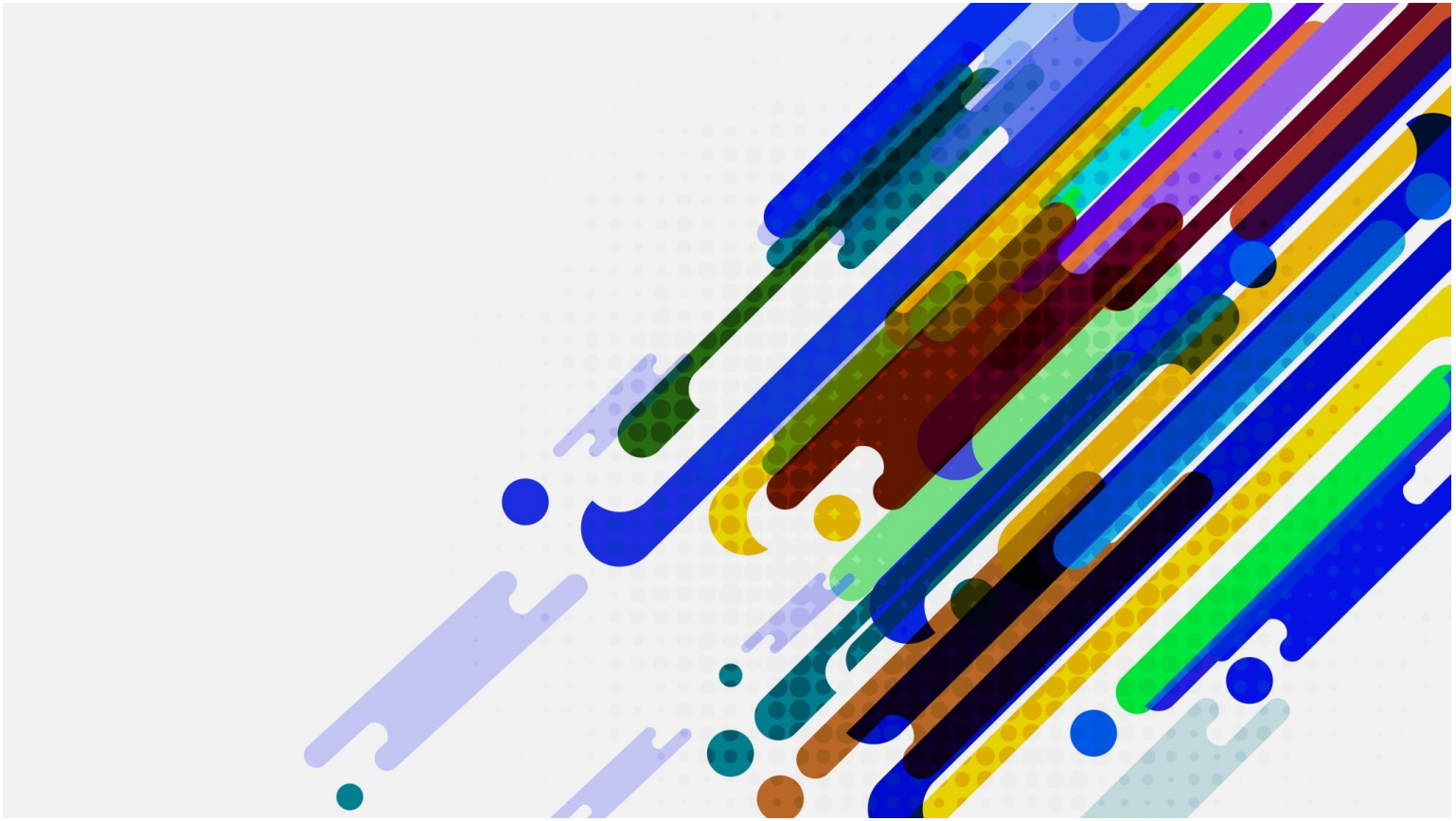


Web Design and CSS

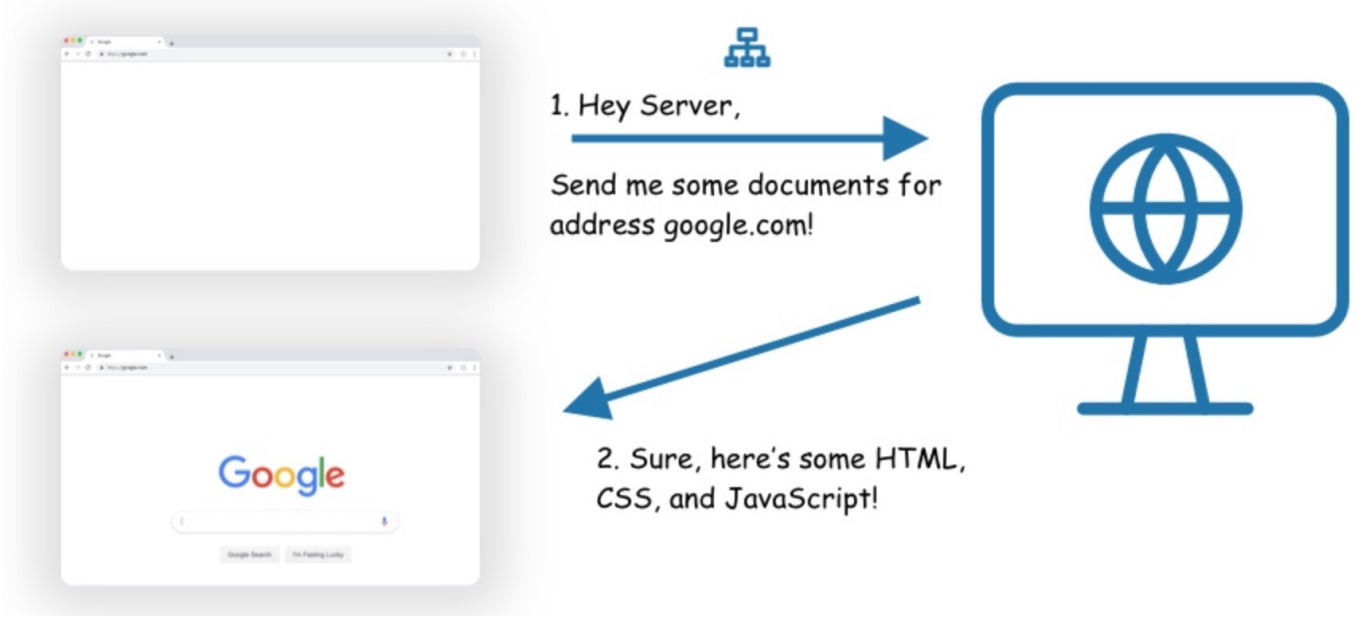
Instructor: PhD Bui Binh Giang

TABLE OF CONTENTS

Introduction to Web Design

How do websites work?

- 1. Browser (Chrome, Firefox, Safari,...) send the request to the server
- 2. Server returns static files (HTML, CSS, JS, image)
- 3. Browser uses static file render UI on the screen



HTML – CSS - JS

HTML (Hypertext Markup Language) **Create the structure**	• Controls the layout content • Provides structure for the web page design • The fundamental building block of any web page
CSS (Cascading Style Sheet) **Stylize the website**	• Applies style to the web page elements • Targets various screen sizes to make web pages responsive • Primarily handles the “look and feel” of a web page
JS (JavaScript) **Increase interactivity**	• Add interactivity to a web page • Handles complex functions and features • Programmatic code which enhances functionality

Choose the right font

- Chose a font which reflects the look and feel you want for your website
- Decide: sans-serif or serif typeface
- Use a good font
- Use right font size, line spacing

F (sans-serif)	F (serif)
• More neutral • Clean • Simple • Modern website	• Has small extensions or stroke • Traditional purposes • Storytelling • Long reading

Choose colors wisely

Red: power, passion, strength and excitement

Orange: friendliness, confidence, courage

Yellow: happiness, liveliness, curiosity, brightness

Green: harmony, nature, life and health

Blue: patience, peace, trustworthiness, stability

Purple: wisdom, nobility, luxury, mystery

Pink: romance, care, peace, affection

Brown: earthiness, nature, durability, comfort

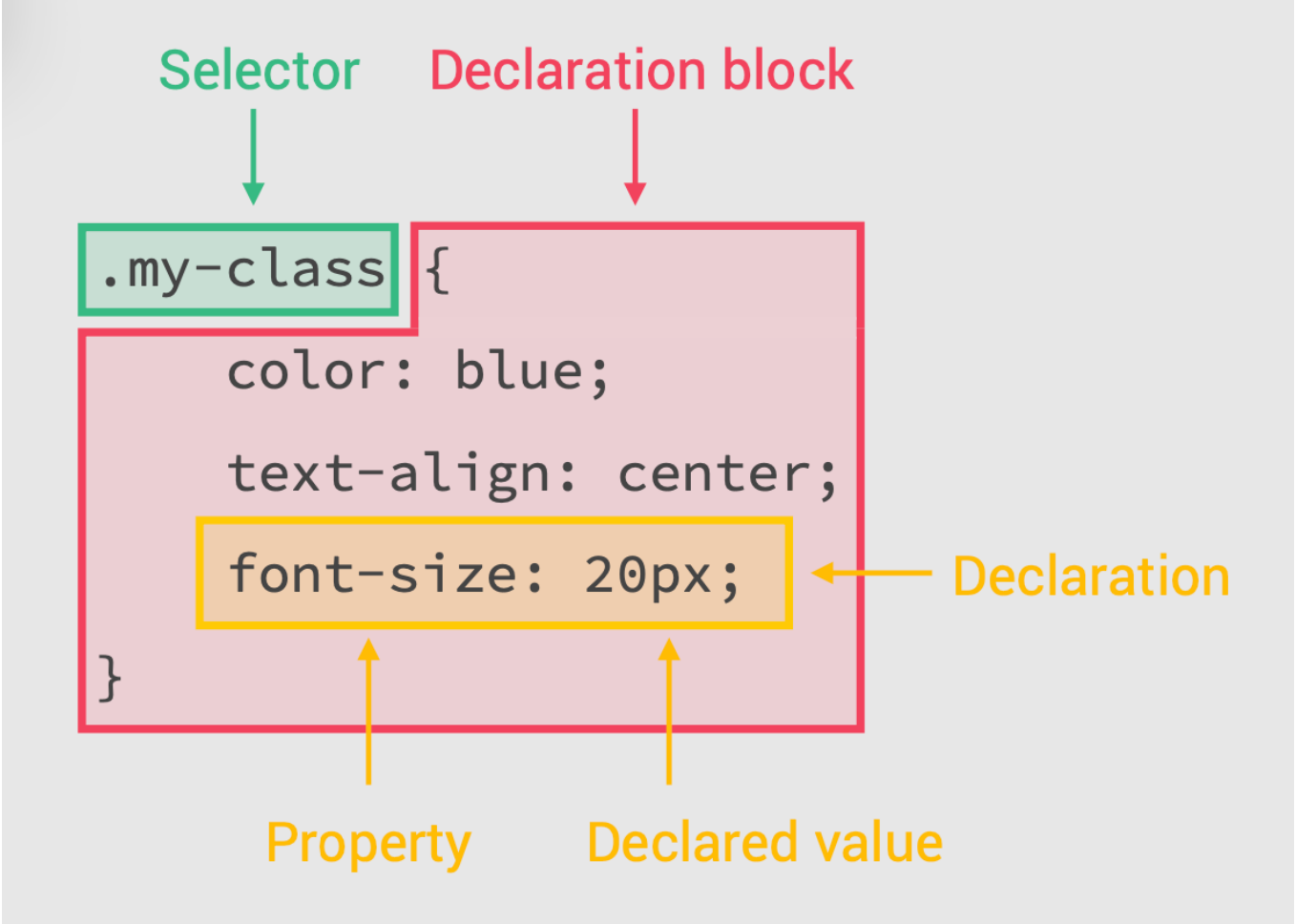
Introduction to CSS

Cascading Style Sheet

- CSS is a style sheet language that styles web content
- CSS handles the look and feel of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how column are sized and laid out, what background images or colors are used, as well as a variety of other effects

A CSS Rule

- Selector: use to select one or more HTML elements that we want to style
- Declaration block: where we write the actual styles
- Declaration: property + declared value
- Property: CSS property
- Declared value: value that we assign to a property



CSS Selectors

Selector name	What does it select	Example

Element selector (tag or type selector)	All HTML elements of the specified type	`<p>Hello World</p>`
		
		`p {`
		`  text-align: center;`
		`  color: red;`
		`}`

ID selector	The element on the page with the specified ID. On a given HTML page, each id value should be unique	`<p id="para1">Hello World</p>`
		
		`#para1 {`
		`  text-align: center;`
		`  color: red;`
		`}`

Class selector	The element(s) on the page with the specified class. Multiple instances of the same class can appear on a page	`<p class="center">Hello World</p>`
		
		`.center {`
		`  text-align: center;`
		`  color: red;`
		`}`

CSS Selectors

Selector name	What does it select	Example

Attribute selector	The element(s) on the page with the specified attribute	`<a href="http://www.wikipedia.org" target="_top">wikipedia.org</a>`
		
		`a[target] {`
		`  background-color: yellow;`
		`}`

Pseudo-class selector	The specified element(s), but only when in the specified state (For example, when a cursor hovers over a link)	`<a href="https://www.wikipedia.org">wikipedia.org</a>`
		
		`a:hover {`
		`  background-color: yellow;`
		`}`

Applying CSS to HTML

- Method 1: Inline CSS
- Method 2: Internal CSS
- Method 3: External CSS

Inline CSS

- Inline styles are CSS declarations that affect a single HTML element, contained within a `style` attribute.
- Example: set heading text color is blue, background color is yellow and border black

```
<h1 style="color: blue; background-color: yellow; border: 1px solid black;"> Hello World! </h1>
```

- Result:

Internal CSS

- An internal stylesheet resides within an HTML document. To create an internal stylesheet, we place CSS inside a <style> element contained inside the HTML <head>
- Example: set heading text color is blue, background color is yellow and border black

```
<!DOCTYPE html>
<html lang="en-GB">
  <head>
    <meta charset="utf-8" />
    <title>My CSS</title>
    <style>
      h1 {
        color: blue;
        background-color: yellow;
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

- Result

Hello World!

External CSS

- With an external style sheet, you can change the look of an entire website by changing just one file
- Each HTML page must include a reference to the external style sheet file (*.css) inside the <link> element, inside the head section
- Example: set heading text color is navy and margin left 20px, body background color is light blue

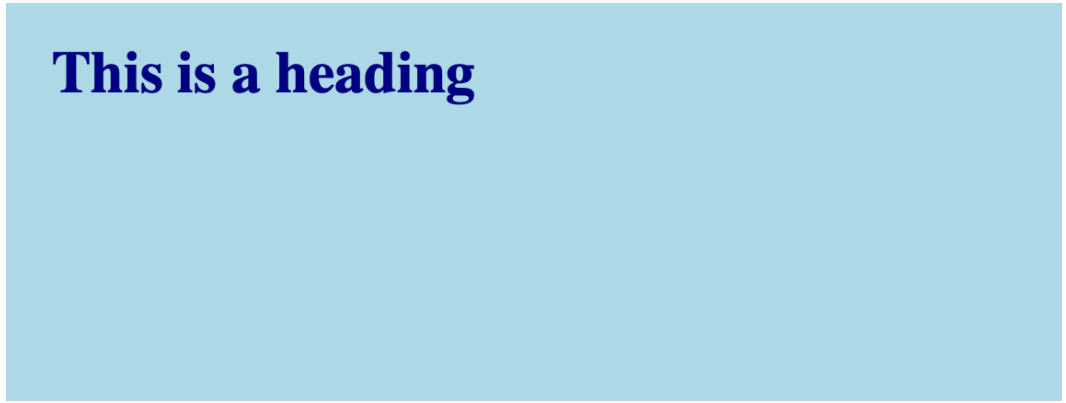
index.html

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css"
href="mystyle.css">
</head>
<body>

<h1>This is a heading</h1>

</body>
</html>
```

mystyle.css

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

h1 {
    color: navy;
    margin-left: 20px;
}
```

- Result

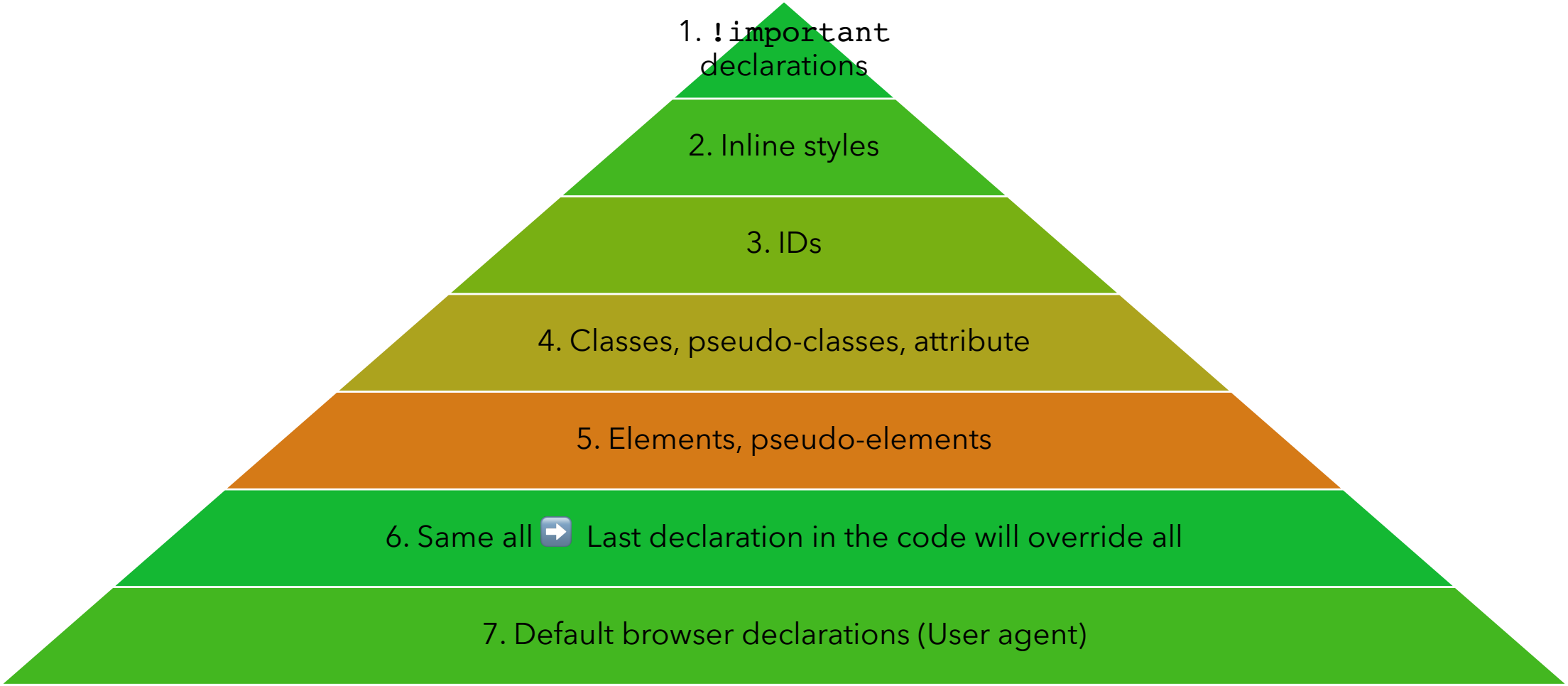


The cascade

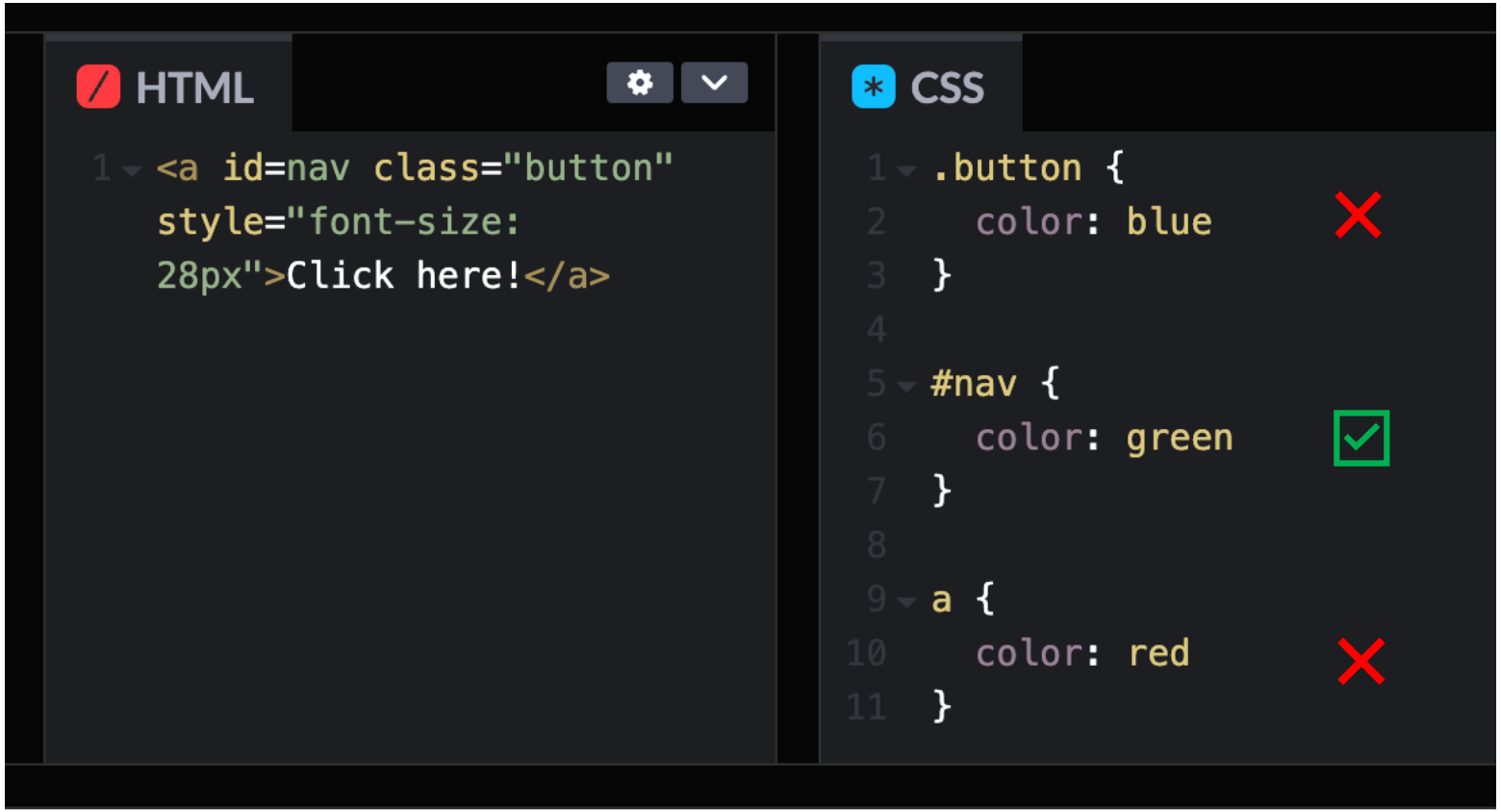
- Process of combining different stylesheets and resolving conflicts between different CSS rules and declarations, when more than one rule applies to a certain element

This is a heading

Cascading order



IDs > Class > Element



Click here!

`!important` > Inline style > IDs

This is a heading

CSS Color

- Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.
- `background-color`: set the background color
- `color`: set the color of text
- `border`: set the borders

CSS Color Example

HTML

```
1 <h1 style="background-color:Orange;">Hello World</h1>
2 <h1 style="color:Tomato;">Hello World</h1>
3 <h1 style="border:2px solid Violet;">Hello World</h1>
4
```

CSS Color Value

HTML

```html
<h1 style="background-color:rgb(255, 99, 71);">rgb(255, 99, 71)</h1>
<h1 style="background-color:#ff6347;">#ff6347</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">hsl(9, 100%, 64%)</h1>
<h1 style="background-color:rgba(255, 99, 71, 0.5);">rgba(255, 99, 71, 0.5)</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">hsla(9, 100%, 64%, 0.5)</h1>
```

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

rgba(255, 99, 71, 0.5)

hsla(9, 100%, 64%, 0.5)

CSS Background

- `background-color`: set the background color
- `opacity`: set the transparency of an element (0 - 1)
- `background-image`: set an image to use as the background of an element

HTML

```html
<h1>Hello World!</h1>
<p>This text is not easy to read on
this background image.</p>
```

CSS

```css
body {
  background-image: url("https://picsum.photos/200/300");
}
```

CSS Margins

- Margins are used to create space around elements, outside of any defined borders.
- `margin, margin-top, margin-left, margin-bottom, margin-right`

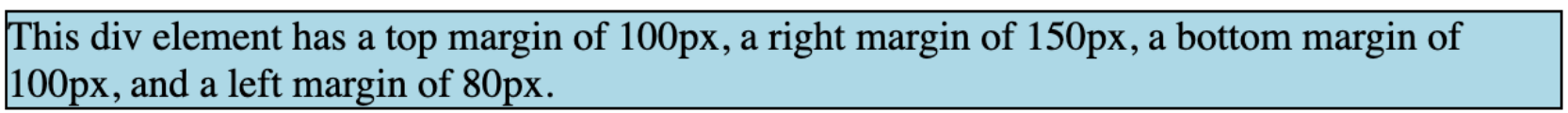
HTML

```
<div>This div element has a top margin
of 100px, a right margin of 150px, a
bottom margin of 100px, and a left
margin of 80px.</div>
```

CSS

```
div {
  border: 1px solid black;
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
  background-color: lightblue;
}
```

JS



CSS Padding

- Padding is used to create space around an element's content, inside of any defined borders.
- `padding, padding-top, padding-left, padding-bottom, padding-right`

HTML

```html
<div>This div element has a top padding of 50px, a right padding of 30px, a bottom padding of 50px, and a left padding of 80px.</div>
```

CSS

```css
div {
  border: 1px solid black;
  background-color: lightblue;
  padding-top: 50px;
  padding-right: 30px;
  padding-bottom: 50px;
  padding-left: 80px;
}
```

JS

This div element has a top padding of 50px, a right padding of 30px, a bottom padding of 50px, and a left padding of 80px.

CSS Height, Width, Max Height, Max Width

- The CSS `height` and `width` properties are used to set the height and width of an element.
- The CSS `max-width/max-height` property is used to set the maximum width/height of an element.

CSS Height, Width Values

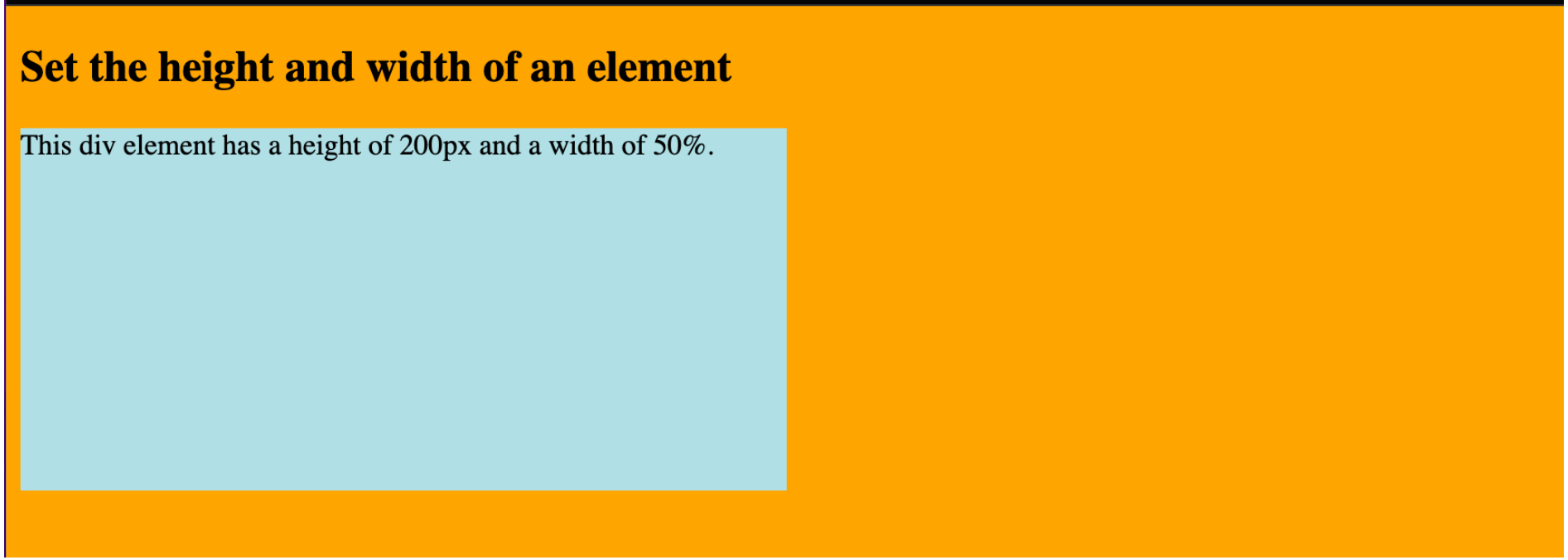
- `auto` - This is default. The browser calculates the height and width
- `length` - Defines the height/width in px, cm, etc.
- `%` - Defines the height/width in percent of the containing block
- `initial` - Sets the height/width to its default value
- `inherit` - The height/width will be inherited from its parent value

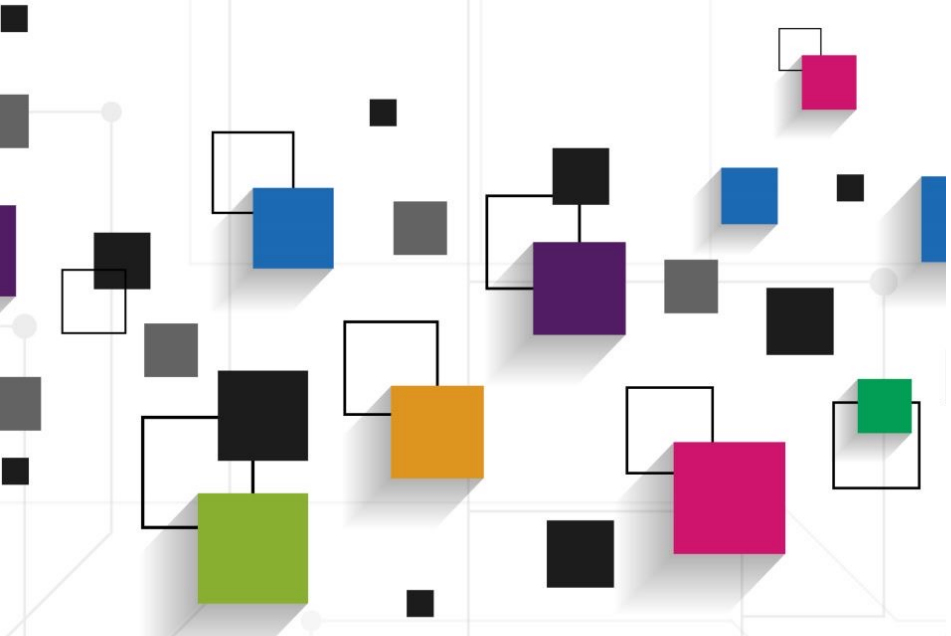
HTML

```html
<h2>Set the height and width of an element</h2>

<div>This div element has a height of 200px and a width of 50%.</div>
```

CSS

```css
body {
    background-color: orange;
}
div {
  height: 200px;
  width: 50%;
  background-color: powderblue;
}
```

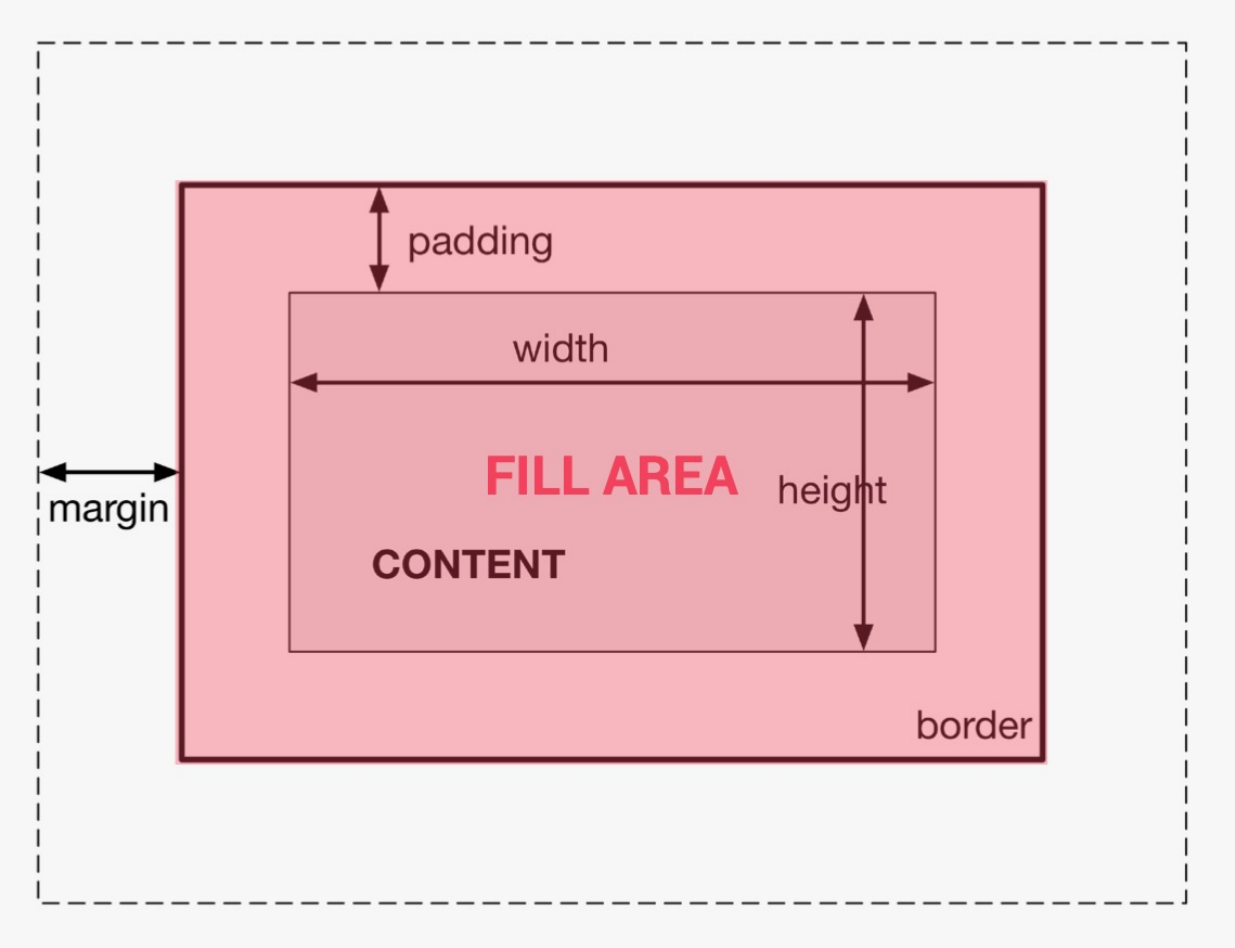




CSS Box Model

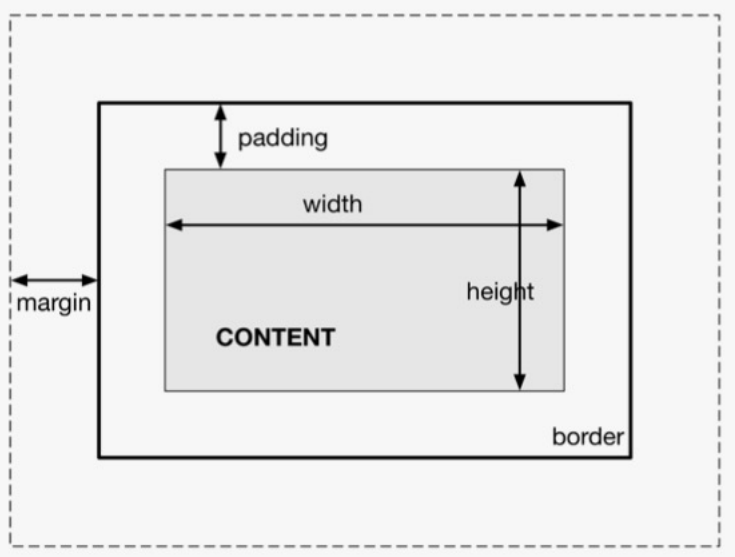
- All HTML elements can be considered as boxes.
- Key to being able to create more complex layouts with CSS, or to align items with other items
- It consists of margins, borders, padding, and the actual content

The Box Model



- **Content**: text, images, etc.
- **Padding**: transparent area around the content, inside of the box
- **Border**: goes around the padding and the content
- **Margin**: space between boxes
- **Fill area**: area that gets filled with background color or background image

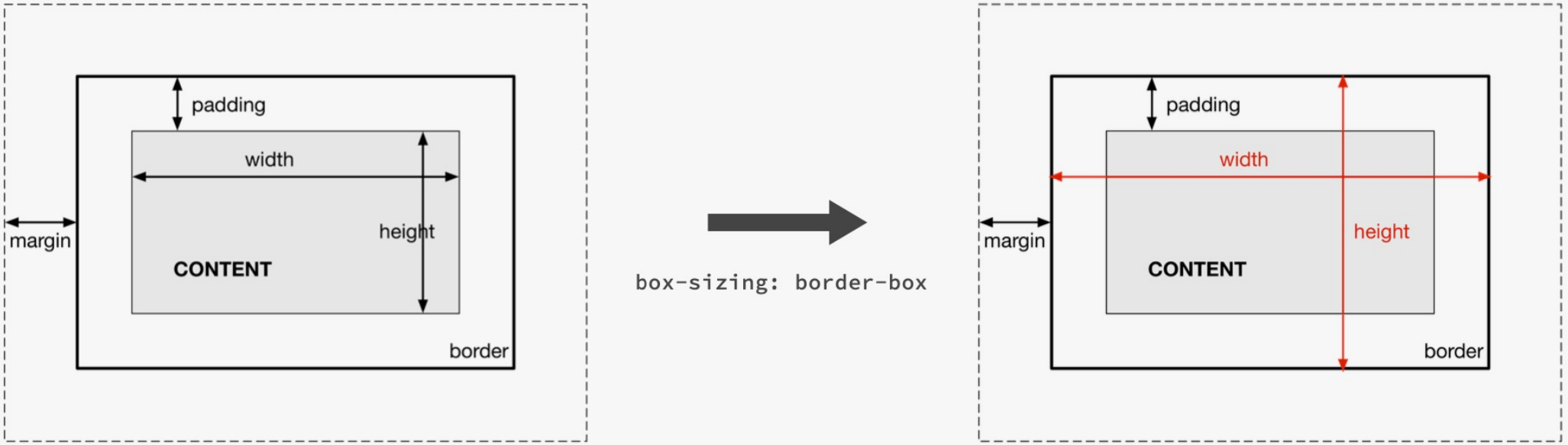
The Box Model Height and Width



- total width = right border + right padding + specified width + left padding + left border
- total height = top border + top padding + specified height + bottom padding + bottom border
- Example: height = 0 + 20px + 100px + 20px + 0 = 140px

The Box Model with `box-sizing: border-box`

- The `box-sizing` property allows us to include the padding and border in an element's total width and height.



The Box Model with box-sizing: border-box

- total width = ~~right border~~ + ~~right padding~~ + specified width + ~~left padding~~ + ~~left border~~
- total height = ~~top border~~ + ~~top padding~~ + specified height + ~~bottom padding~~ + ~~bottom border~~
- Example: height = 0 + ~~20~~px + 100px + ~~20~~px + 0 = 100px

Box types

- `display`: set box types
- Determine how boxes are laid out on a page
- It consists of block boxes, inline-block boxes and inline boxes

Block-level boxes

- Elements formatted visually as blocks
- 100% of parent's width
- Vertically, one after another
- Box-model applies as showed

```
display: block

(display: flex)
(display: list-item)
(display: table)
```

Inline-block boxes

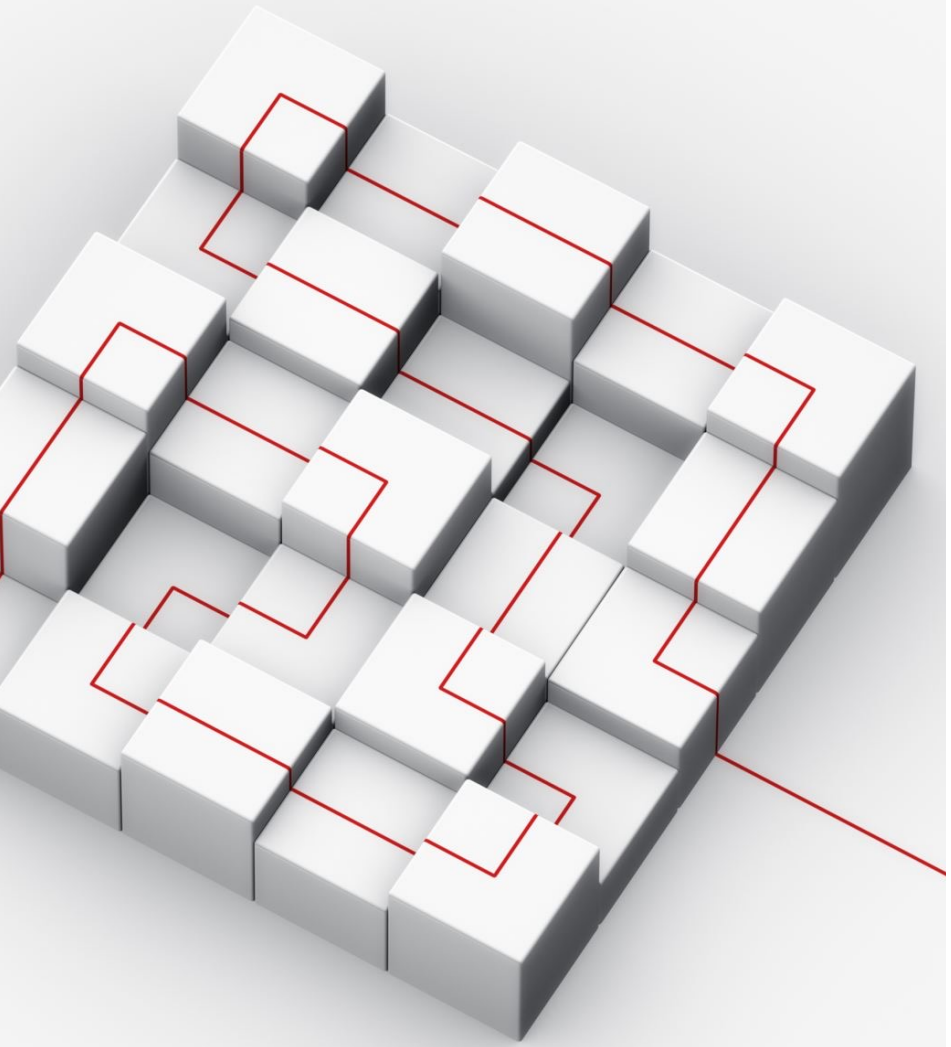
- A mix of block and inline
- Occupies only content's space
- No line-breaks
- Box-model applies as showed

```
display: inline-block
```

Inline boxes

- Content is distributed in lines
- Occupies only content's space
- No line-breaks
- No heights and widths (no box-model)
- Paddings and margins only horizontal (left and right)

```
display: inline
```



CSS Layout

- CSS page layout techniques allow us to take elements contained in a web page and control where they're positioned relative to the following factors: their default position in normal layout flow, the other elements around them, their parent container, and the main viewport/window
- It consists of normal flow, floats, positioning, grid, flexbox, table layout

Normal flow

- Default positioning scheme
- **NOT** floated
- **NOT** absolutely positioned
- Elements laid out according to their source order (in HTML)

```
Default
position: relative
```

Floats

- Element is removed from the normal flow
- Text and inline elements will wrap around the floated element
- The container will not adjust its height to the element

```
float: left
float: right
```

Absolute positioning

- Element is removed from the normal flow
- No impact on surrounding content or elements
- We use `top`, `bottom`, `left` and `right` to offset the element from its relatively positioned container

```
position: absolute
position: fixed
```

Normal flow

HTML

```
<p>I love my cat.</p>

<ul>
  <li>Buy cat food</li>
  <li>Exercise</li>
  <li>Cheer up friend</li>
</ul>

<p>The end!</p>
```

I love my cat.

- Buy cat food
- Exercise
- Cheer up friend

The end!

Floats

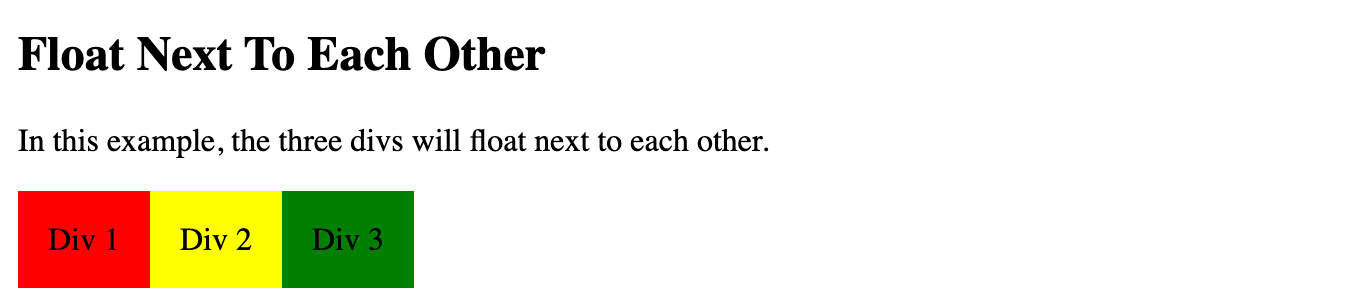
- `left` – Floats the element to the left.
- `right` – Floats the element to the right.
- `none` – Specifies no floating at all. This is the default value.
- `inherit` – Specifies that the value of the float property should be inherited from the element's parent element.

HTML

```
<h2>Float Next To Each Other</h2>

<p>In this example, the three divs
will float next to each other.</p>

<div class="div1">Div 1</div>
<div class="div2">Div 2</div>
<div class="div3">Div 3</div>
```

CSS

```
div {
  float: left;
  padding: 15px;
}
.div1 {
  background: red;
}
.div2 {
  background: yellow;
}
.div3 {
  background: green;
}
```



Relative Positioning

- `position: relative` allows us to offset an item from its default position in normal flow. We can set the top, right, bottom, and left properties

HTML

```
<h2>position: relative;</h2>

<p>An element with position: relative; is
positioned relative to its normal position:</p>

<div class="relative">
This div element has position: relative;
</div>
```

CSS

```
.relative {
  position: relative;
  left: 30px;
  border: 3px solid #73AD21;
}
```

position: relative;

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

This div element has position: relative;

Absolute Positioning

- `position: absolute` is used to completely remove an element from the normal flow and instead position it using offsets from the edges of a containing block

HTML

```html
<h2>position: absolute;</h2>

<p>An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):</p>

<div class="relative">This div element has position: relative;
  <div class="absolute">This div element has position: absolute;</div>
</div>
```

CSS

```css
.relative {
  position: relative;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}

.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #73AD21;
}
```

position: absolute;

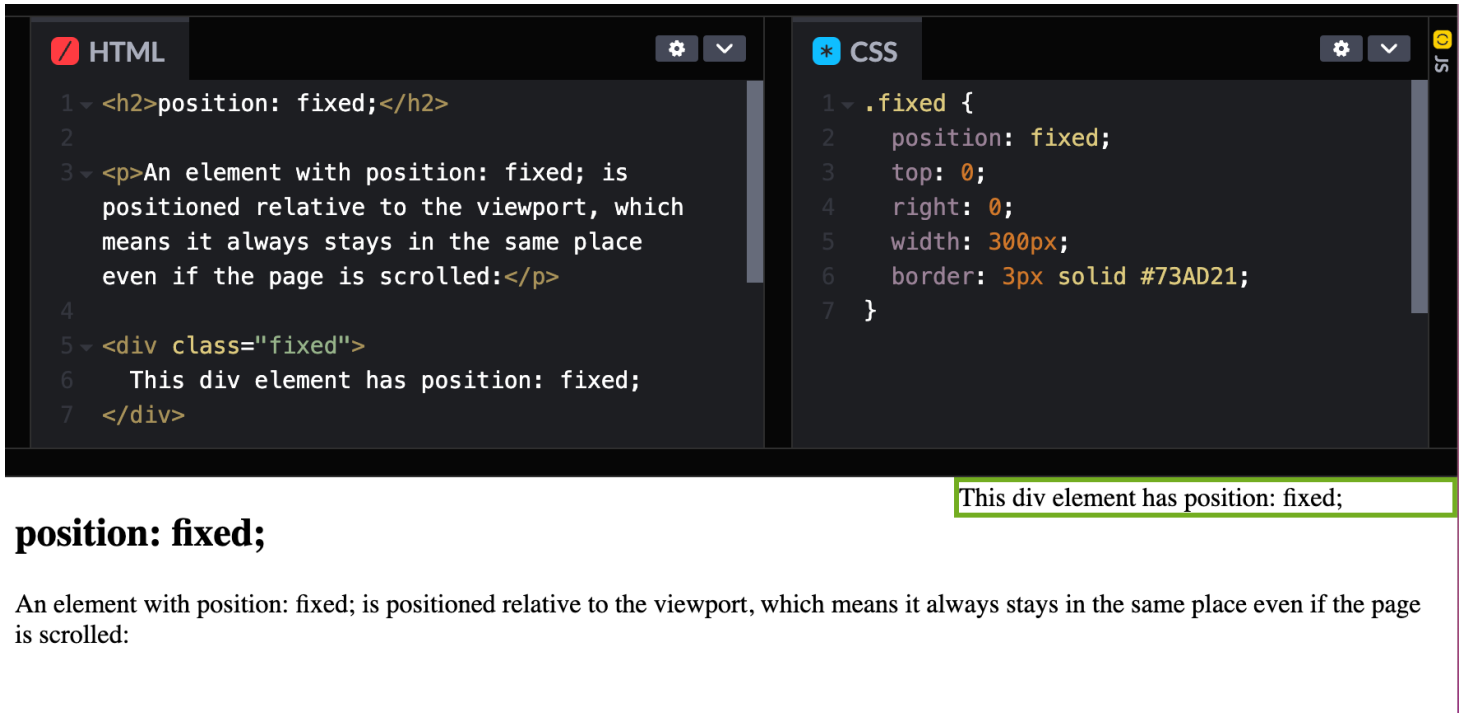
An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):

This div element has position: relative;

This div element has position: absolute;

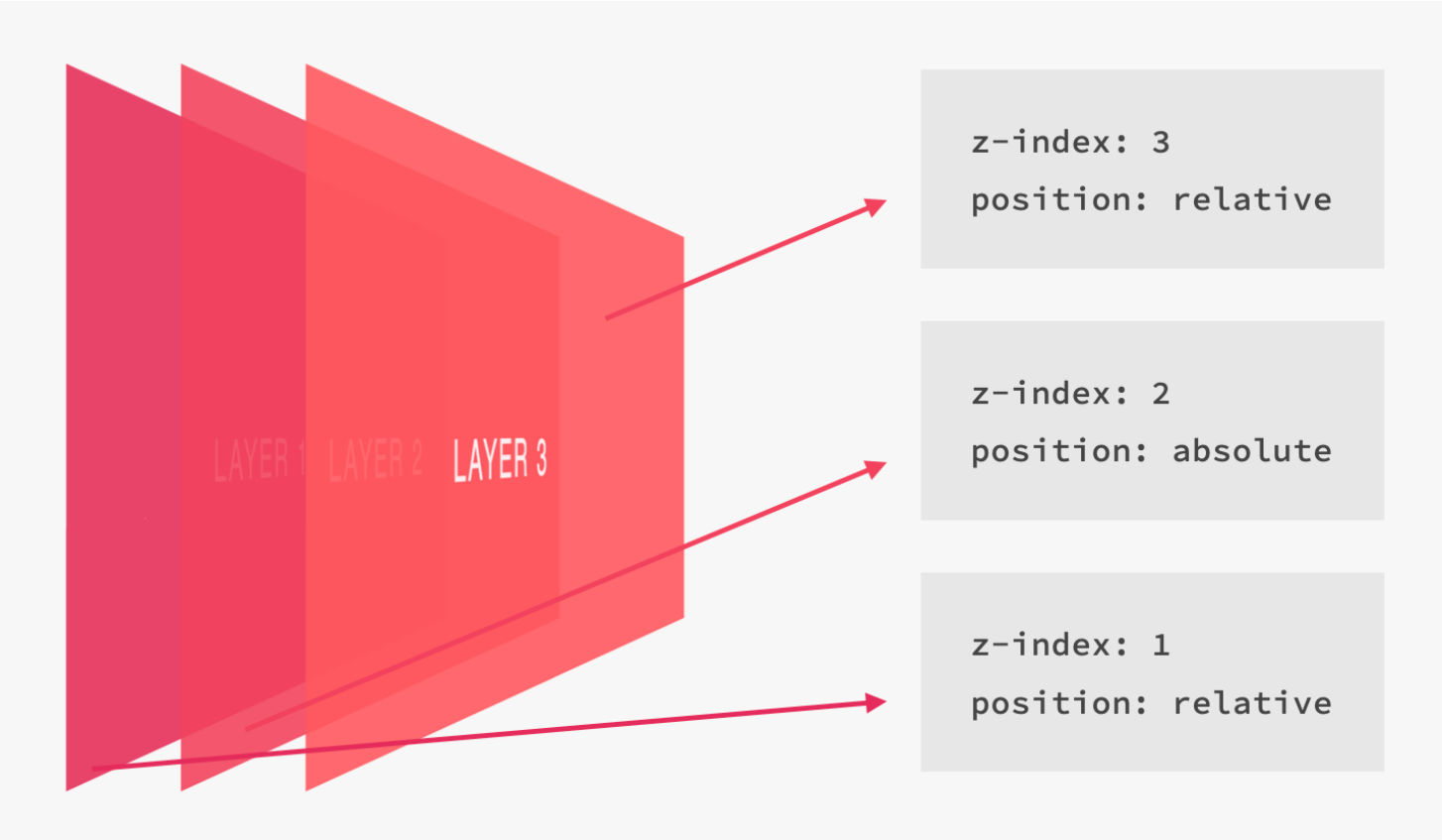
Fixed Positioning

- `position: fixed` removes our element from document flow in the same way as absolute positioning. However, instead of the offsets being applied from the container, they are applied from the viewport
- We can set the top, right, bottom, and left properties



Stacking Contexts

- When elements are positioned, they can overlap other elements.
- The `z-index` property specifies the stack order of an element (which element should be placed in front of, or behind, the others)



HTML

```html
<h1>This is a heading</h1>
<img src="https://picsum.photos/200/300">
<p>Because the image has a z-index of -1,
it will be placed behind the text.</p>
```

CSS

```css
img {
  position: absolute;
  left: 0px;
  top: 0px;
  z-index: -1;
}
```

This is a heading

Because the image has a z-index of -1, it will be placed behind the text.

CSS Units

- CSS has several different units for expressing a length
- Many CSS properties take "length" values, such as `width`, `margin`, `padding`, `font-size`, etc.
- There are two types of length units: absolute and relative.

Absolute Lengths

- The absolute length units are fixed, and a length expressed in any of these will appear as exactly that size
- Absolute length units are not recommended for use on screen, because screen sizes vary so much

Unit	Description
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)

```
h1 {font-size: 1.5cm;}
h2 {font-size: 30px;}
p {
  font-size: 15pt;
  line-height: 20px;
}
```

Relative Lengths

- Relative length units specify a length relative to another length property. Relative length units scale better between different rendering mediums
- **Tip:** The em and rem units are practical in creating perfectly scalable layout

Unit	Description
em	Relative to the font-size of the element (2em means 2 times the size of the current font)
rem	Relative to font-size of the root element
vw	Relative to 1% of the width of the viewport
vh	Relative to 1% of the height of the viewport
%	Relative to the parent element

```
h1 {font-size: 1.5rem;}
h2 {font-size: 2em;}
p {
  font-size: 150%;
  margin-top: 1.25vh;
}
```

How Units are converted from relative to absolute (px)

		Example (x)	How to convert to pixels	Result in pixels
	% (fonts)	150%	x% * parent's computed font-size	24px
	% (lengths)	10%	x% * parent's computed **width**	100px
Font-based	em (font)	3em	x * **parent** computed font-size	72px (3 * 24)
Font-based	em (lengths)	2em	x * **current element** computed font-size	48px
Font-based	rem	10rem	x * **root** computed font-size	160px
Viewport-based	vh	90vh	x * 1% of viewport height	90% of the current viewport height
Viewport-based	vw	80vw	x * 1% of viewport width	80% of the current viewport width

4. Computed value
(converting relative values to absolute)

```
html, body {
  font-size: 16px;
  width: 80vw;
}

header {
  font-size: 150%;
  padding: 2em;
  margin-bottom: 10rem;
  height: 90vh;
  width: 1000px;
}

.header-child {
  font-size: 3em;
  padding: 10%;
}
```

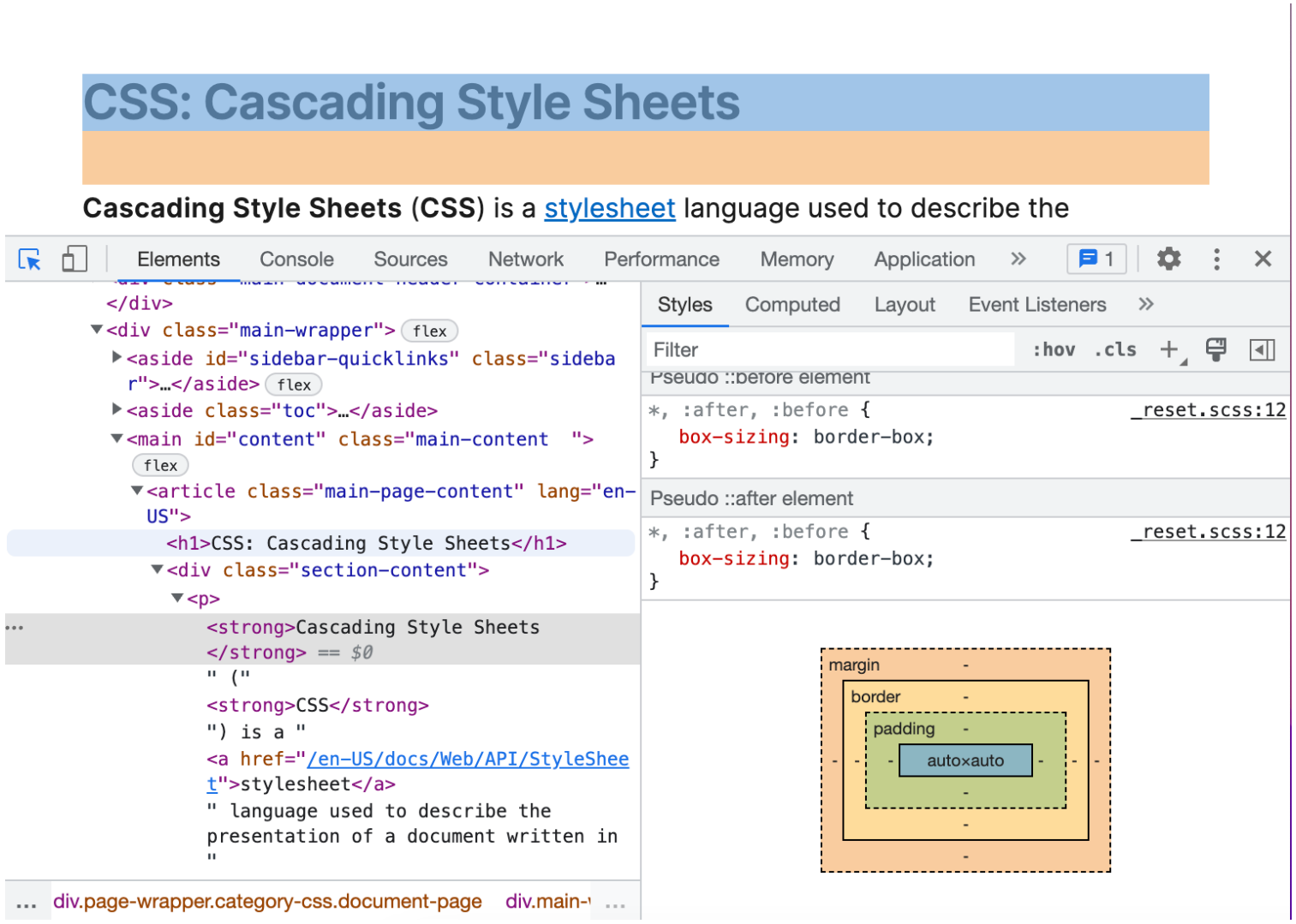
CSS Value Processing

- Browsers specify a root `font-size` for each page (usually 16px)
- Percentages and relative values are always converted to pixels
- Percentages are measured relative to their parent's font-size, if used to specify `font-size`
- Percentages are measured relative to their parent's width, if used to specify lengths
- em are measured relative to their parent font-size, if used to specify `font-size`
- em are measured relative to the current font-size, if used to specify lengths
- rem are always measured relative to the document's root `font-size`
- vh and vw are simply percentage measurements of the viewport's `height` and `width`

Inspecting the applied CSS

- Select an element on a page, either by right/ctrl-clicking on it and selecting Inspect, or selecting it from the HTML tree on the left of the DevTools display

DevTools


CSS: Cascading Style Sheets
Cascading Style Sheets (CSS) is a stylesheet language used to describe the
Elements
Console
Sources
Network
Performance
Memory
Application
1

 flex
<aside id="sidebar-quicklinks" class="sideba r">…</aside> flex
<aside class="toc">…</aside>
<main id="content" class="main-content ">
flex
<article class="main-page-content" lang="en-US">
<h1>CSS: Cascading Style Sheets</h1>

<p>
Cascading Style Sheets == $0
" ("
CSS
") is a "
stylesheet
" language used to describe the presentation of a document written in "
div.page-wrapper.category-css.document-page
div.main-
Styles
Computed
Layout
Event Listeners
Filter
:hov .cls
Pseudo ::before element
*, :after, :before {
box-sizing: border-box;
}
_reset.scss:12
Pseudo ::after element
*, :after, :before {
box-sizing: border-box;
}
_reset.scss:12
margin
border
padding
auto×auto

CSS Fonts

- We use the `font-family` property to specify the font of a text
- The `font-style` property is mostly used to specify italic text
- The `font-weight` property specifies the weight of a font
- The `font-size` property sets the size of the text

HTML

```
<p>This is a paragraph, shown in
the Times New Roman font.</p>
```

CSS

```
p {
  font-family: "Times New Roman", Times, serif;
  font-size: 30px;
  font-weight: bold;
  color: green
}

```

This is a paragraph, shown in the Times New Roman font.

CSS Text

- The `color` property is used to set the color of the text
- The `text-align` property is used to set the horizontal alignment of a text
- The `text-decoration` property is used to add a decoration to text
- The `text-transform` property is used to specify uppercase and lowercase letters in a text
- The `line-height` property is used to specify the space between lines

HTML

```html
<p>Demo Text</p>
```

CSS

```css
p {
  color: blue;
  text-align: center;
  text-decoration: underline;
  text-transform: uppercase;
  line-height: 1.8;
}
```

DEMO TEXT

CSS Links

- Links can be styled with any CSS property (e.g. `color`, `font-family`, `background`, etc.)
- In addition, links can be styled differently depending on what state they are in:
  - `a:link` - a normal, unvisited link
  - `a:visited` - a link the user has visited
  - `a:hover` - a link when the user mouses over it
  - `a:active` - a link moment it is clicked

HTML

```html
<h2>Styling a link depending on state</h2>

<p><b><a href="https://www.google.com/"
target="_blank">This is a link</a></b></p>

<p><b>Note:</b> a:hover MUST come after
a:link and a:visited in the CSS definition
in order to be effective.</p>
<p><b>Note:</b> a:active MUST come after
a:hover in the CSS definition in order to
be effective.</p>
```

CSS

```css
a:link {
  color: red;
}
/* visited link */
a:visited {
  color: green;
}
/* mouse over link */
a:hover {
  color: hotpink;
}
/* selected link */
a:active {
  color: blue;
}
```

Styling a link depending on state

This is a link

Note: a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective.

Note: a:active MUST come after a:hover in the CSS definition in order to be effective.

CSS Pseudo-elements

- A CSS pseudo-element is used to style specified parts of an element
- Example:
  - Style the first letter, or line, of an element
  - Insert content before, or after, the content of an element

HTML

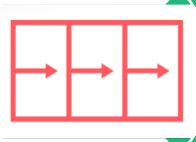
```html
<p>Demo Text</p>
```

CSS

```css
p {
  color: blue
}

p::first-letter {
  color: red;
  font-size: 30px;
}

p::after {
  content: url("https://www.easygifanimator.net/images/samples/eglite.gif");
}
```

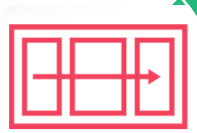
Demo Text

Layout types



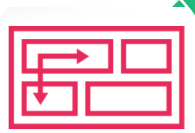
FLOAT LAYOUTS

The **old way of building layouts** of all sizes, using the float CSS property. Still used, but getting outdated



FLEXBOX

Modern way of laying out elements in a **1-dimensional** row without using floats. Perfect for **component layouts**



CSS GRID

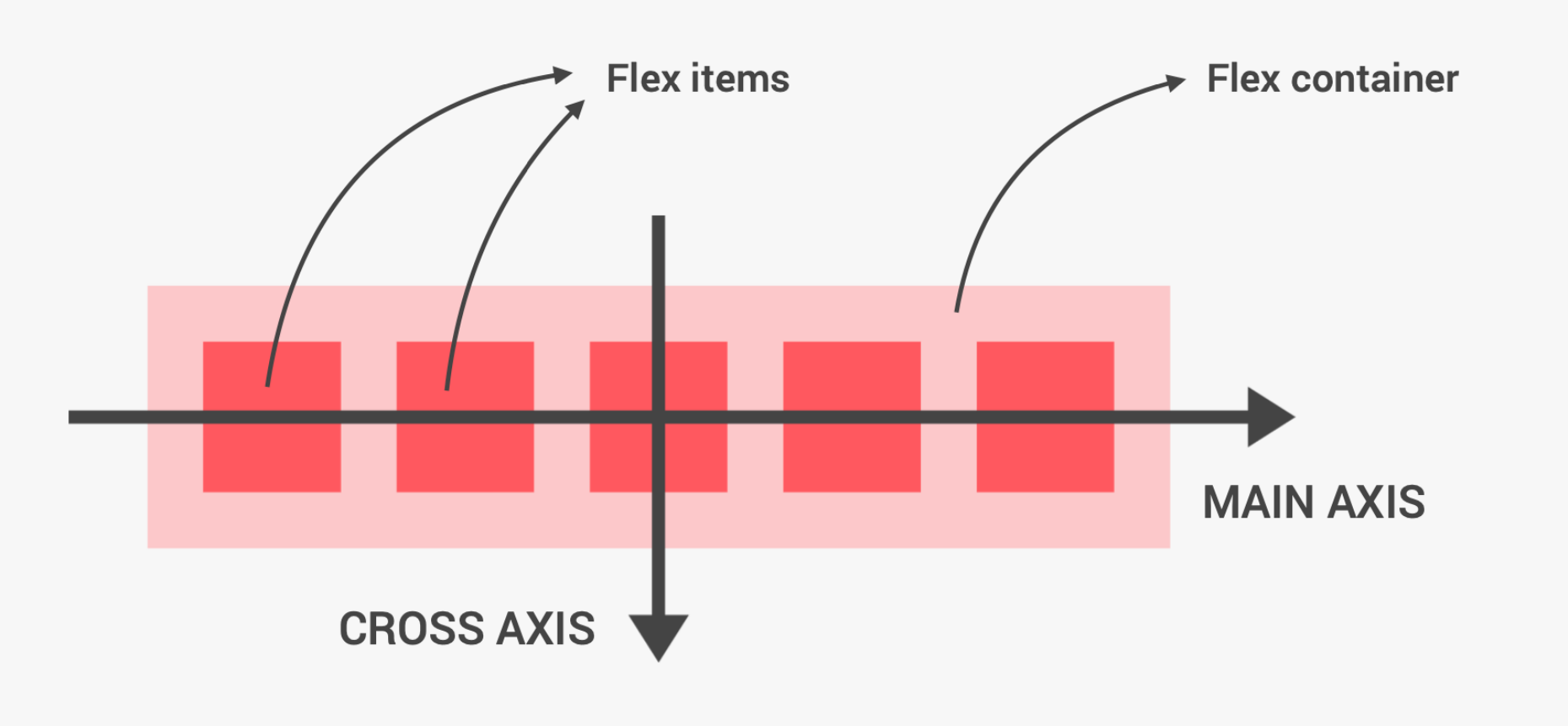
For laying out element in a fully-fledged **2-dimensional grid**. Perfect for **page layouts and complex components**

Flexbox

- Flexbox is a new module in CSS3 that makes it easy to align elements to one another, in different directions and orders
- The main idea behind flexbox is to give the container the ability to expand and to shrink elements to best use all the available space
- Flexbox replaces float layouts, using less, and more readable and logical code
- Flexbox completely changes the way that we build one-dimensional layouts

Flexbox concepts

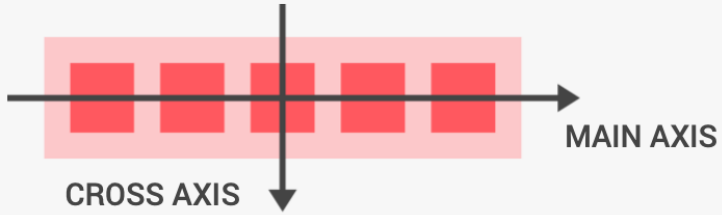
```
display: flex
(display: flex-inline)
```



Flexbox concepts

- The main axis is the axis running in the direction the flex items are laid out in. The start and end of this axis are called the main start and main end
- The cross axis is the axis running perpendicular to the direction the flex items are laid out in. The start and end of this axis are called the cross start and cross end
- The parent element that has `display: flex` set on it is called the flex container
- The items laid out as flexible boxes inside the flex container are called flex items

Flexbox Properties



CONTAINER

1. **flex-direction: row** | row-reverse | column | column-reverse
2. **flex-wrap: nowrap** | wrap | wrap-reverse
3. **justify-content: flex-start** | flex-end | center | space-between | space-around | space-evenly
4. **align-items: stretch** | flex-start | flex-end | center | baseline
5. **align-content: stretch** | flex-start | flex-end | center | space-between | space-around

ITEM

1. **align-self: auto** | stretch | flex-start | flex-end | center | baseline
2. **order: 0** | <integer>
3. **flex-grow: 0** | <integer>
4. **flex-shrink: 1** | <integer>
5. **flex-basis: auto** | <length>

flex: 0 1 auto | <int> <int> <len>

Columns or rows

- `flex-direction` that specifies which direction the main axis runs (which direction the flexbox children are laid out in)
- By default, this is set to `row`
- `flex-direction: column` puts the items back in a column layout

HTML

```html
<div class="flex-row">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
<div class="flex-column">
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
```

CSS

```css
.flex-row {
  display: flex;
  background-color: red;
}

.flex-column {
  display: flex;
  flex-direction: column;
  background-color: DodgerBlue;
}
```

Wrapping

- The `flex-wrap` property specifies whether the flex items should wrap or not
- `flex-wrap: wrap` specifies that the flex items will wrap if necessary
- `flex-wrap: nowrap` specifies that the flex items will not wrap (this is default)

HTML

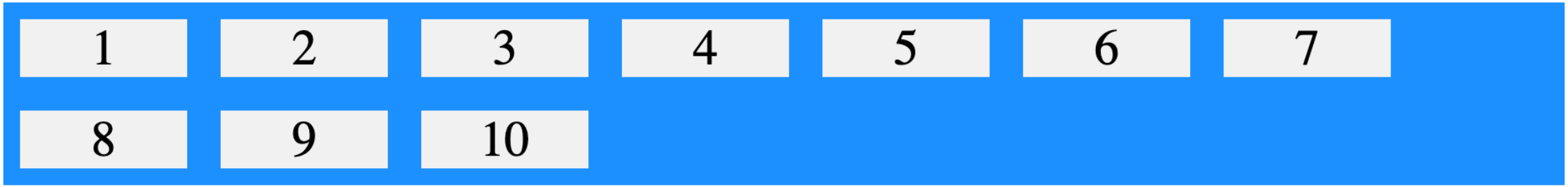
```html
<p>The "flex-wrap: wrap;" specifies that the flex items will wrap if necessary:</p>
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
  <div>10</div>
</div>
```

CSS

```css
.flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
  text-align: center;
  font-size: 30px;
}

.flex-container {
  display: flex;
  flex-wrap: wrap;
  background-color: DodgerBlue;
}
```

JS

The "flex-wrap: wrap;" specifies that the flex items will wrap if necessary:



Horizontal and vertical alignment

- The `justify-content` property is used to align the flex items on the main axis
- The `align-items` property is used to align the flex items on the cross axis

justify-content

- The default value is `flex-start`, which makes all the items sit at the start of the main axis
- You can use `flex-end` to make them sit at the end
- `center` make the flex items sit in the center of the main axis
- `space-around` distributes all the items evenly along the main axis with a bit of space left at either end
- `space-between` is very similar to space-around except that it doesn't leave any space at either end

align-items

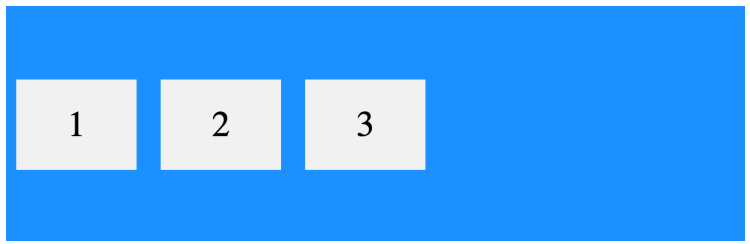
- By default, the value is **`stretch`**, which stretches all flex items to fill the parent in the direction of the cross axis
- **`center`** make the flex items centered along the cross axis
- You can also have values like **`flex-start`** and **`flex-end`**, which will align all items at the start and end of the cross axis respectively

`justify-content: space-between;`

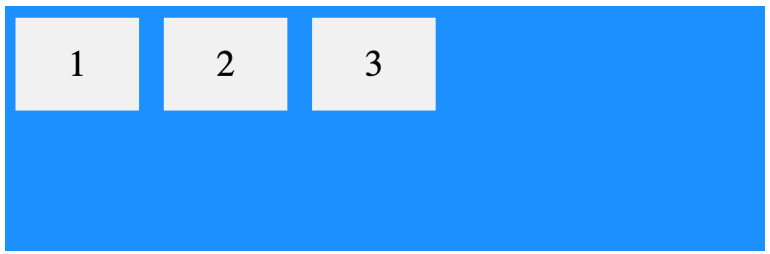
`justify-content: center;`

`justify-content: flex-end;`

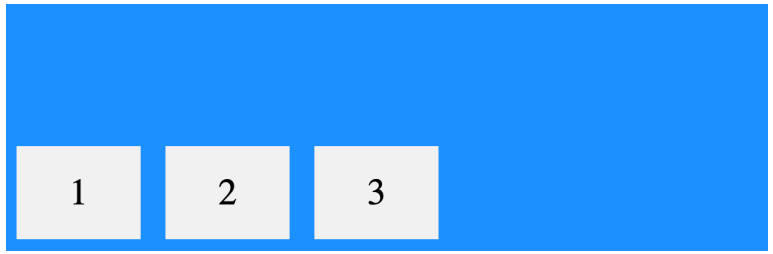
`align-items: center;`



`align-items: flex-start;`



`align-items: flex-end;`



Ordering flex items

- Flexbox also has a feature for changing the layout order of flex items without affecting the source order. This is another thing that is impossible to do with traditional layout methods
- The `order` value must be a number, default value is 0

HTML

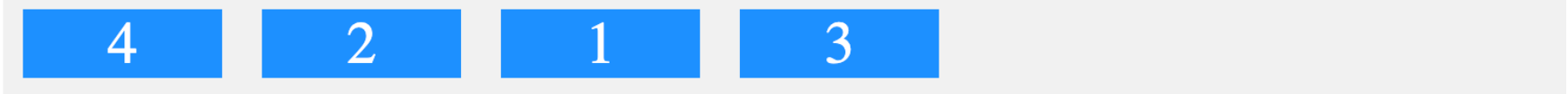
```html
<h1>The order Property</h1>

<p>Use the order property to sort the
flex items as you like:</p>

<div class="flex-container">
  <div style="order: 3">1</div>
  <div style="order: 2">2</div>
  <div style="order: 4">3</div>
  <div style="order: 1">4</div>
</div>
```

CSS

```css
.flex-container>div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  font-size: 30px;
}
.flex-container {
  display: flex;
  align-items: stretch;
  background-color: #f1f1f1;
}
```

The order Property

Use the order property to sort the flex items as you like:



flex-grow

- The `flex-grow` property specifies how much a flex item will grow relative to the rest of the flex items
- The value must be a number, default value is 0

HTML

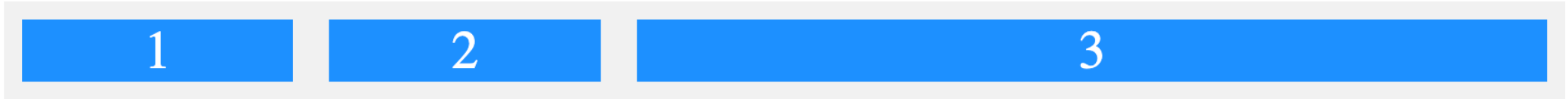
```html
<h1>The flex-grow Property</h1>

<p>Make the third flex item grow eight times faster than the other flex items:</p>

<div class="flex-container">
  <div style="flex-grow: 1">1</div>
  <div style="flex-grow: 1">2</div>
  <div style="flex-grow: 8">3</div>
</div>
```

CSS

```css
.flex-container>div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  font-size: 30px;
}
.flex-container {
  display: flex;
  align-items: stretch;
  background-color: #f1f1f1;
}
```

The flex-grow Property

Make the third flex item grow eight times faster than the other flex items:



flex-shrink

- The `flex-shrink` property specifies how much a flex item will shrink relative to the rest of the flex items
- The value must be a number, default value is 1

HTML

```html
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-shrink: 0">3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
  <div>10</div>
  <div>11</div>
  <div>12</div>
</div>
```

CSS

```css
.flex-container>div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  font-size: 30px;
}
.flex-container {
  display: flex;
  align-items: stretch;
  background-color: #f1f1f1;
}
```

Do not let the third flex item shrink as much as the other flex items:

flex-basis

- The `flex-basis` property specifies the initial length of a flex item

HTML

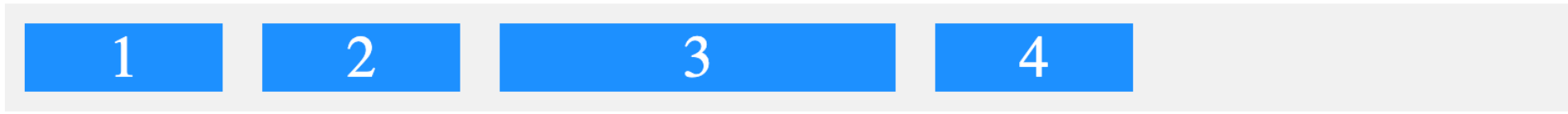
```html
<h1>The flex-basis Property</h1>

<p>Set the initial length of the third flex item to 200 pixels:</p>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-basis:200px">3</div>
  <div>4</div>
</div>
```

CSS

```css
.flex-container>div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  font-size: 30px;
}
.flex-container {
  display: flex;
  align-items: stretch;
  background-color: #f1f1f1;
}
```

The flex-basis Property

Set the initial length of the third flex item to 200 pixels:



align-self

- The `align-self` property specifies the alignment for the selected item inside the flexible container
- The `align-self` property overrides the default alignment set by the container's `align-items` property

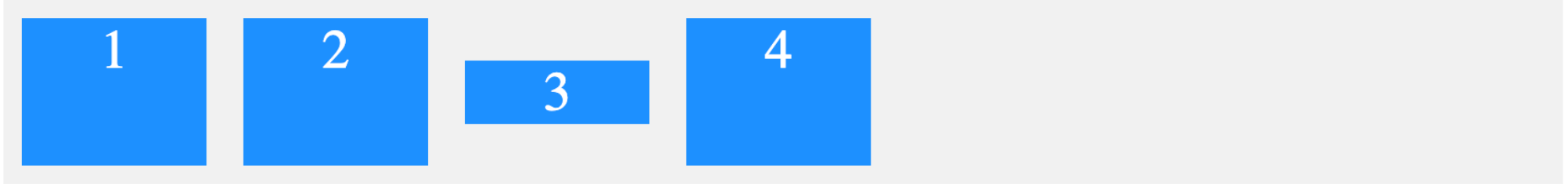
HTML

```html
<p>The "align-self: center;" aligns the selected flex item in the middle of the container:</p>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="align-self: center">3</div>
  <div>4</div>
</div>

<p>The align-self property overrides the align-items property of the container.</p>
```

CSS

```css
.flex-container {
  display: flex;
  height: 100px;
  background-color: #f1f1f1;
}
.flex-container > div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  font-size: 30px;
}
```

The "align-self: center;" aligns the selected flex item in the middle of the container:



The align-self property overrides the align-items property of the container.

Responsive Design

Responsive Web Design

- Responsive web design makes your web page look good on all devices
- Web pages can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device
- Using CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen

Media Queries

- It uses the `@media` rule to include a block of CSS properties only if a certain condition is true
- You can add multiple media queries within a stylesheet, tweaking your whole layout or parts of it to best suit the various screen sizes. The points at which a media query is introduced, and the layout changed, are known as breakpoints

Media Queries Syntax

```
@media media-type and (media-feature-rule) {
  /* CSS rules go here */
}
```

- A media type, which tells the browser what kind of media this code is for (e.g. print, or screen)
- A media expression, which is a rule, or test that must be passed for the contained CSS to be applied
- A set of CSS rules that will be applied if the test passes and the media type is correct

HTML

```html
<p>Resize the browser window. When the width of this document is 1000 pixels or less, the background-color is "lightblue", otherwise it is "lightgreen".</p>
```

CSS

```css
body {
  background-color: lightgreen;
}

@media only screen and (max-width: 1000px) {
  body {
    background-color: lightblue;
  }
}
```

Resize the browser window. When the width of this document is 1000 pixels or less, the background-color is "lightblue", otherwise it is "lightgreen".

Typical Device Breakpoints

```
/* Extra small devices (phones, 576px and down) */
@media only screen and (max-width: 576px) {
}
/* Small devices (portrait tablets and large phones, 576px and up) */
@media only screen and (min-width: 576px) {
}
/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {
}
/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {
}
/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {
}
```

Responsive images

- If the `width` property is set to a percentage and the height property is set to "`auto`", the image will be responsive and scale up and down

```
img {
  width: 100%;
  height: auto;
}
```

- If the `max-width` property is set to `100%`, the image will scale down if it must, but never scale up to be larger than its original size

```
img {
  max-width: 100%;
  height: auto;
}
```

Responsive typography

- Responsive typography describes changing font sizes within media queries or using viewport units to reflect lesser or greater amounts of screen real estate

```
@media (min-width: 1200px) { h1 { font-size: 4rem; } }
h1 { font-size: 1em; }
h2 { font-size: 2rem; }
h3 { font-size: 6vw; }
```

Other Responsive Layout Technologies

- Multiple-column layout
- Flexbox
- CSS Grid
- Third-party grid systems such as Bootstrap

THE END