CSS3

IMY 220 ● Lecture 2

Introduction

CSS3 is the latest evolution of the Cascading Style Sheets language and aims at extending CSS2.1.

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents.

Vendor prefixes

Browser vendors sometimes add prefixes to experimental or nonstandard CSS properties.

Vendor prefixes

Typically the vendors use these prefixes:

- -webkit- (Chrome, newer versions of Opera)
- -moz- (Firefox)
- -o- (Old versions of Opera)
- -ms- (Internet Explorer)

Vendor prefixes

```css
div{
        box-shadow: 0 0 0 10px #000;
        -moz-box-shadow: 0 0 0 10px #000;
        -o-box-shadow: 0 0 0 10px #000;
        -ms-box-shadow: 0 0 0 10px #000;
}
```

Use https://caniuse.com/ to check browser support for new features

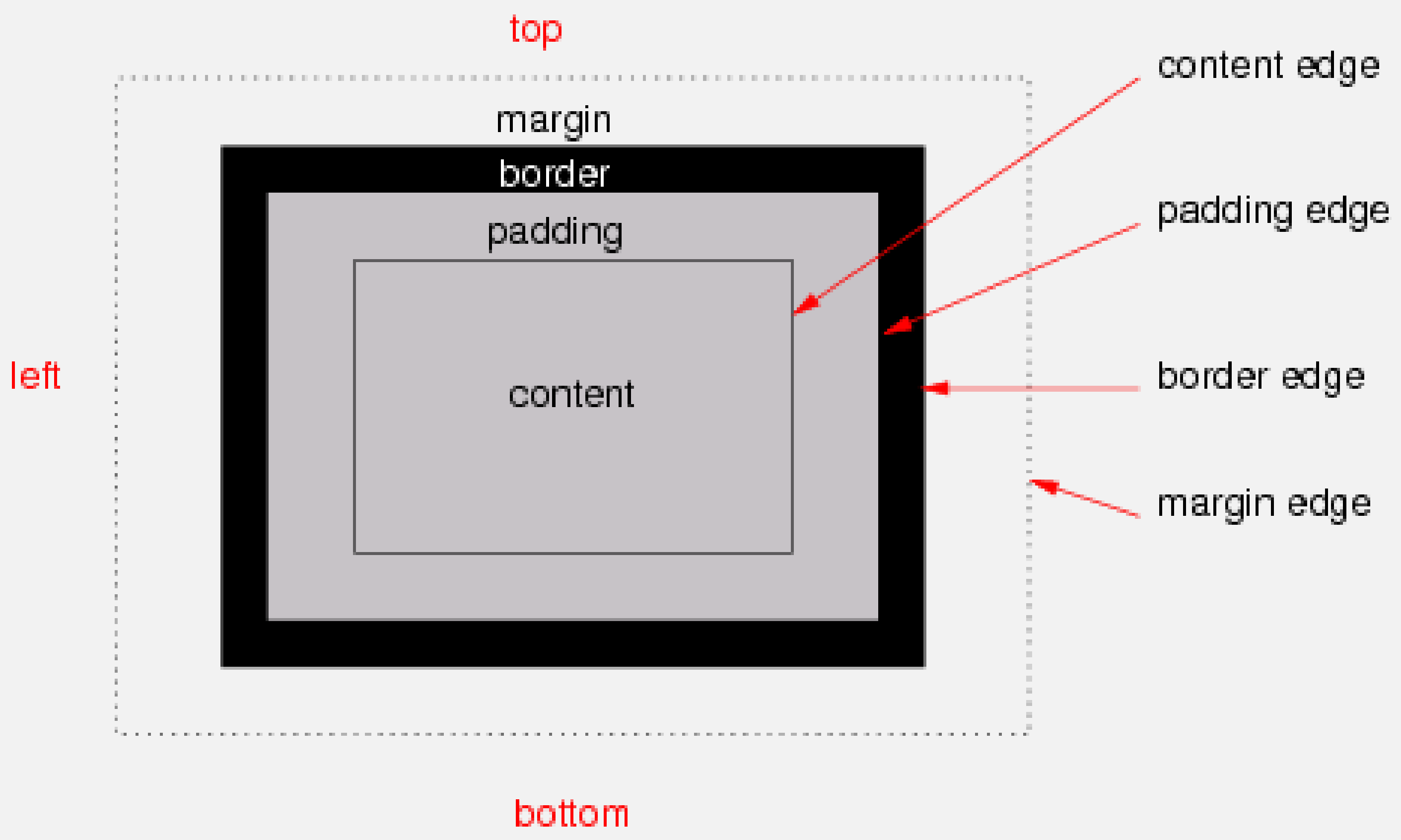
Box model

The box model describes how text and other objects are strung into lines, and lines into blocks (paragraphs), and how those blocks are put below each other or side by side with the help of margins and borders.

Each box has a content area (e.g., text, an image, etc.) and optional surrounding padding, border, and margin areas.

Box model

The CSS box model describes the rectangular boxes that are generated for elements in the document tree and laid out according to the visual formatting model.



https://drafts.csswg.org/css-box-3/

Box model: dimensions

The size of each area is specified by properties defined below.

- Margin properties: margin-top, margin-right, margin-bottom, margin-left, and margin

- Padding properties: padding-top, padding-right, padding-bottom, padding-left, and padding

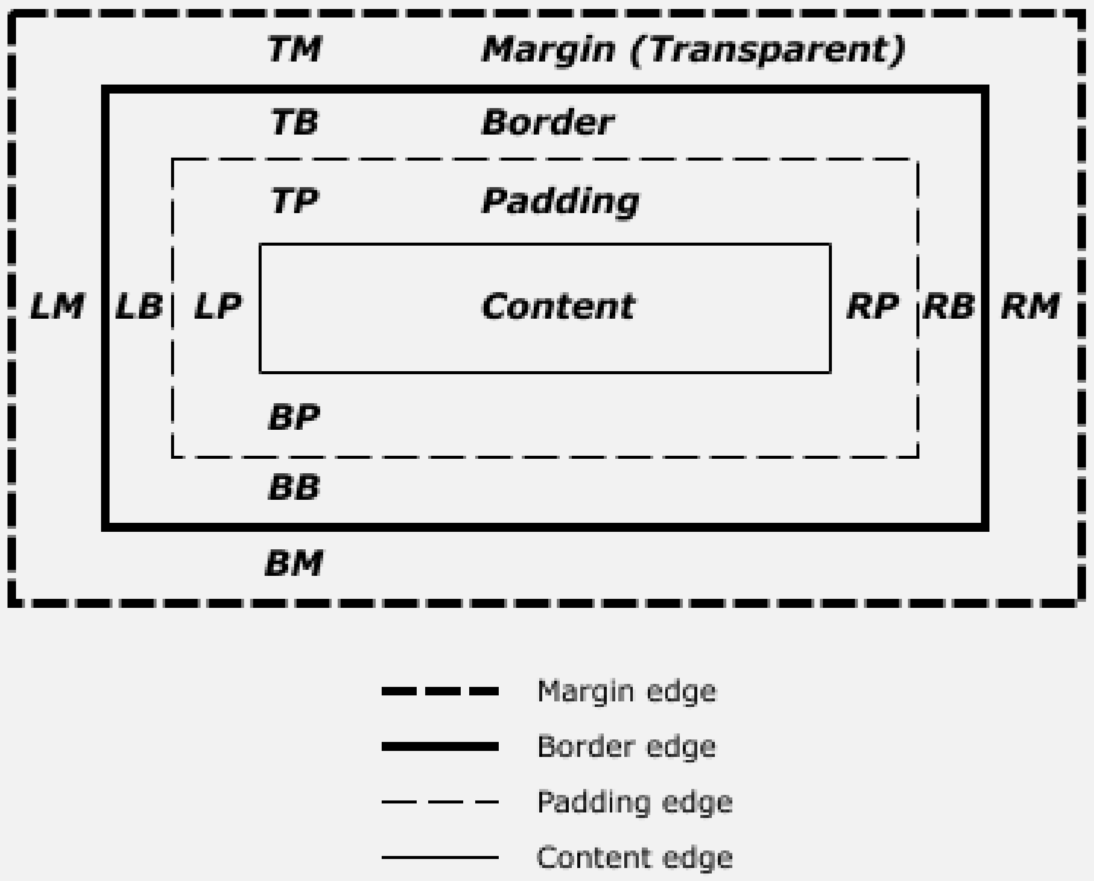
Box model: dimensions

The size of each area is specified by properties defined below.

- Border properties:
  - Border width: border-top-width, border-right-width, border-bottom-width, border-left-width, and border-width
  - Border color: border-top-color, border-right-color, border-bottom-color, border-left-color, and border-color
  - Border style: border-top-style, border-right-style, border-bottom-style, border-left-style, and border-style
  - Border shorthand properties: border-top, border-right, border-bottom, border-left, and border

Box model

The following diagram shows how these areas relate and the terminology used to refer to pieces of margin, border, and padding:



http://www.w3.org/TR/CSS2/box.html

Modules and the standardization process

CSS is divided into smaller components called modules.

Each module is an independent part of the language and moves towards standardization at its own pace.

While some modules are already W3C Recommendations, other still are early Working Drafts.

New modules are also added when new needs are identified.

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3

CSS recommendation levels

A few CSS modules are already fairly stable and have reached one of the three recommendation levels of the CSSWG:

- Candidate Recommendation,
- Proposed Recommendation or
- Recommendation.

These can be used without prefix and are pretty stable, though a few features can still be dropped at the Candidate Recommendation stage.

(https://www.w3.org/2018/Process-20180201/#RecsCR)

CSS3

CSS Color Module Level 3

Selectors Level 3

Media Queries

CSS Color Module Level 3

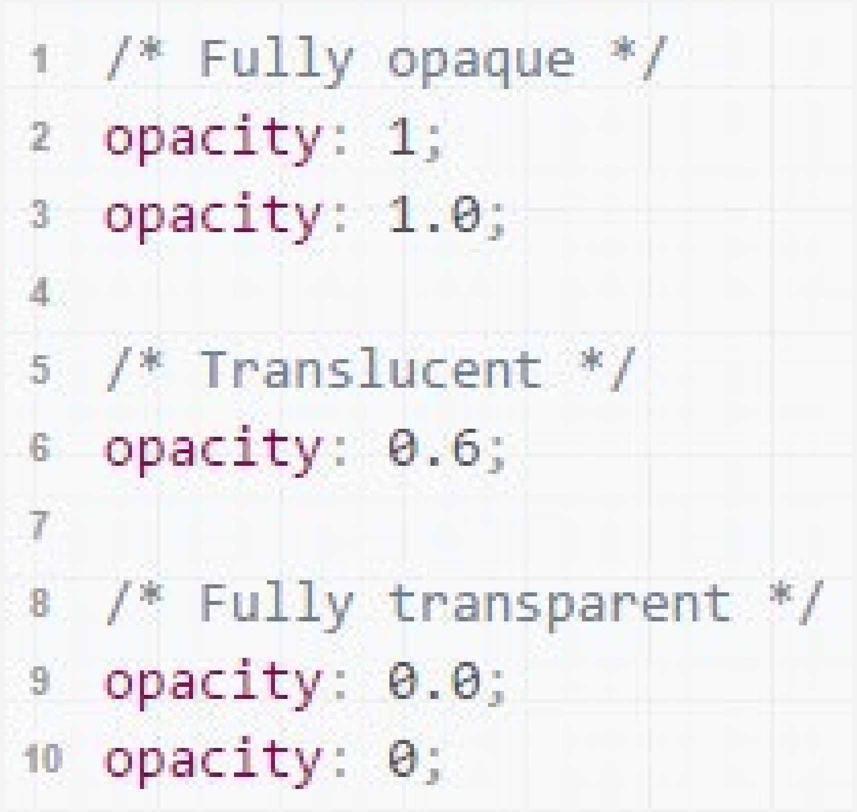
Adds opacity property

hsl(), hsla(), rgba() and rgb() functions

Defines the currentColor keyword as a valid color (which gets its value from the color-setting from an element or a cascading parent)

The transparent color is now a real color (thanks to the support for the alpha channel) and is a now an alias for rgba(0,0,0,0.0)

Opacity



```css
/* Fully opaque */
opacity: 1;
opacity: 1.0;

/* Translucent */
opacity: 0.6;

/* Fully transparent */
opacity: 0.0;
opacity: 0;
```

hsl(), hsla(), rgba() and rgb() functions

hsl()

Colors also can be defined the Hue-saturation-lightness model (HSL) using the hsl() functional notation.

Hue is represented as an angle of the color circle (i.e. the rainbow represented in a circle). This angle is given as a unitless <number>

By definition red=0=360, and the other colors are spread around the circle, so green=120, blue=240, etc.

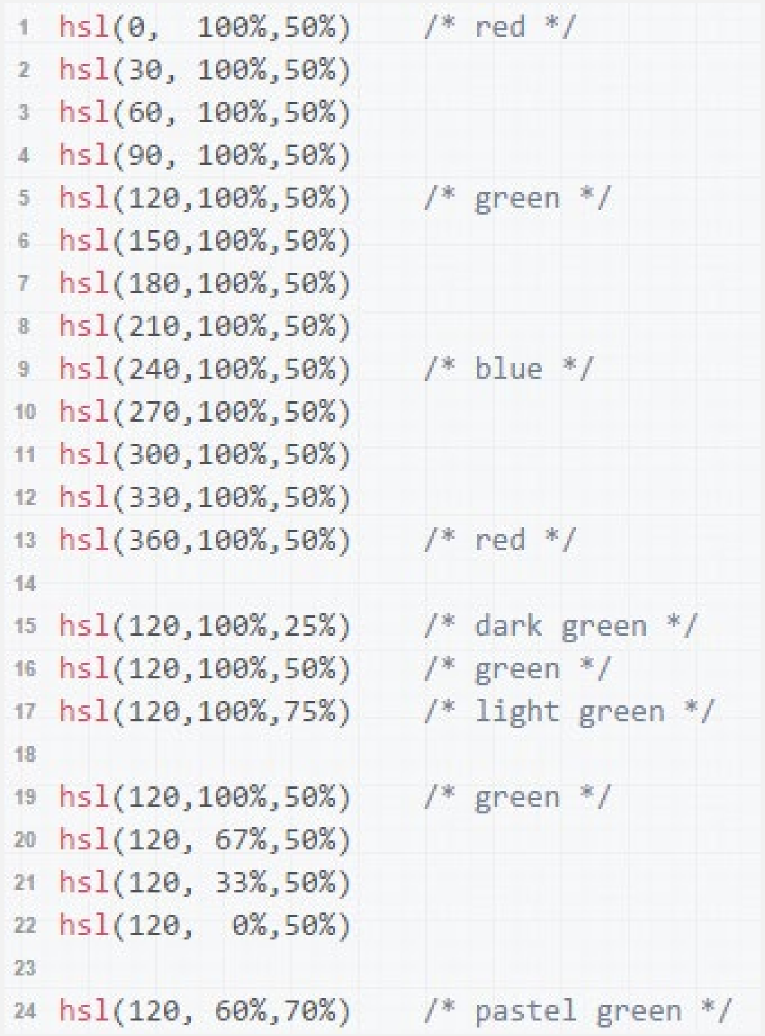
hsl(), hsla(), rgba() and rgb() functions

hsl()

- Saturation and lightness are represented as percentages.
  - 100% is full saturation, and 0% is a shade of grey.
  - 100% lightness is white, 0% lightness is black, and 50% lightness is "normal".

hsl(), hsla(), rgba() and rgb() functions

hsl()



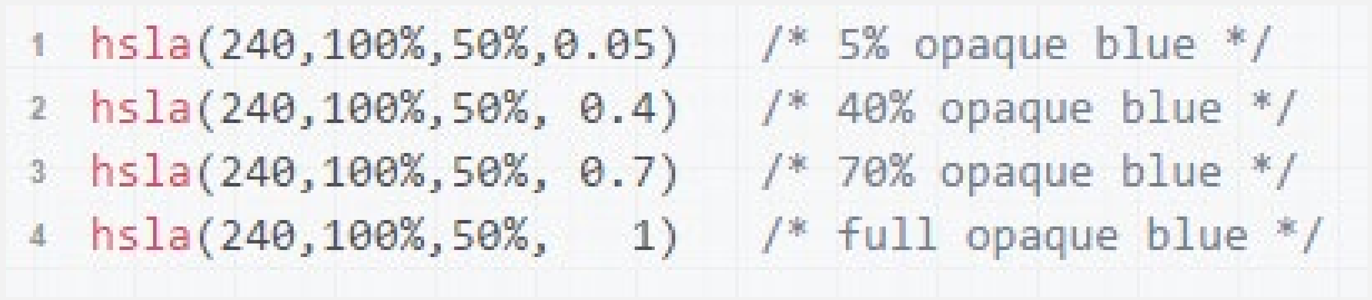
```
hsl(0,  100%,50%)    /* red */
hsl(30, 100%,50%)
hsl(60, 100%,50%)
hsl(90, 100%,50%)
hsl(120,100%,50%)    /* green */
hsl(150,100%,50%)
hsl(180,100%,50%)
hsl(210,100%,50%)
hsl(240,100%,50%)    /* blue */
hsl(270,100%,50%)
hsl(300,100%,50%)
hsl(330,100%,50%)
hsl(360,100%,50%)    /* red */

hsl(120,100%,25%)    /* dark green */
hsl(120,100%,50%)    /* green */
hsl(120,100%,75%)    /* light green */

hsl(120,100%,50%)    /* green */
hsl(120, 67%,50%)
hsl(120, 33%,50%)
hsl(120,  0%,50%)

hsl(120, 60%,70%)    /* pastel green */
```

hsl(), hsla(), rgba() and rgb() functions

hsla()

Colors can be defined in the hue-saturation-lightness-alpha model (HSLa) using the hsla() functional notation. HSLa extends the HSL color model to include the alpha channel, allowing specification of the opacity of a color.

a means opacity: 0=transparent; 1=opaque;



```
hsla(240,100%,50%,0.05)   /* 5% opaque blue */
hsla(240,100%,50%, 0.4)   /* 40% opaque blue */
hsla(240,100%,50%, 0.7)   /* 70% opaque blue */
hsla(240,100%,50%,   1)   /* full opaque blue */
```

hsl(), hsla(), rgba() and rgb() functions

rgb()

- Colors can be defined using the red-green-blue (RGB) model in two ways:
  - Hexadecimal notation #RRGGBB and #RGB
  - Functional Notation rgb(R,G,B)

hsl(), hsla(), rgba() and rgb() functions

rgb()

- Hexadecimal notation #RRGGBB and #RGB
  - "#", followed by six hexadecimal characters (0-9, A-F).
  - "#", followed by three hexadecimal characters (0-9, A-F).
  - The three-digit RGB notation (#RGB) and the six-digit form (#RRGGBB) are equal e.g., #f03 and #ff0033 represent the same color.

hsl(), hsla(), rgba() and rgb() functions

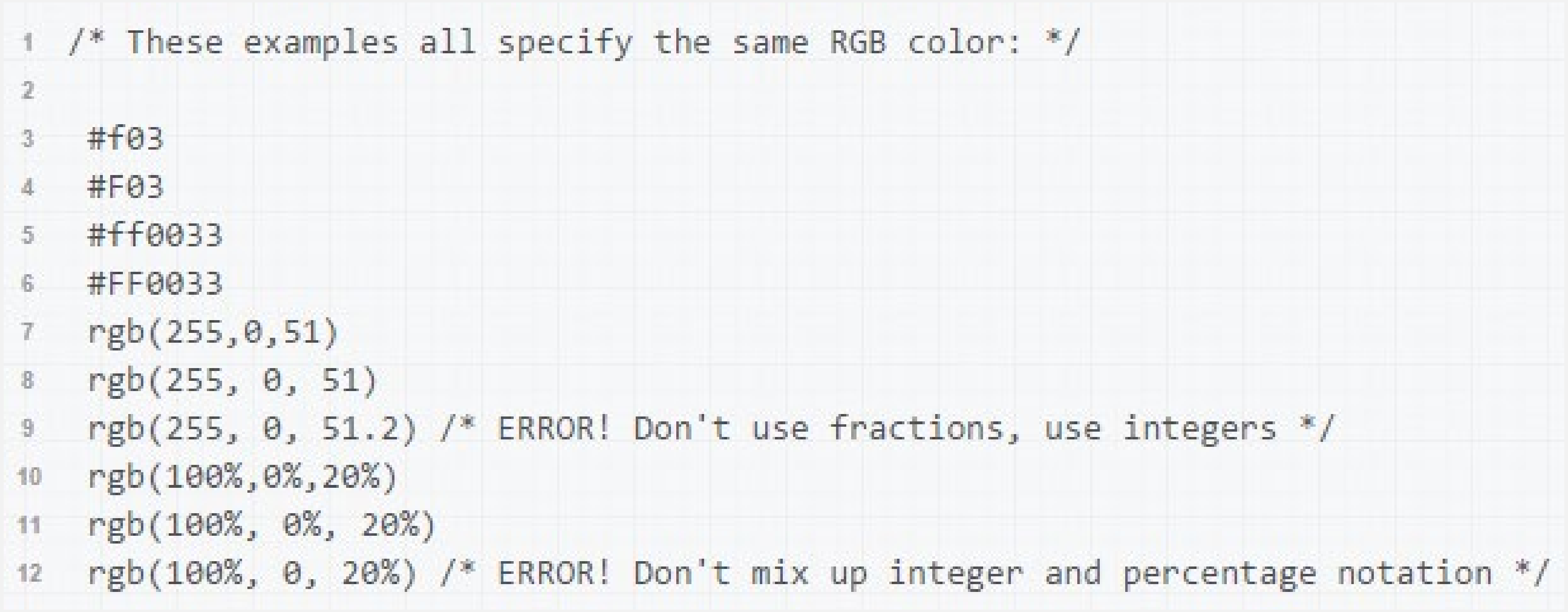
rgb()

- Functional Notation rgb(R,G,B)
  - "rgb", followed by three <integer> or three <percentage> values.

    The integer number 255 corresponds to 100%, and to F or FF in the hexadecimal notation.

hsl(), hsla(), rgba() and rgb() functions

rgb()



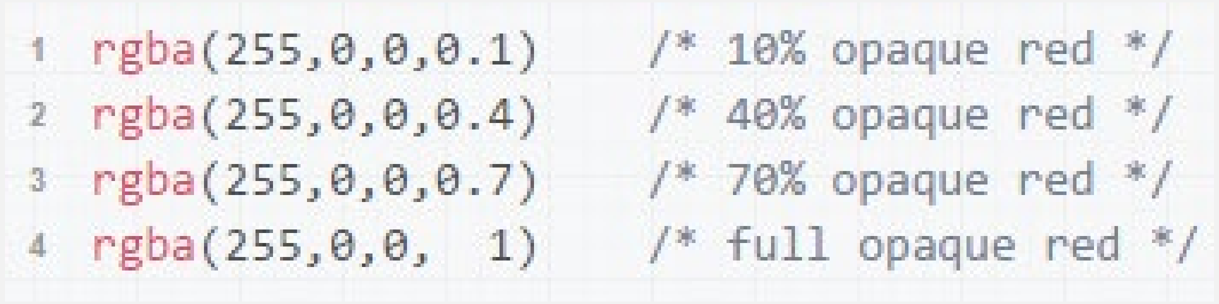
```
/* These examples all specify the same RGB color: */

 #f03
 #F03
 #ff0033
 #FF0033
 rgb(255,0,51)
 rgb(255, 0, 51)
 rgb(255, 0, 51.2) /* ERROR! Don't use fractions, use integers */
 rgb(100%,0%,20%)
 rgb(100%, 0%, 20%)
 rgb(100%, 0, 20%) /* ERROR! Don't mix up integer and percentage notation */
```

hsl(), hsla(), rgba() and rgb() functions

rgba()

Colors can be defined in the Red-green-blue-alpha model (RGBa) using the rgba() functional notation. RGBa extends the RGB color model to include the alpha channel, allowing specification of the opacity of a color.

a means opacity: 0=transparent; 1=opaque;



```
rgba(255,0,0,0.1)     /* 10% opaque red */
rgba(255,0,0,0.4)     /* 40% opaque red */
rgba(255,0,0,0.7)     /* 70% opaque red */
rgba(255,0,0,  1)     /* full opaque red */
```

Selectors Level 3

Substring matching attribute selectors

New pseudo-classes

Pseudo-elements are now characterized by two colons rather than one

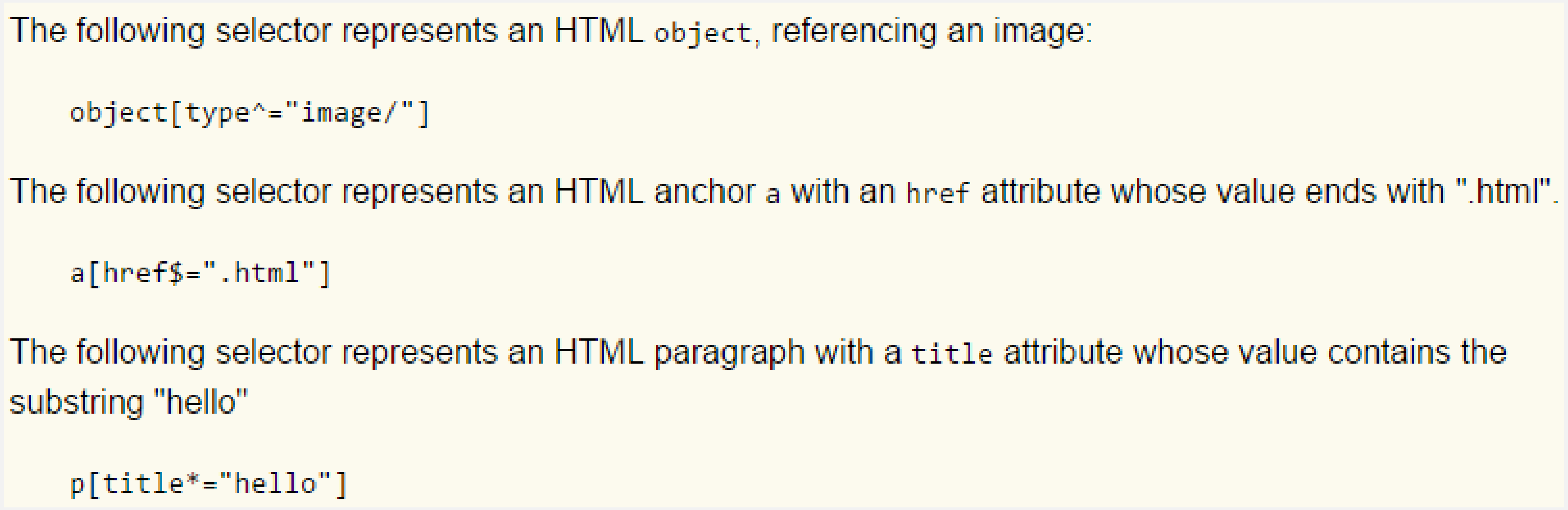
Selectors Level 3

Substring matching attribute selectors,

```
E[attribute^="value"]
E[attribute$="value"]
E[attribute*="value"]
```

Pattern	Represents
E[foo^="bar"]	an E element whose "foo" attribute value begins exactly with the string "bar"
E[foo$="bar"]	an E element whose "foo" attribute value ends exactly with the string "bar"
E[foo*="bar"]	an E element whose "foo" attribute value contains the substring "bar"

Substring matching attribute selectors


The following selector represents an HTML `object`, referencing an image:

```
object[type^="image/"]
```

The following selector represents an HTML anchor `a` with an `href` attribute whose value ends with ".html".

```
a[href$=".html"]
```

The following selector represents an HTML paragraph with a `title` attribute whose value contains the substring "hello"

```
p[title*="hello"]
```

https://drafts.csswg.org/selectors-3/

Selectors Level 3

New pseudo-classes:

- :target
- :enabled and :disabled
- :checked and :indeterminate
- :root
- :nth-child and :nth-last-child
- :nth-of-type and :nth-last-of-type
- :last-child
- :first-of-type and :last-of-type
- :only-child and :only-of-type
- :empty and :not

New pseudo-classes

- Structural pseudo classes. Note that counting, in this case, starts at 1, not 0

Pattern	Represents
E:root	an E element, root of the document
E:nth-child(n)	an E element, the n-th child of its parent
E:nth-last-child(n)	an E element, the n-th child of its parent, counting from the last one
E:nth-of-type(n)	an E element, the n-th sibling of its type
E:nth-last-of-type(n)	an E element, the n-th sibling of its type, counting from the last one

https://drafts.csswg.org/selectors-3/

Selectors Level 3

Pseudo-elements are now characterized by two colons rather than one:

- :after becomes ::after
- :before becomes ::before
- :first-letter becomes ::first-letter
- :first-line becomes ::first-line

Media Queries

Extends the former media type (print, screen, etc.) to a full language allowing queries on the device media capabilities like only screen and (color)

Media queries are not only used in CSS document but also in some attributes of HTML Elements, like the media attribute of the <link> element

https://drafts.csswg.org/mediaqueries-3/

Media Queries

A media query consists of a media type and zero or more expressions that check for the conditions of particular media features.

```
<link rel="stylesheet" media="screen and (color)" href="example.css" />
```

This example expresses that a certain style sheet (example.css) applies to devices of a certain media type (‘screen’) with certain feature (it must be a color screen).

Here the same media query written in an @import-rule in CSS:

```
@import url(color.css) screen and (color);
```

Responsive web design

Build a page that works on different devices (e.g., smartphones, tablets, laptops, desktop computers etc.), adapting its layout according to the device's capabilities and characteristics.

You can make it look different on small screens than on large screens or anywhere in between.

Different CSS rules target different devices.

The HTML never changes.

Before Responsive Web Design

Before Responsive Web Design, if you wanted to cater to mobile users, you'd build a separate site specifically for mobile.

Now you can build a single site that will work on all devices, now and in the future.

Components of a responsive page

Flexible images and media

- Assets are sized with percentages so that they scale up and down in the space available.

A flexible (fluid), grid-based layout

- All width properties are set in percentages so that layout components can shrink or expand.
- Other horizontal properties typically use a relative unit too (em, percentage, or rem).

Media queries

- Allows you to adjust your page design based on the width of the browser's viewable area and other characteristics.

Media Queries

You can target your CSS to specific media types in two ways:

- the media attribute of the link element

- the @media rule in your style sheet

Media queries enhance the media type methods, allowing you to target your styles to specific device features, they're particularly handy for changing how your site looks on different screen sizes.

Media Queries

Media features you can include in media queries:

- width
- height
- device-width
- device-height
- orientation
- aspect-ratio
- device-aspect-ratio
- color
- color-index
- monochrome
- resolution
- scan
- grid

Media Queries

For all but orientation, scan, and grid, you can include min- and max- prefixes.

- min- targets values that are "greater than or equal to"
- max- targets values that are "smaller that or equal to"

You'll use min-width and max-width over and over for responsive webpages.

Media Queries

Basic syntax:

For a link to an external style sheet:

```
<link rel="stylesheet" media="logic type and (feature: value)"
href="stylesheet.css" />
```

For a media query within a style sheet:

```
@media logic type and (feature: value){
  /*targeted CSS rules go here*/
}
```

For responsive pages you will place the media queries in your style sheet most of the time

Media Queries

logic

Optional and can have a value of either only or not.

- **only** ensures that older browsers don’t try to read the rest of the media query.
- **not** negates the result of the media query, making the opposite true.
- e.g., when used on the link element, **media="not screen"** will load the style sheet if the media type is anything other than screen.

type

The media type, such as **screen** or **print**.

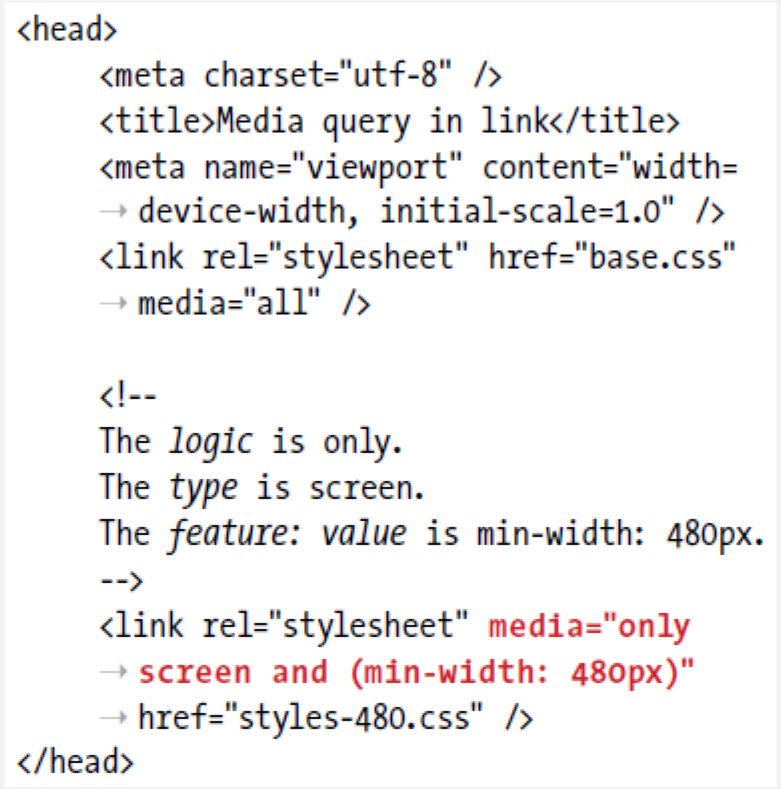
feature: value pair

Optional, but if present it must be enclosed in parentheses and preceded by the word **and**.

The feature is one of the predefined media features, such as **min-width**, **max-width**, or **resolution**. The value is optional for the color, color-index, and monochrome features.

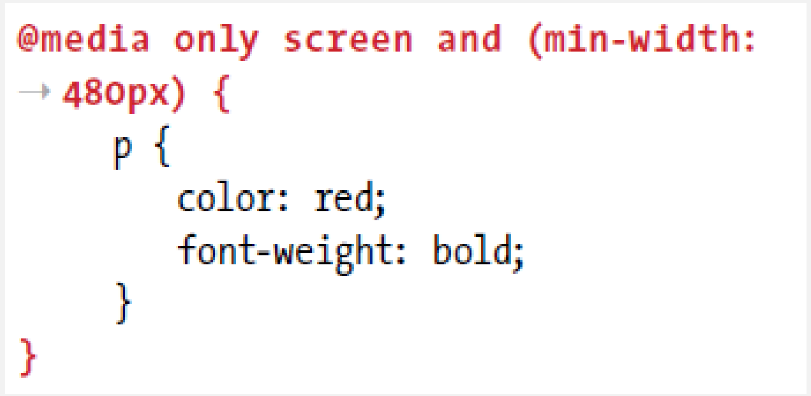
Media Queries

Example: Load and use the rules in styles-480.css only when the media type is screen and the minimum width of the viewport is 480px:



```
<head>
     <meta charset="utf-8" />
     <title>Media query in link</title>
     <meta name="viewport" content="width=
     → device-width, initial-scale=1.0" />
     <link rel="stylesheet" href="base.css"
     → media="all" />

     <!--
     The logic is only.
     The type is screen.
     The feature: value is min-width: 480px.
     -->
     <link rel="stylesheet" media="only
     → screen and (min-width: 480px)"
     → href="styles-480.css" />
</head>
```

Use the following rules only when the media type is screen and the minimum width of the viewport is 480px:



```
@media only screen and (min-width:
→ 480px) {
     p {
         color: red;
         font-weight: bold;
     }
}
```

Media Queries

In responsive web design lingo, you leverage media queries to define styles for each **breakpoint** in your page.

Breakpoint: each width at which your content would benefit from adjustments.

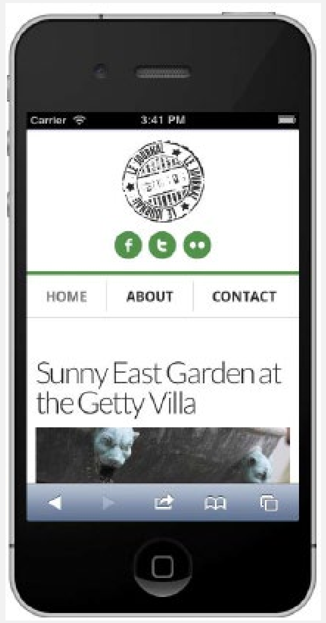
Keep in mind that for each min-width case with no max-width counterpart, the styles target devices at that min-width and all the way up, including the desktop and beyond.

Media Queries

Examples of breakpoints:

```
/* 20em (320px and up)
---------------------------------- */
@media only screen and (min-width: 20em) {
     .nav-main li {
        border-left: 1px solid #c8c8c8;
        display: inline-block;
        text-align: left;
     }

     .nav-main li:first-child {
        border-left: none;
     }

     .nav-main a {
        display: inline-block;
        font-size: 1em;
        padding: .5em .9em .5em 1.15em;
     }
}
```

- A minimum width of 20em (which generally is 320 pixels).
- This targets the iPhone, the iPod touch, and numerous Android and other mobile phones in portrait mode.

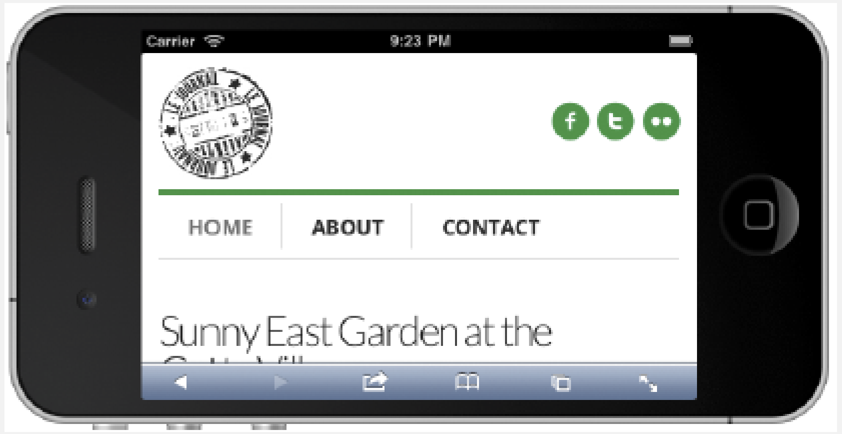


Media Queries

Examples of breakpoints:

```
/* 30em (480px and up)
---------------------------------- */
@media only screen and (min-width: 30em) {
     .masthead { position: relative; }

     .social-sites {
        position: absolute;
        right: -3px;
        top: 41px;
     }

     .logo {
        margin-bottom: 8px;
        text-align: left;
     }

     .nav-main {
        margin-top: 0;
     }
}
```

- A minimum width of 30em (which generally is 480 pixels).
- This targets larger mobile phones, as well as many of the 320-pixel devices when in landscape mode (the iPhone, the iPod touch, and certain Android models among them).

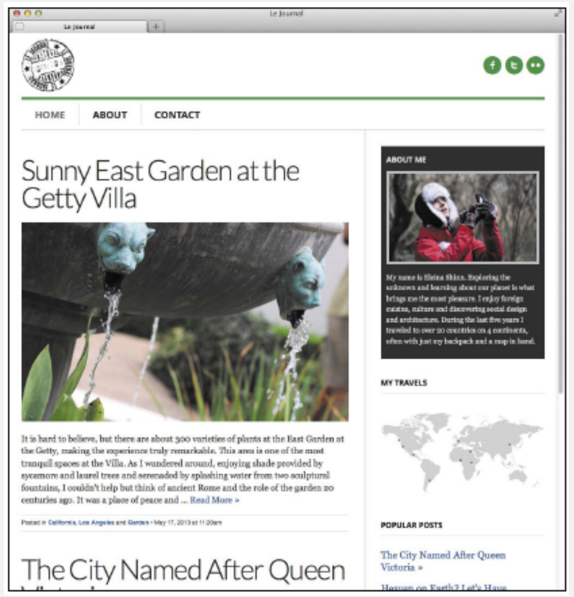


Media Queries

Examples of breakpoints:

```
/* 48em (768px and up)
----------------------------------- */
@media only screen and (min-width: 48em) {
     .container {
        background: url(../img/bg.png)
        → repeat-y 65.9375% 0;
        padding-bottom: 1.875em;
     }

     main {
        float: left;
        width: 62.5%; /* 600px/960px */
     }

     .sidebar {
        float: right;
        margin-top: 1.875em;
        width: 31.25%; /* 300px/960px */
     }

     .nav-main { margin-bottom: 0; }
}
```

- A minimum width of 48em (which generally is 768 pixels).
- This suits the iPad and other tablets, desktop browsers at typical widths, and everything wider.



Sunny East Garden at the Getty Villa

It is hard to believe, but there are about 300 varieties of plants at the East Garden at the Getty, making the experience truly remarkable. This area is one of the most tranquil spaces at the Villa. As I wandered around, enjoying shade provided by sycamore and laurel trees and serenaded by splashing water from two sculptural fountains, I couldn't help but think of ancient Rome and the role of the garden 20 centuries ago. It was a place of peace and ... Read More »

Posted in California, Los Angeles and Garden • May 17, 2013 at 11:20am

The City Named After Queen

My name is Elena Shinn. Exploring the unknown and learning about our planet is what brings me the most pleasure. I enjoy foreign cuisine, culture and discovering social design and architecture. During the last five years I traveled to over 20 countries on 4 continents, often with just my backpack and a map in hand.

The City Named After Queen Victoria »

The Viewport

The viewport is the area within a desktop or mobile browser that displays your page.

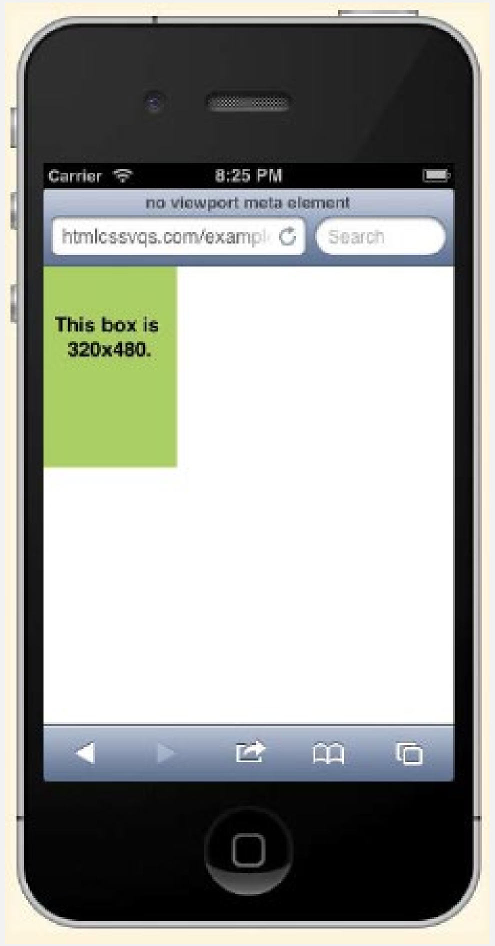
It doesn’t include things like the browser’s address bar or buttons, just the browsing area.

The media query width feature maps to the viewport width.

However, this is different than the device-width media feature, which is the width of the screen.

The Viewport

Sometimes the viewport width doesn't correspond to the (device) screen width.



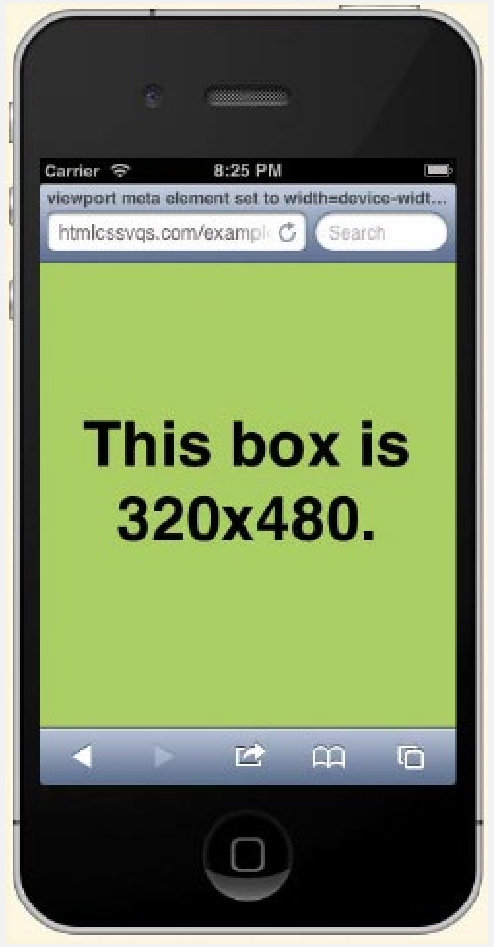
The Viewport

Fortunately, there's a quick solution. Simply add the viewport meta element to the head of your pages.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title> Your Page Title </title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
    </head>
    <body>
```

The Viewport

Viewport width is set to be the same as device width.



This box is 320x480.

CSS3

CSS Backgrounds and Borders Module Level 3

CSS Fonts Module Level 3

CSS Custom Properties for Cascading Variables Module Level 1

CSS Backgrounds and Borders Module Level 3

Support for multiple background images.

The background-repeat space and round values, and for the 2-value syntax of this CSS property.

The background-attachment local value.

The CSS background-origin, background-size, and background-clip properties.

Layering Multiple Background Images

```
div{
    background-image: url(flower.png), url(ball.png), url(grass.png);
    background-position: center center, 20% 80%, top left;
    background-origin: border-box, content-box;
    background-repeat: no-repeat;
}
```

CSS Backgrounds and Borders Module Level 3

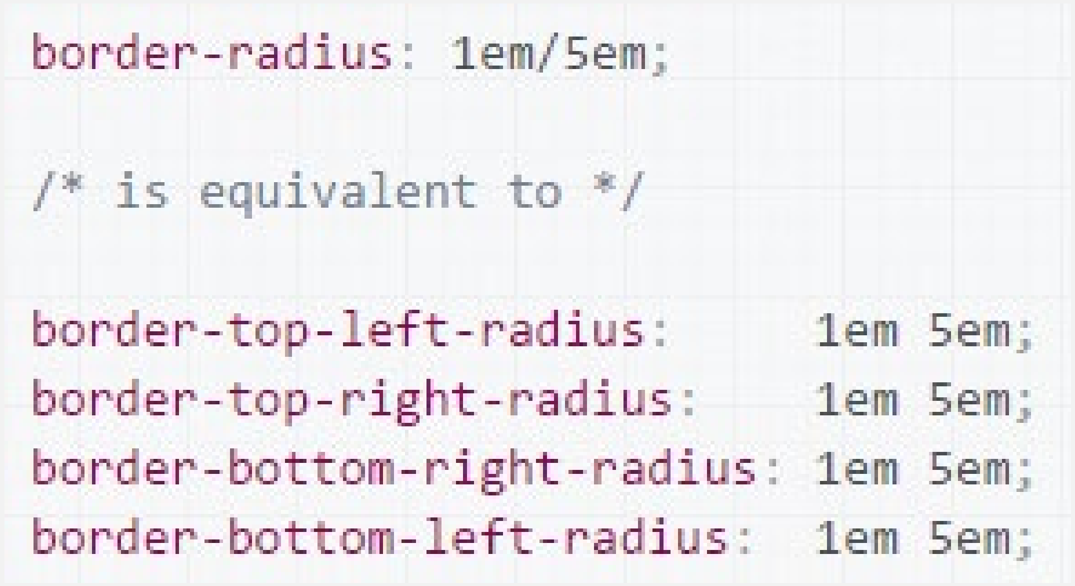
background-origin values:

- border-box: The background extends to the outside edge of the border (but underneath the border in z-ordering)
- padding-box: No background is drawn below the border (background extends to the outside edge of the padding)
- content-box: The background is painted within (clipped to) the content box

CSS Backgrounds and Borders Module Level 3

Support for curved border corners, with CSS

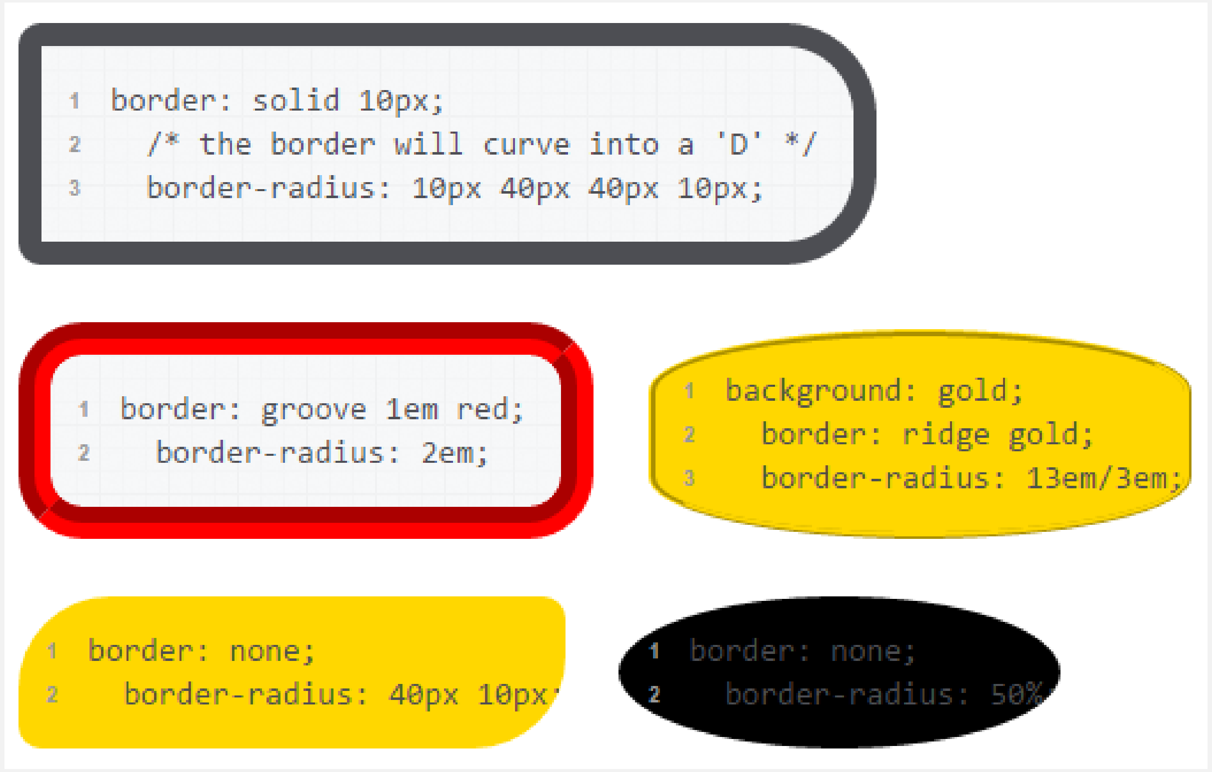
- border-radius
- border-top-left-radius
- border-top-right-radius
- border-bottom-left-radius
- border-bottom-right-radius



```
border-radius: 1em/5em;

/* is equivalent to */

border-top-left-radius:     1em 5em;
border-top-right-radius:    1em 5em;
border-bottom-right-radius: 1em 5em;
border-bottom-left-radius:  1em 5em;
```

CSS Backgrounds and Borders Module Level 3

Support for curved border corners, with CSS



```
border: solid 10px;
  /* the border will curve into a 'D' */
  border-radius: 10px 40px 40px 10px;
```

```
border: groove 1em red;
  border-radius: 2em;
```

```
background: gold;
  border: ridge gold;
  border-radius: 13em/3em;
```

```
border: none;
  border-radius: 40px 10px
```

```
border: none;
  border-radius: 50%
```

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

CSS Backgrounds and Borders Module Level 3

Support for curved border corners, with the CSS border-radius, border-top-left-radius, border-top-right-radius, border-bottom-left-radius, and border-bottom-right-radius properties.

Support for the use of an <image> as the border with the CSS border-image, border-image-source, border-image-slice, border-image-width, border-image-outset, and border-image-repeat properties.

Support for shadows of the element with the CSS box-shadow property.

CSS Multi-column Layout Module

Adds support for easy multi-column layouts using the CSS

- columns,
- column-count
- column-fill
- column-gap
- column-rule
- column-rule-color
- column-rule-style
- column-rule-width, column-span
- column-width
- break-after
- break-before
- break-inside

CSS Replaced Content Module Level 3

Support for

- linear-gradient()
- repeating-linear-gradient()
- radial-gradient()
- repeating-radial-gradient()

A gradient is an image that smoothly fades from one color to another. These are commonly used for subtle shading in background images, buttons, and many other things.

https://drafts.csswg.org/css-images-3/

The @font-face rule

The @font-face rule allows for linking to fonts that are automatically fetched and activated when needed.

The @font-face rule consists of the @font-face at-keyword followed by a block of descriptor declarations.

```
@font-face {
    font-family: myFont;
    src: url(Madison.ttf);
}

p {
    font-family: myFont;
}
```

https://drafts.csswg.org/css-fonts-3/

Cascading variables

CSS (finally) allows you to define cascading variables and use them as the values for CSS properties

This is done with two new CSS features:

- Custom properties: “*allow an author to assign arbitrary values to a property with an author-chosen name*”
- var() function: “*allow an author to then use those values in other properties elsewhere in the document*”

Cascading variables

How it works: first, define a custom property. In this case, we are defining it globally by defining it in :root

```
:root {
    --my-favourite-color: #7ff8ff;
}
```

Then, use it with the var() function

```
div {
    background-color: var(--my-favourite-color);
}
```

Cascading variables

A custom property has to be declared and used with two dashes:
--variable-name

Custom properties are case-sensitive

Custom properties can contain any possible values for CSS properties
(They’re also generally very permissive: https://drafts.csswg.org/css-variables/#syntax)

Cascading variables

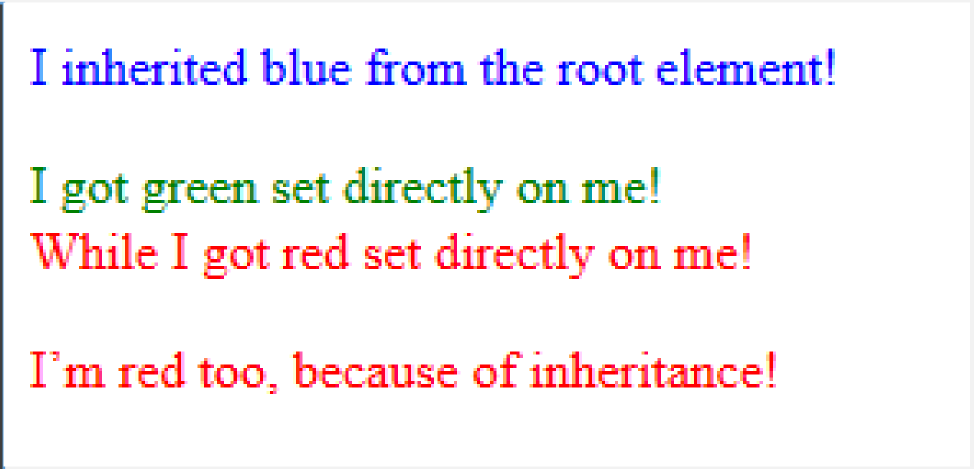
Custom properties can also be redefined in different selectors

```
:root {
    --my-favourite-color: #7ff8ff;
}

div{
    --my-favourite-color: #95ff75;
    background-color: var(--my-favourite-color);
    /* all divs will now use #95ff75 as their background color */
}
```

Cascading variables

Custom properties follow cascading rules, similar to regular CSS

```
*       { color: var(--color); }
:root   { --color: blue; }
div     { --color: green; }
#alert  { --color: red; }

<p>I inherited blue from the root element!</p>
<div>I got green set directly on me!</div>
<div id='alert'>
  While I got red set directly on me!
  <p>I’m red too, because of inheritance!</p>
</div>
```


I inherited blue from the root element!

I got green set directly on me!
While I got red set directly on me!

I’m red too, because of inheritance!

https://drafts.csswg.org/css-fonts-3/

Cascading variables

Custom properties can also have fallback values…

```
div{
    background-color: var(--color1, blue);
    /* if --color1 is not defined, use blue */
}
```

…which can also have fallback values…

```
div{
    background-color: var(--color1, var(--color2, blue));
    /* if neither --color1 nor --color2 are defined, use blue */
}
```

https://drafts.csswg.org/css-fonts-3/

CSS3

CSS Transitions

CSS Animations

CSS Transforms

Transitions

Effects that let an element gradually change from one style to another over a given duration.

To do this, you must specify two things:

- the CSS property you want to add an effect to
- the duration of the effect

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

Animations

Created in the @keyframe rule

Must be bound to a selector, otherwise the animation will have no effect.

Bind the animation to a selector (element) by specifying at least these two properties:

- the name of the animation
- the duration of the animation

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

Transformations

CSS transforms change the shape and position of the affected content without disrupting the normal document flow.

Variety of functions allowing you to do 2D or 3D transformations

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

Choosing Colours

https://www.websitebuilderexpert.com/designing-websites/how-to-choose-color-for-your-website/

Guide for choosing and using colours

Choosing Colours

https://color.adobe.com/

https://coolors.co/

Create colour schemes

Adobe Capture CC

https://play.google.com/store/apps/details?id=com.adobe.creativeapps.gather

https://itunes.apple.com/us/app/adobe-capture-cc/id1040200189?mt=8

Create colour schemes, patterns, and find fonts using photos and images

Homework

- Go to https://www.awwwards.com/
- Visit some awwward-winning websites, and see how they applied CSS3
  - Specifically breakpoints (to create responsive websites)
  - Also other features (different fonts, transitions, animations, etc.)
- Check out the linked article and adobe color