value of **position**.

**position** documentation:

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

# Position

**static**

```
#greendiv {  
  position: static;  
  width: 100px;  
  height: 100px;  
}
```

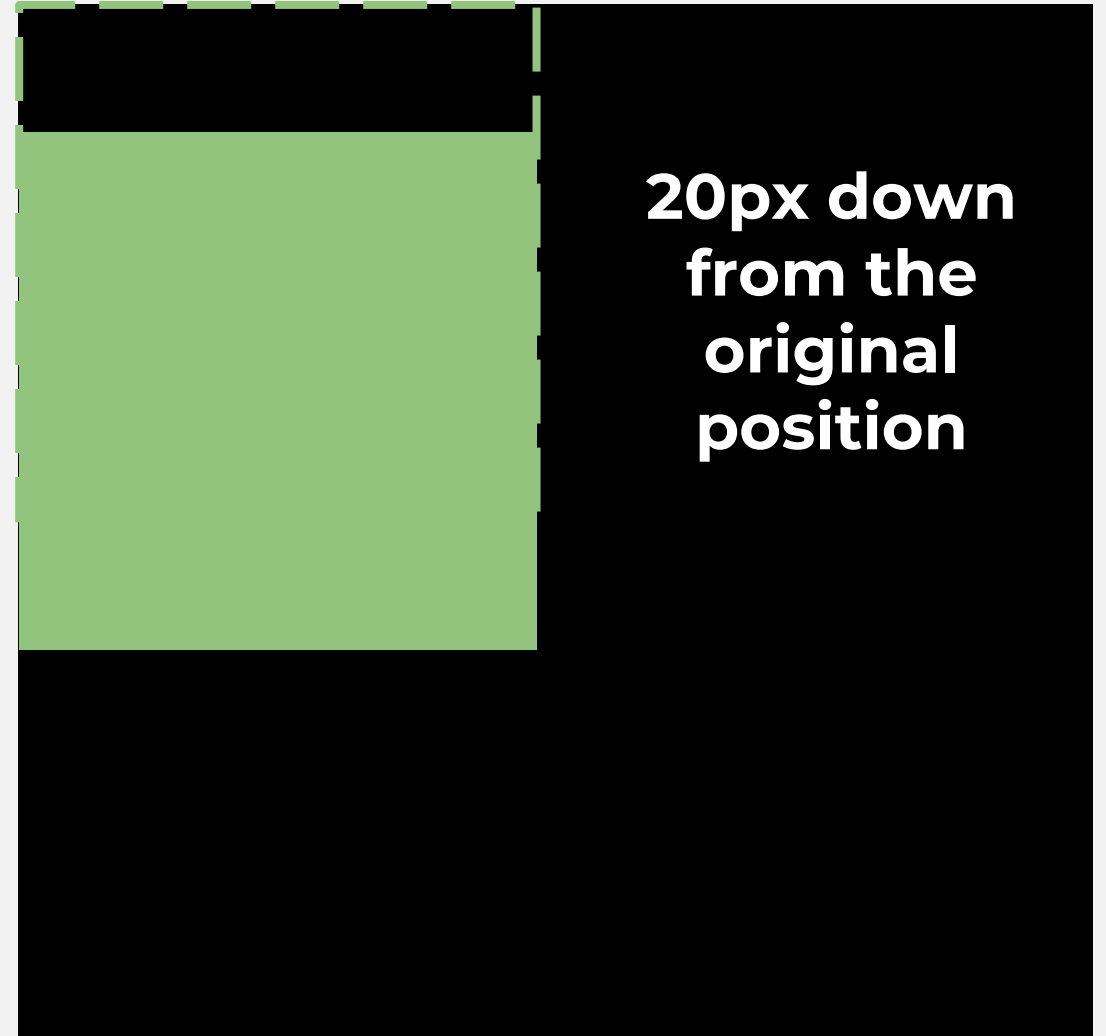


**This is the default.**

# Position

## relative

```
#greendiv {  
  position: relative;  
  width: 100px;  
  height: 100px;  
  top: 20px;  
}
```

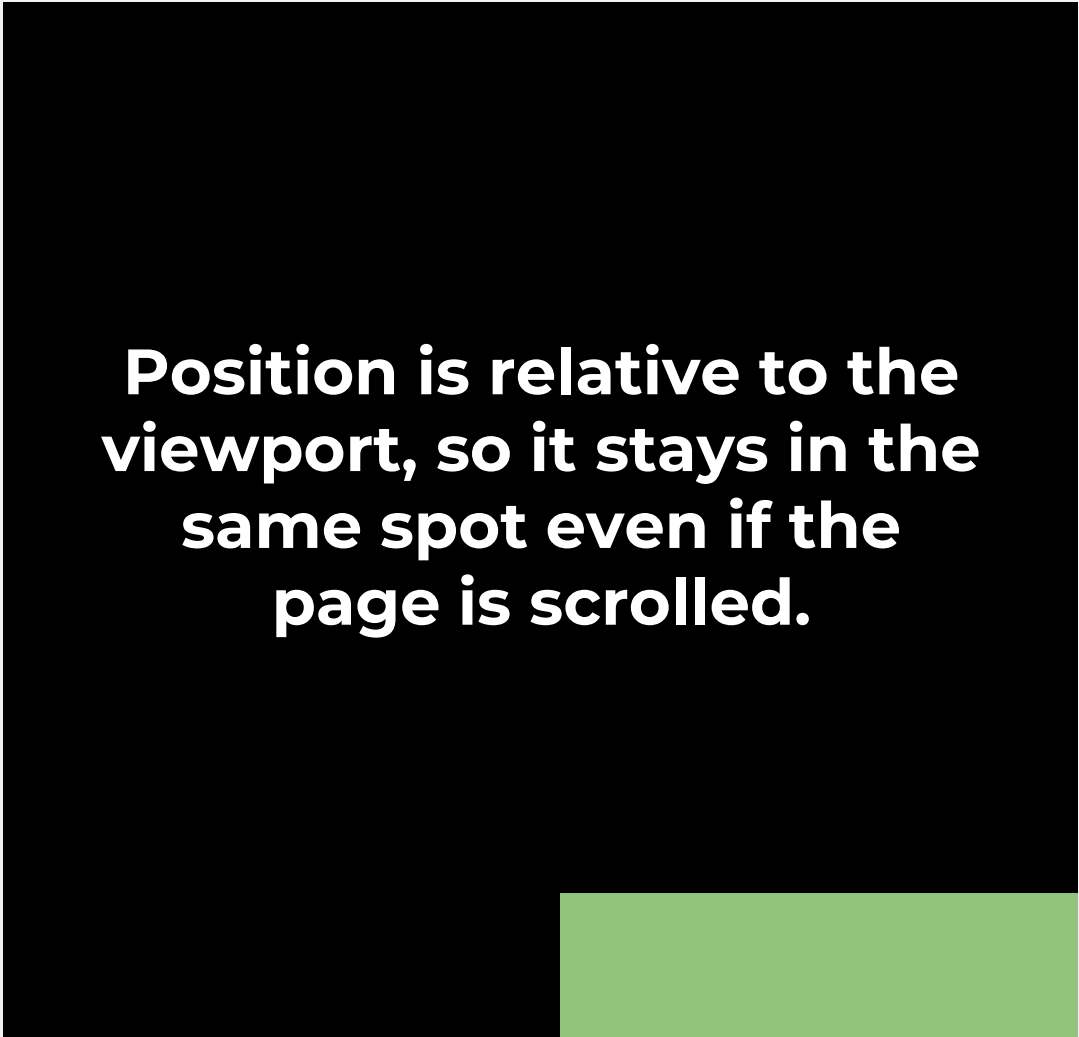


# Position

**fixed**

```
#greendiv {  
  position: fixed;  
  width: 100px;  
  height: 10px;  
  bottom: 0;  
  right: 0;  
}
```


**Position is relative to the viewport, so it stays in the same spot even if the page is scrolled.**



# Position

## absolute

```
#greendiv {  
  position: absolute;  
  width: 100px;  
  height: 100px;  
  right: 0px;  
}
```

A diagram illustrating absolute positioning. It features a large black rectangle representing a container. In the top-right corner of this container, there is a smaller green rectangle. This visualizes how an element with 'position: absolute' is placed exactly where specified, regardless of the normal document flow.

**Removes the div from the order of things and places it exactly where it is specified to go.**

Note: float and clear will not work properly with absolute

# Position

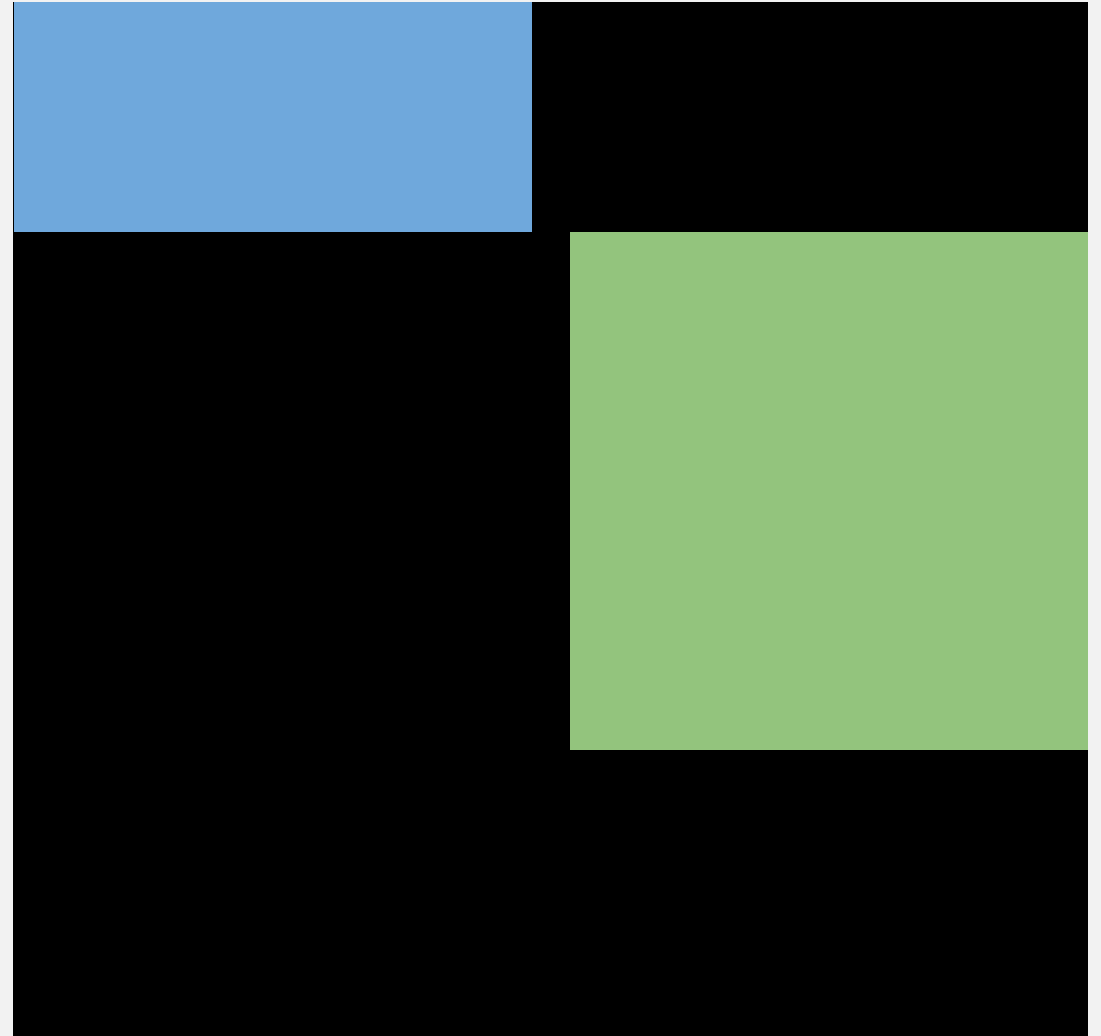
```
#bluediv {  
  width: 100px;  
  height: 30px;  
  right: 0px;  
}
```

```
#greendiv {  
  width: 100px;  
  height: 100px;  
}
```



# Position

```
...  
#greendiv {  
  position: absolute;  
  width: 100px;  
  height: 100px;  
  right: 0px;  
}
```





# Position

```
...  
#greendiv {  
  position: absolute;  
  width: 100px;  
  height: 100px;  
  right: 0px;  
  top: 0px;  
}
```



# Position

## sticky

```
#greendiv {  
  position: sticky;  
  width: 200px;  
  height: 10px;  
  top: 0px;  
}
```

I am a paragraph placed above this sticky #greendiv....




More text after the #greendiv! There's a bunch more stuff down here that makes the user have to scroll the page.

# Position

## sticky

```
#greendiv {  
  position: sticky;  
  width: 200px;  
  height: 10px;  
  top: 0px;  
}
```



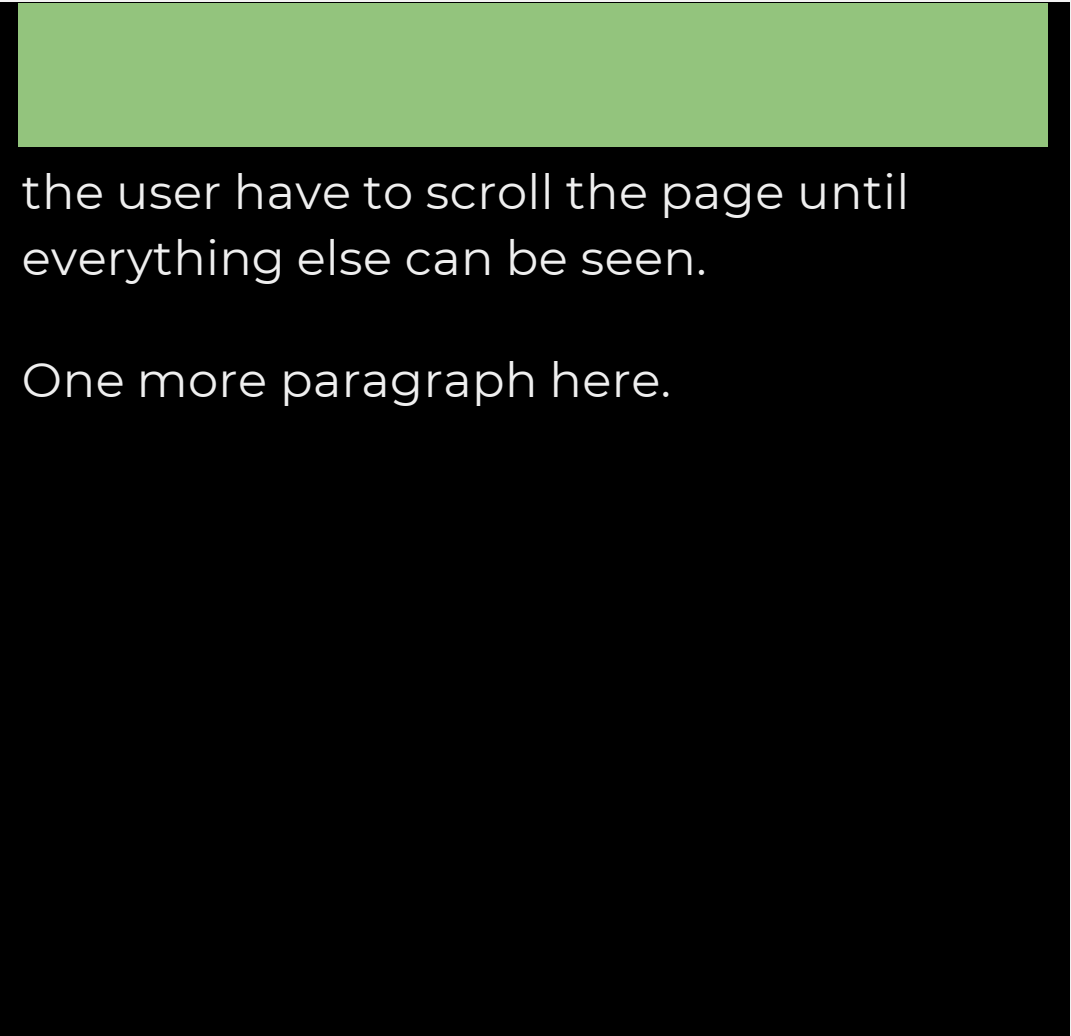
More text after the #greendiv! There's a bunch more stuff down here that makes the user have to scroll the page until everything else can be seen.

One more paragraph here.

# Position

## sticky

```
#greendiv {  
  position: sticky;  
  width: 200px;  
  height: 10px;  
  top: 0px;  
}
```



the user have to scroll the page until everything else can be seen.

One more paragraph here.

element will **stick** on the screen - useful for navigation headers

# Flexbox

Adhered model when `display` is set to `flex`.

This layout allowed for an easier way to design a flexible and responsive structure without using **float** and **position**.

flex documentation:

[https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Flexible\\_Box\\_Layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout)

# Flex

Suppose we are starting with the following setup:

```
.container {  
  padding: 5px;  
  background-color: teal;  
  height: 300px;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



# Flex

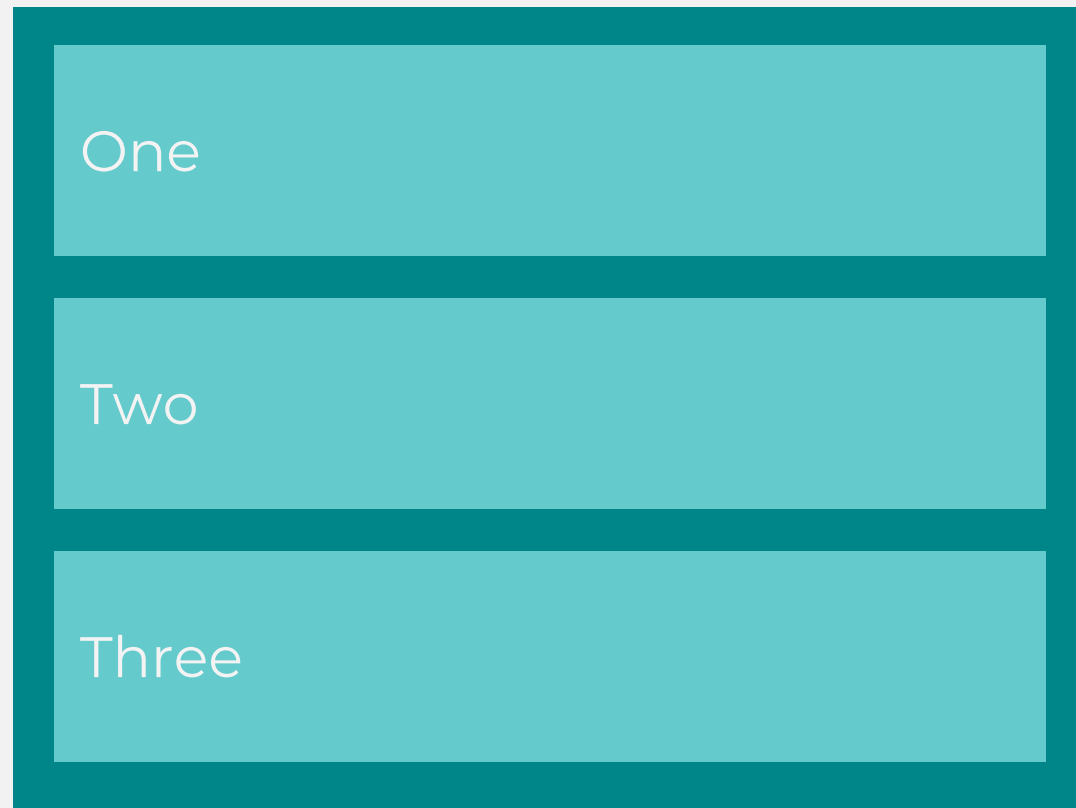
```
.container {  
  padding: 5px;  
  background-color: teal;  
  height: 300px;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



*What if we want the unit divs  
to adjust evenly according to  
the parent container div?*

# Flex

```
.container {  
  display: flex  
  padding: 5px;  
  background-color: teal;  
  height: 300px;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```

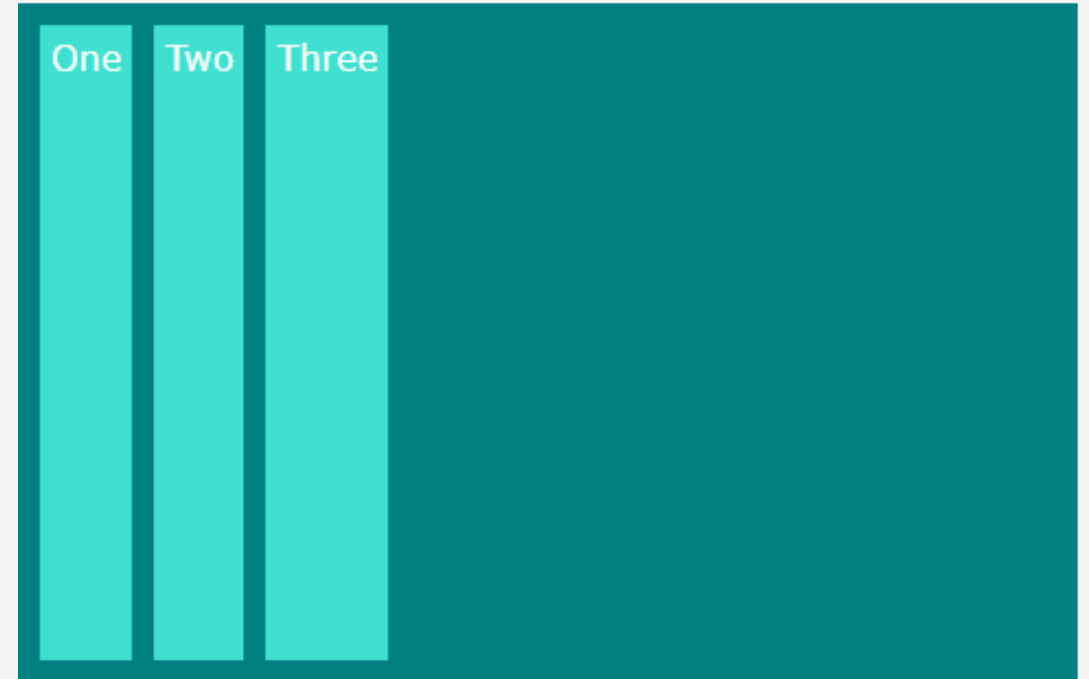


**goal**



# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  height: 300px;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



**actual result**

*This is because the default direction of the flex .container is “row”, making .unit ‘inline’*

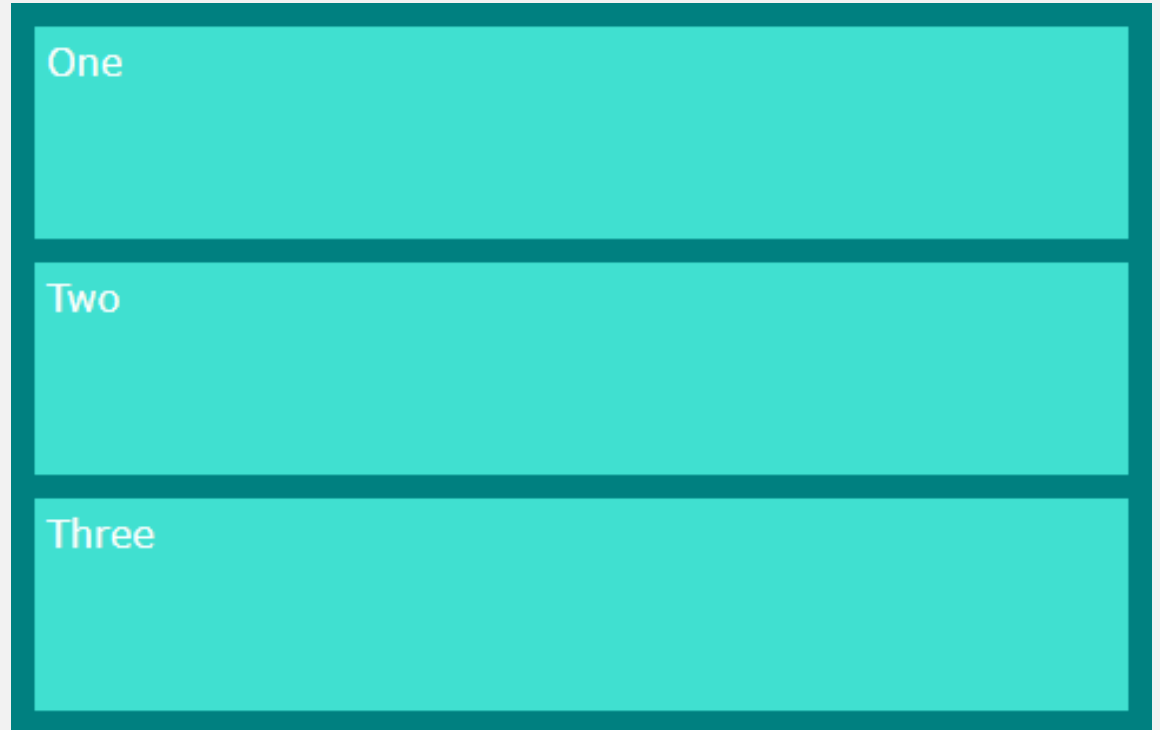
# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  height: 300px;  
  flex-direction: column;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



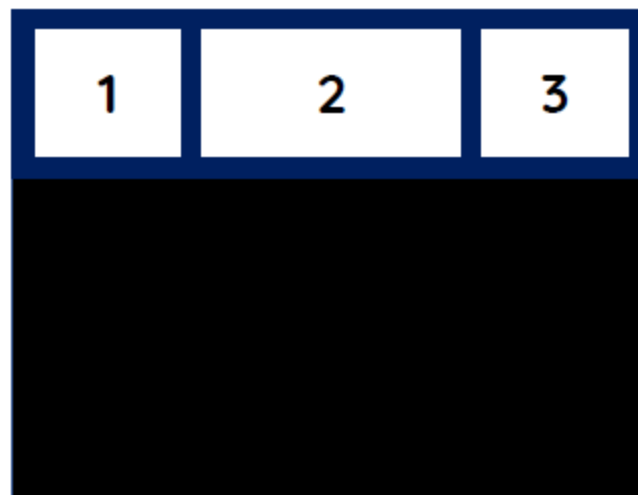
# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  height: 300px;  
  flex-direction: column;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
  flex-grow: 1;  
}
```

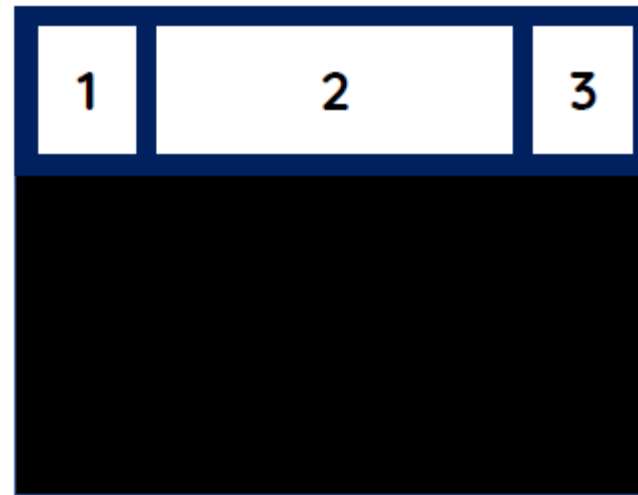


**flex-grow** can also be used per item to grow them in terms of priority.

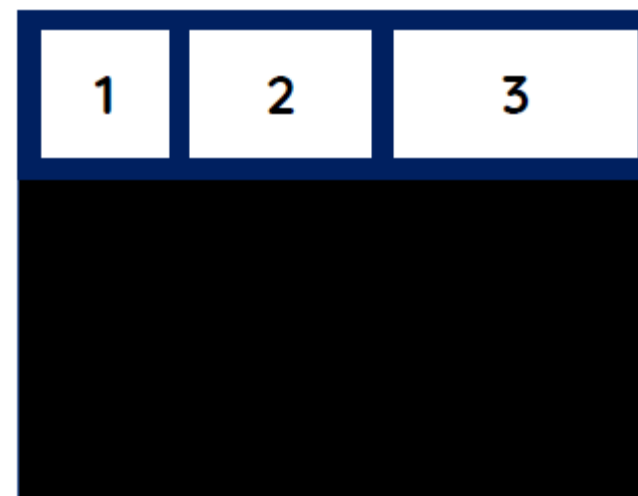
```
#unit-1 { flex-grow: 1; }  
#unit-2 { flex-grow: 2; }  
#unit-3 { flex-grow: 1; }
```



```
#unit-1 { flex-grow: 1; }  
#unit-2 { flex-grow: 4; }  
#unit-3 { flex-grow: 1; }
```



```
#unit-1 { flex-grow: 1; }  
#unit-2 { flex-grow: 2; }  
#unit-3 { flex-grow: 3; }
```



# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  flex-direction: row;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



*If the viewport and the container cannot fit the units, the units will overflow.*

# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  flex-direction: row;  
  flex-wrap: wrap;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```

One banana banana banana

Two monkeys monkeys monkeys

Three cute puppies puppies puppies

fourcute puppies puppies puppies

five cute puppies puppies puppies

six cute puppies puppies puppies

*flex-wrap: wrap; will  
automatically move the units  
to the next row/column.  
(By default, the value is nowrap)*



# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  flex-direction: row;  
flex-wrap: wrap;  
  justify-content: center;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



*justify-content aligns items inside the flex container along its **main axis**.  
flex-direction: row => horizontally  
flex-direction: column => vertically*

# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  flex-direction: row;  
  justify-content: flex-start;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



One Two Three

# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  flex-direction: row;  
  justify-content: flex-end;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



One Two Three

# Flex

```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  flex-direction: row;  
  justify-content: space-evenly;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



Spaces in between the items and the container are even

# Flex

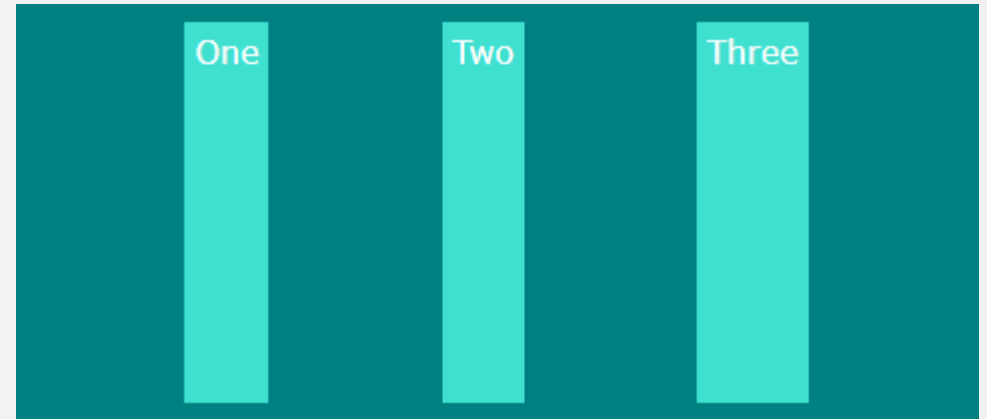
```
.container {  
  display: flex;  
  padding: 5px;  
  background-color: teal;  
  flex-direction: row;  
  justify-content: space-around;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



Spacing between each pair of adjacent items is the same. The empty space before the first and after the last item is half the space between each pair of adjacent item.

# Flex

```
.container {  
  display: flex;  
  height: 300px;  
  padding: 5px;  
  background-color: teal;  
  flex-direction: row;  
  justify-content: space-between;  
  align-items: stretch;  
}  
  
.unit {  
  margin: 5px;  
  padding: 5px;  
  background-color: turquoise;  
  color: white;  
}
```



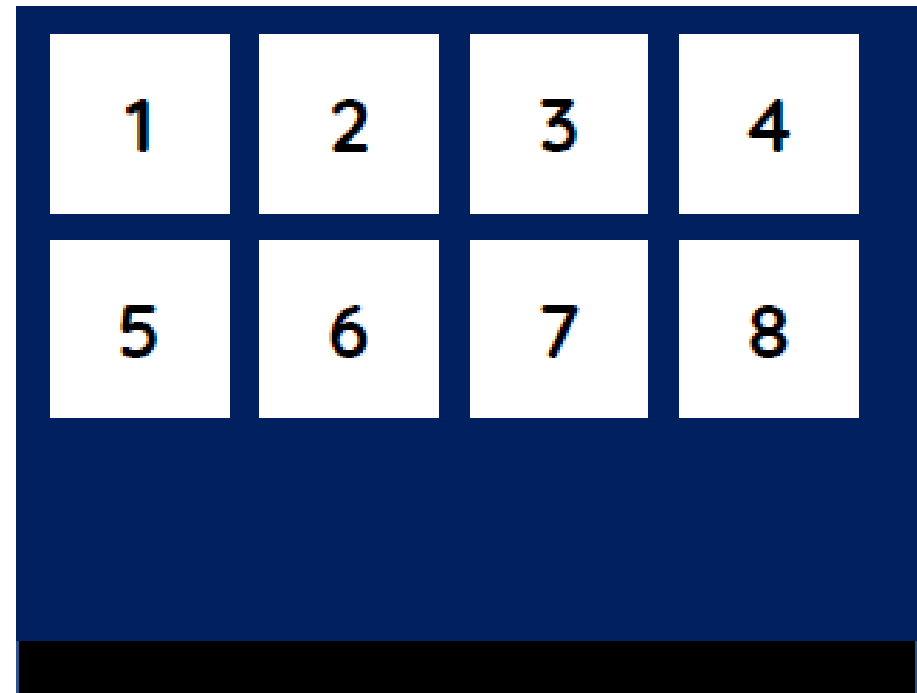
**align-items** aligns the items of a flex container along its **cross axis**. It is set to **stretch** by default. Same options are available for align-items

- flex-direction: row => vertically*
- flex-direction: column => horizontally*

# ALIGN CONTENT

Aligns flex lines instead of flex items

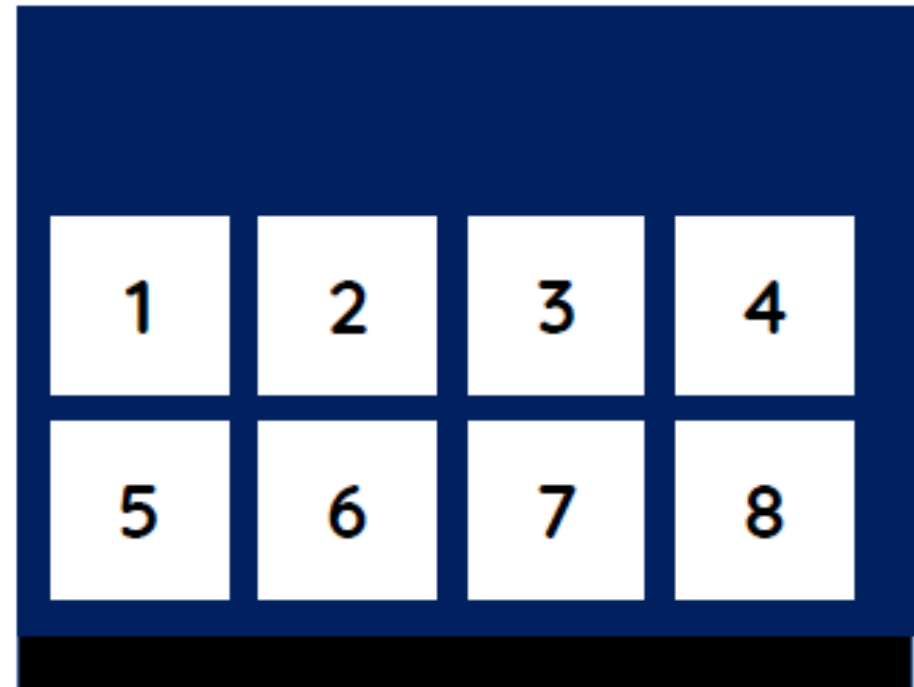
```
#container {  
  display: flex;  
  flex-wrap: wrap;  
  height: 400px;  
  align-content: flex-start;  
  background-color: #000285;  
}
```



# ALIGN CONTENT

Aligns flex lines instead of flex items

```
#container {  
  display: flex;  
  flex-wrap: wrap;  
  height: 400px;  
  align-content: flex-end;  
  background-color: #000285;  
}
```





# Text & Font

# Text

- With CSS Text, it is possible to style the ff:
  - color / background color (color and background-color)
  - alignment (text-align, text-align-last, direction, etc.)
  - decoration (text-decoration, text-emphasis, text-shadow, etc.)
  - transform (Using text-transform: uppercase/lowercase, language-specific case mapping, etc.)
  - spacing (text-indent, letter-spacing, line-height, white-space, etc.)
  - etc.

CSS Text:

[https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Text](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Text)

[https://www.w3schools.com/css/css\\_text.asp](https://www.w3schools.com/css/css_text.asp)

# Text

Possible color value formats:

- name – specify a color name like “blue”
- hex – specify a hex value like “#0000ff”
- RGB – specify an RGB value like “rgb(0, 0, 255)”

CSS Text:

[https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Text](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Text)

[https://www.w3schools.com/css/css\\_text.asp](https://www.w3schools.com/css/css_text.asp)

# Fonts

- With CSS Font, it is possible to style the ff:
  - font-family
  - font-size
  - font-stretch
  - font-style
  - font-weight
  - etc.

You can use a single property, `font`, which is shorthand property for the other different properties.

CSS Fonts:

[https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Fonts](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Fonts)

[https://www.w3schools.com/css/css\\_font.asp](https://www.w3schools.com/css/css_font.asp)

**font** syntax:

<https://developer.mozilla.org/en-US/docs/Web/CSS/font#syntax>

# Text & Fonts – Additional Fonts

**Google Fonts:** <https://fonts.google.com/>

Easy, hassle-free way to add different “free” fonts

## Importing example:

- via <link>:

```
<link rel="preconnect" href="https://fonts.googleapis.com">  
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>  
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap"  
rel="stylesheet">
```

- via @import: (you can place this inside your .css file as well)

```
<style>  
@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap');  
</style>
```

For importing locally sourced fonts, read:

<https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face>

# Lists & Tables

# Lists

- With CSS lists, you can style the ff:
  - styling for text also applies
  - bullets styles (basic and custom)
  - padding
  - background
  - etc.

You can also use a shorthand property, `list-style`, for bullet- or numbering-related styling

**For an overview of what else you can do with lists:**

[https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling\\_text/Styling\\_lists](https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text/Styling_lists)

# Tables

- With CSS tables, you can style the ff:
  - texts and fonts also apply
  - border
  - padding
  - background
  - etc.

**For an overview of what else you can do with tables:**

[https://developer.mozilla.org/en-US/docs/Learn/CSS/Building\\_blocks/Styling\\_tables](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Styling_tables)



# Other Things

Variables, Sprites, Responsive Design, and Animations

# Variables

It is possible to create variables in CSS3. To declare a variable in a global scope:

```
:root {  
  --primary: #008688;  
  --accent: #64cacc;  
}
```

:root is a pseudo-class which is the root of the HTML document.

-- indicates a variable is being declared.

## **To use the variable:**

```
var(--primary)
```

## **For example:**

```
color: var(--primary);
```

# Sprites

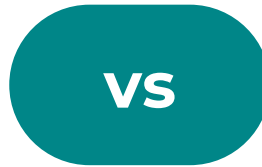
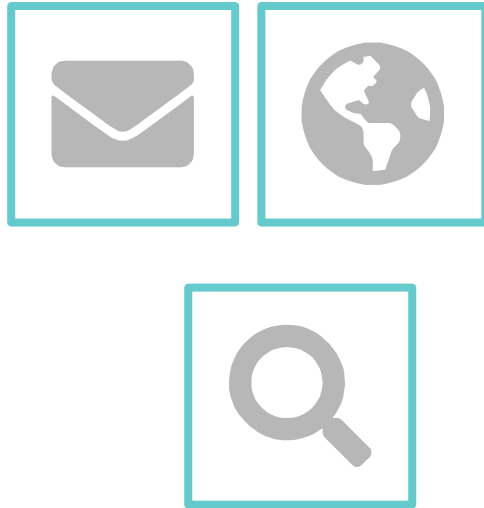


# Sprites

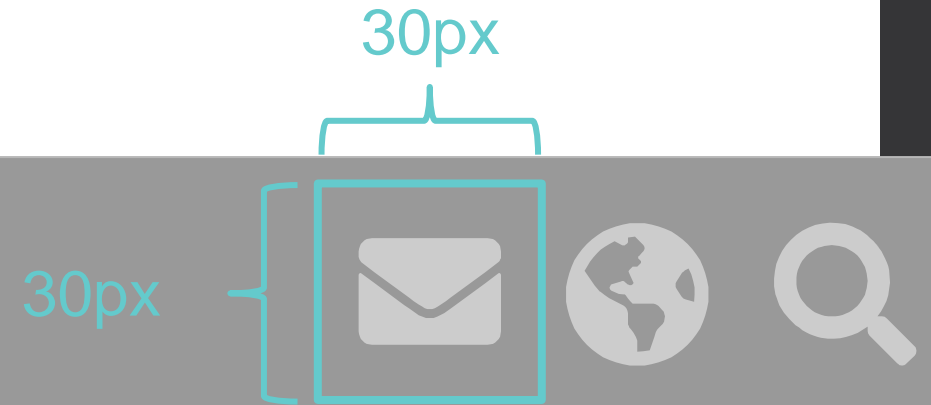


**You are downloading these images individually.  
A separate request is made for each item or  
resource you add on your page.**

# Sprites

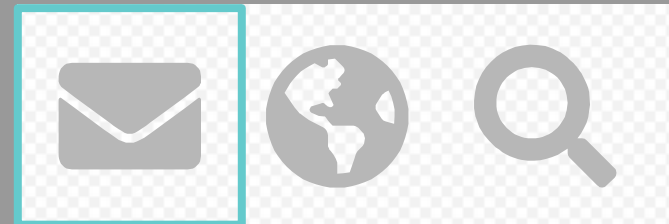


# Sprites



```
<ul>  
  <li id="mail"></li>  
  <li id="notif"></li>  
  <li id="search"></li>  
</ul>
```

# Sprites



```
#mail {  
    background-image: url('sprite.png');  
    width: 30px;  
    height: 30px;  
    background-position: 0px 0px;  
}
```

# Sprites



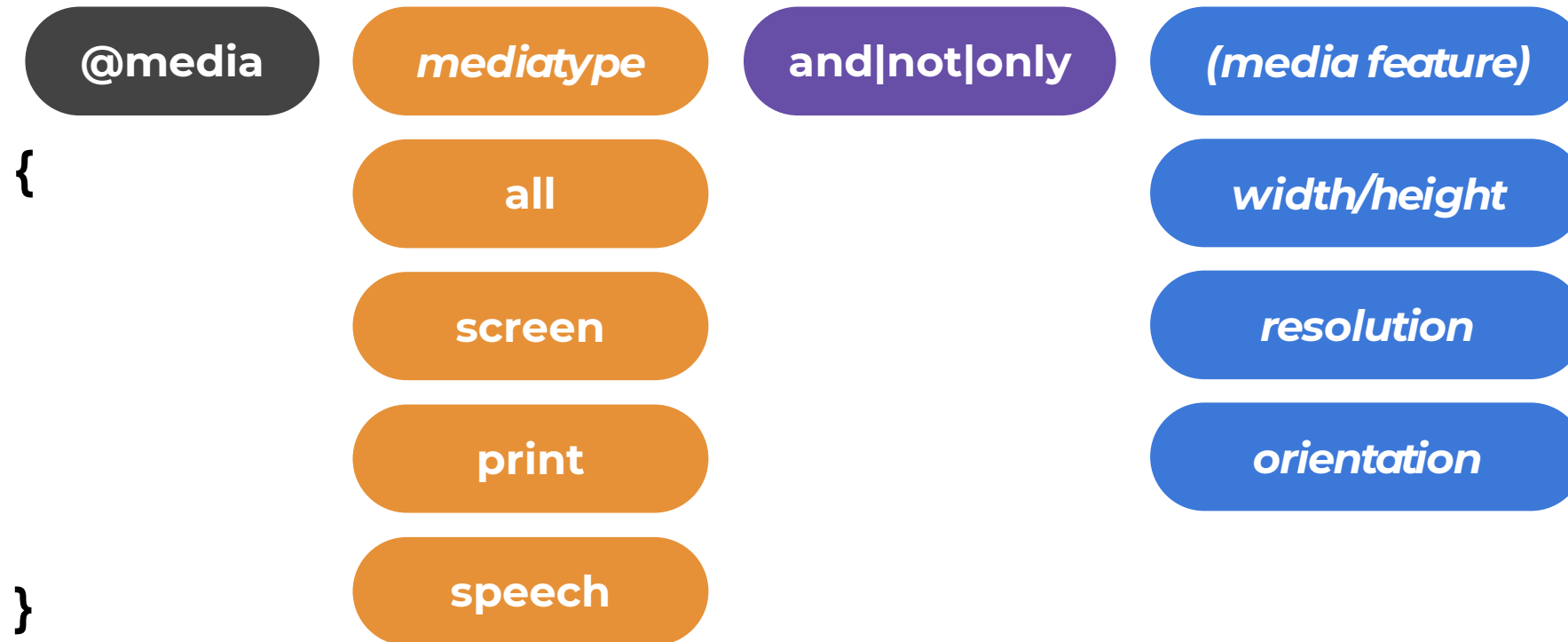
Same Image! ->

```
#notif {  
  background-image: url('sprite.png');  
  width: 30px;  
  height: 30px;  
  background-position: -30px 0px;  
}
```

We only take a portion of the sprite sheet.  
We do this by adjusting the 'window' via **background-position**



# Responsive Design



Reference:

<https://developer.mozilla.org/en-US/docs/Web/CSS/@media>

# Responsive Design

## References:

1. [CSS Tricks: Media Queries for Standard Devices](#)
2. [CSS3 Media Queries](#)

# Animations

- CSS Animations are basically transition from one style to another.
- You can also specify style keyframes.
- Read more on the MDN documentation:
  - [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Animations/Using\\_CSS\\_animations](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations)

# CSS Specificity, Properties, and Other things