Viewport, Breakpoints, and Media/Feature Queries

GIT414

Introduction

CSS makes styling responsive websites possible, but how that code is written can change your site’s accessibility and/or usability in a way that you don’t intend

In this lecture, we’re going to talk about how we’d want to add styles to our site in a way that will be appropriate for all device widths, while respecting our users and their preferences

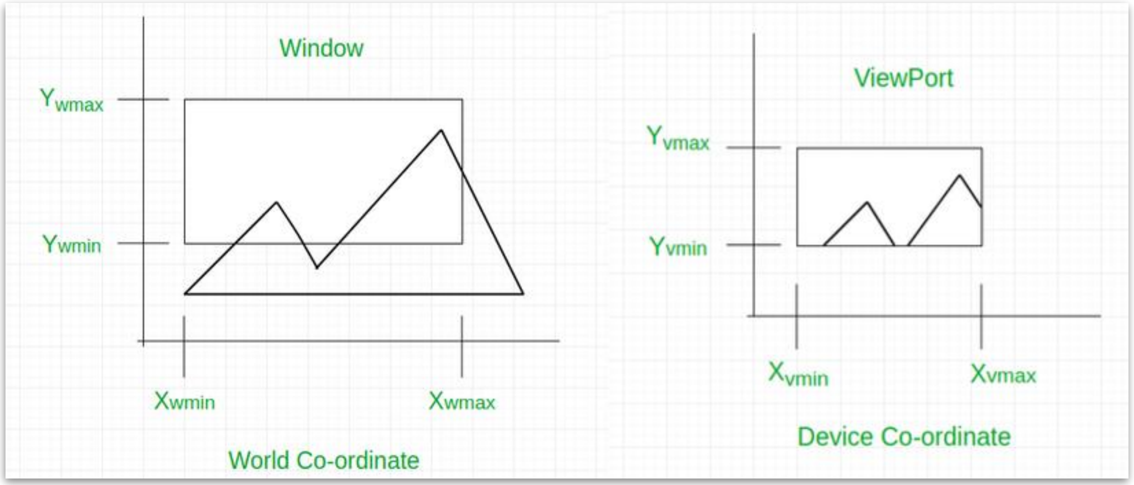
Remember, the viewport meta tag is required in the head of your HTML files in order for your site to be responsive

Viewport and Screen

Viewport and Screen Defined

Viewport: The viewport represents the size of the browser window where the page is being displayed, minus any menu bars or other UI components that are not part of the site

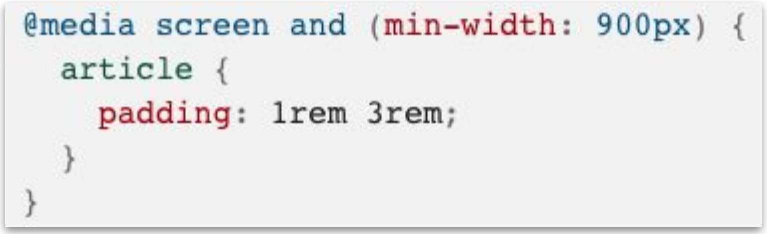
Screen: The screen is the actual screen on your device and its width will not necessarily be the same as that of the viewport



Notes About Screen and Viewport

When defining media queries, you want to be sure that you're targeting the correct item with those styles

In the examples we use in-class, we will use the meta viewport tag to define the viewport in our HTML, and widths to define media query breakpoints as needed. When referring to screen width, CSS uses the viewport. Do not take the user's ability to zoom on your screen away from them in the HTML viewport meta tag, that is an accessibility issue



```css
@media screen and (min-width: 900px) {
  article {
    padding: 1rem 3rem;
  }
}
```

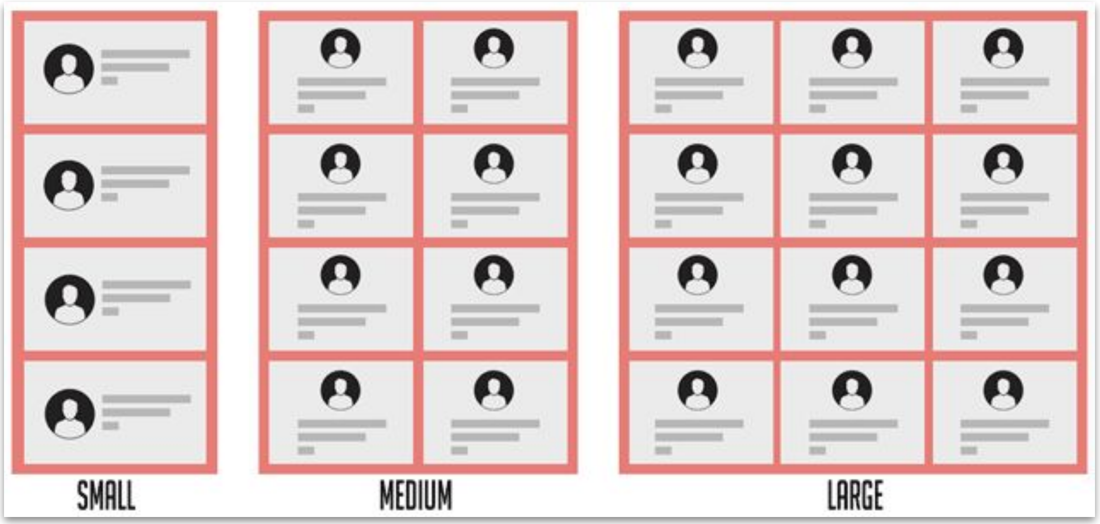
Breakpoints

What is a Breakpoint?

Breakpoints define a screen width where your layout no longer works with your content and styles

We can have many breakpoints or none, depending on the site layout and content

Define breakpoints in terms of the screen/viewport width where the styles need to change



Choosing Breakpoints

Breakpoints should not be tied to specific devices, as there are too many device widths to work with

Instead, use your site’s content to determine when you need your layout to change

- Items too small to see? Add a breakpoint
- Text too wide to be readable? Add a breakpoint

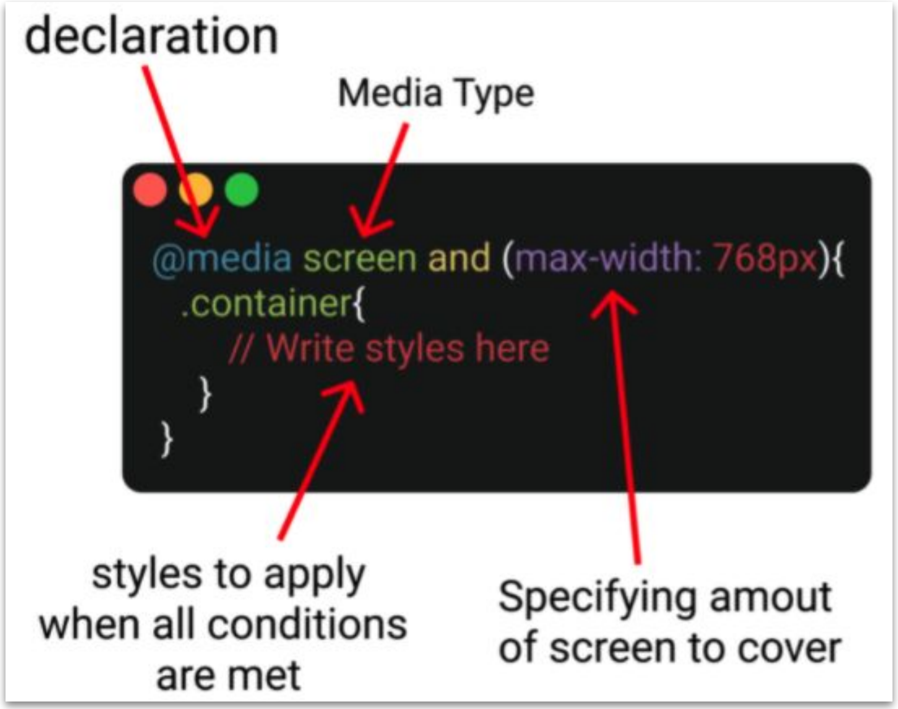
Media Queries

What is a Media Query?

Media queries allow you to deliver your site to the user in a way that will work best with their device or other output

Combining media queries with breakpoints means that you can offer multiple adjustments to your site layout based on many factors

You can also combine media queries with feature queries (covered later in the lecture)


```
@media screen and (max-width: 768px){
  .container{
     // Write styles here
   }
}
```
declaration
Media Type
styles to apply when all conditions are met
Specifying amout of screen to cover

How Do We Add a Media Query to CSS?

There are some best practices to keep in mind when writing your CSS and adding media queries:

1. Begin by writing all of the CSS you need to implement the mobile layout for your site (after the Reset/Normalizer styles and global styles are added)
2. Decide where your site layout needs to be adjusted by opening it in the browser and using the dev tools to change the width until part of the layout changes
3. Use widths to define the breakpoint for that screen width (you can use pixels in these, but ems or rems are more responsive)
4. Write the styles you need inside of that query
5. Repeat the process until your styles are complete, then test

What Would That Look Like in My CSS?

In this case, your global and mobile styles would not be inside of any type of media query, they would be the “default” styles for your site (mobile-first)

You’d then add media queries in terms of a min-width, meaning that once you reach that width, change the styles

- @media screen and (min-width: 650px){...} would start applying those styles when the viewport was at least 650px wide

This way, mobile styles will always work, but for wider screens you’d be able to add styles inside media queries later in the file that would be applied when needed

What Types of Media Can We Test For?

The @media rule recognizes four media types:

1. **all** - suitable for all devices
2. **print** - Intended for documents viewed on the screen in print preview mode (you can style your site in a more useful way for those that might print it out)
3. **screen** - intended for screens (mobile, tablet, desktop, etc.)
4. **speech** - intended for speech synthesizers

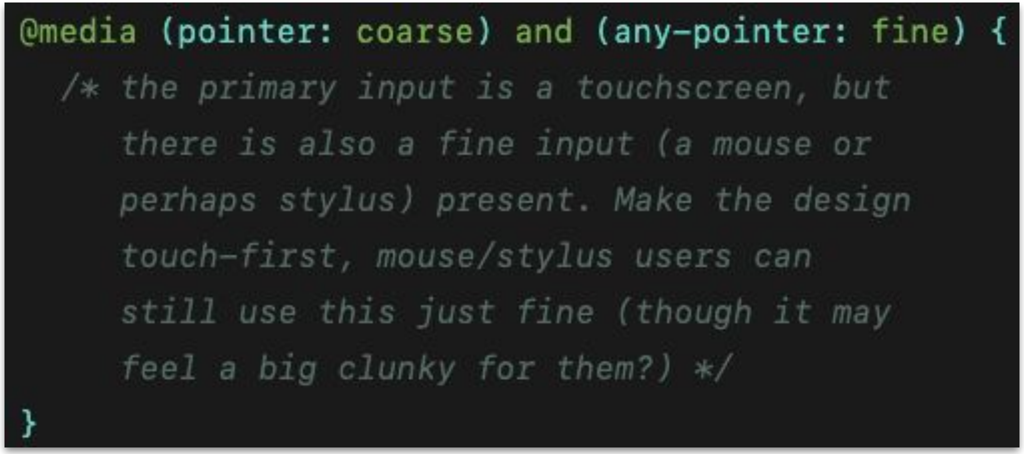
Media Query Best Practices

- Don’t base these on devices, base them on the page itself and its content
- Use responsive units when defining breakpoints (ems, rems)
- Write your CSS mobile-first, then add breakpoints for wider screen/viewport widths (you don’t need a media query for the mobile/global styles)
- Write your styles with responsive units for most items (max-widths will need to be pixels, though)

Using Media Queries to Detect Interaction Features

You can use media queries to check for more than just width, you can determine whether the browser has:

- The ability to support a hover action (touchscreens do not)
- The type of pointer being used (is it fine, like a mouse cursor, or coarse like a touch screen?)
- The device’s pixel density (important for very high definition screens where you may need to adjust layout)

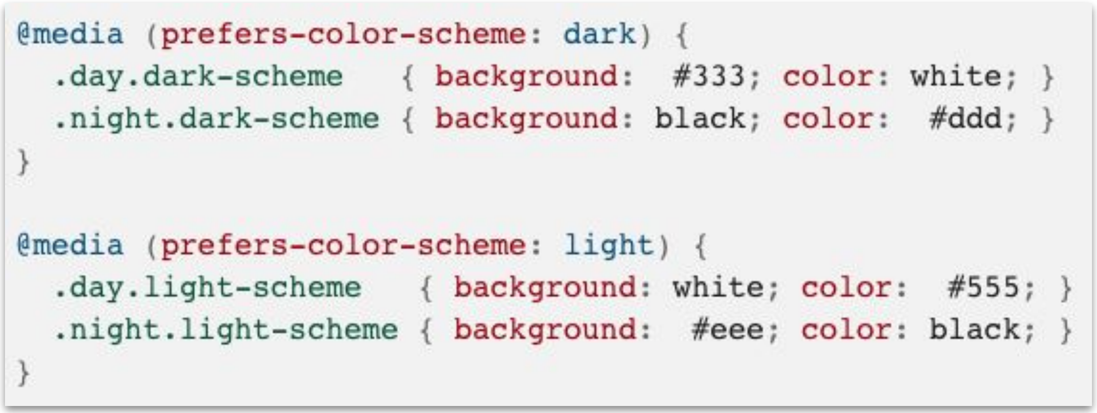


```
@media (pointer: coarse) and (any-pointer: fine) {
  /* the primary input is a touchscreen, but
     there is also a fine input (a mouse or
     perhaps stylus) present. Make the design
     touch-first, mouse/stylus users can
     still use this just fine (though it may
     feel a big clunky for them?) */
}
```

Interaction Features and Media Queries: Example

You can also check for a lot of user preferences and display your site in the way they prefer:

- prefers-color-scheme
- prefers-reduced-data
- prefers-reduced-motion (has accessibility implications)
- prefers-contrast (has accessibility implications)



```
@media (prefers-color-scheme: dark) {
  .day.dark-scheme   { background:  #333; color: white; }
  .night.dark-scheme { background: black; color:  #ddd; }
}

@media (prefers-color-scheme: light) {
  .day.light-scheme   { background: white; color:  #555; }
  .night.light-scheme { background:  #eee; color: black; }
}
```

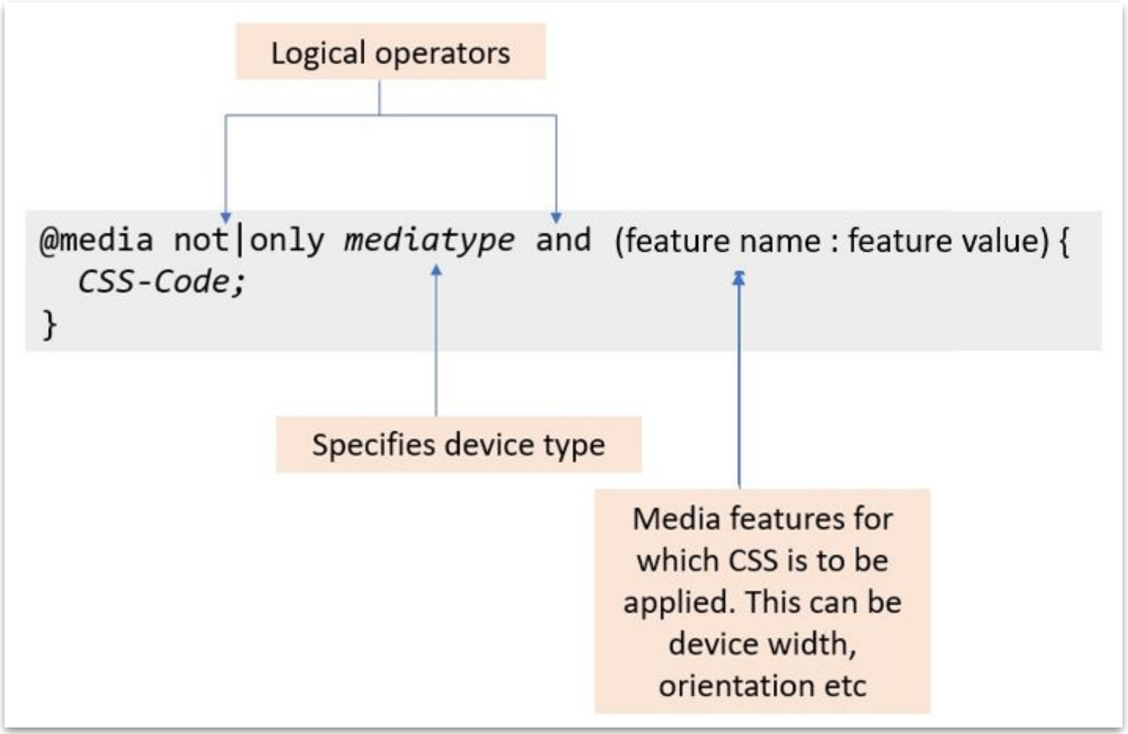
What Else Can You Check For With Media Queries?

- **width/height** - (viewport, can also use max- and min-)
- **Screen orientation**
- **Aspect ratio** - (viewport)
- **Color** (number of bits per color component of output device)
- **Color index** - Number of entries in the output device's color lookup table, or zero if the device does not use such a table
- **Monochrome** - Bits per pixel in the output device's monochrome frame buffer, or zero if the device isn't monochrome
- **Resolution** - Pixel density of the output device

Using Logical Operators with Media Queries

You can combine conditions with logical operators when defining a media query:

```
@media screen and (min-width: 64em){}

@media screen and (min-width: 64em)
and (resolution: 300dpi){}

@media only screen

  and (min-width: 320px)

  and (max-width: 480px)

  and (resolution: 150dpi) {

    body { line-height: 1.4; }

}
```

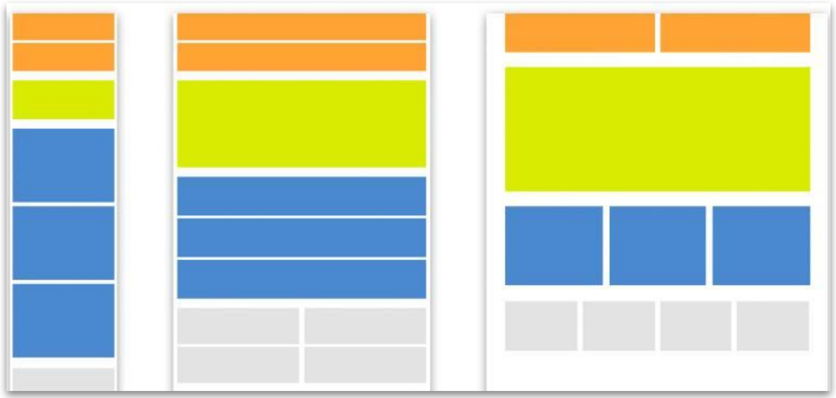

Logical operators
@media not|only mediatype and (feature name : feature value) {
CSS-Code;
}
Specifies device type
Media features for which CSS is to be applied. This can be device width, orientation etc

Using Media Queries

Place only styles you need inside your media queries

Well written CSS should mean that you have very little to adjust for different screen widths

Take advantage of the cascade, and use your global styles to implement CSS that you will see throughout the different screen widths (think of text colors, fonts, image and link styles, etc.)

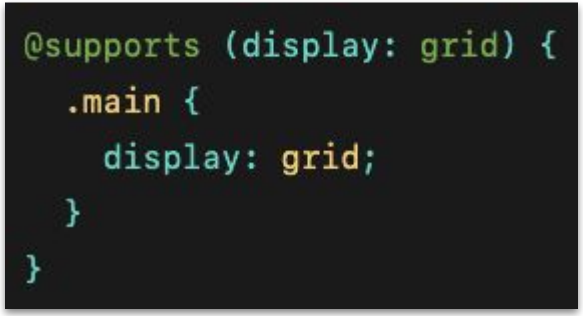


Feature Queries

What is a Feature Query?

Feature queries allow you to test whether or not a browser supports a feature, and use that information to implement some styles where they are supported. Use them for progressive enhancement

You want to avoid using “not” logic here: testing for grid support should give the answer you need, testing for **not** supporting grid could mean that your styles to be used when grid is not supported never actually are used (if the browser also doesn’t recognize @supports, it will never enter that block)

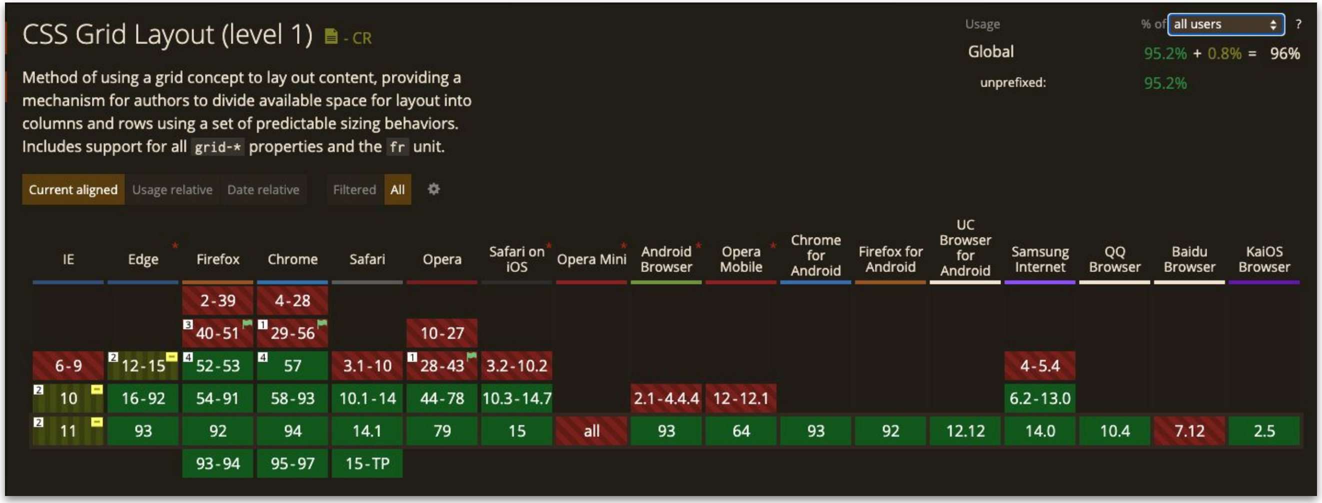


```
@supports (display: grid) {
  .main {
    display: grid;
  }
}
```

When Will You Need a Feature Query?

For styles that are not fully supported in the browsers your site may be viewed in, you need a feature query

Determine support by visiting the caniuse website and searching for your feature



CSS Grid Layout (level 1) - CR

Method of using a grid concept to lay out content, providing a mechanism for authors to divide available space for layout into columns and rows using a set of predictable sizing behaviors. Includes support for all `grid-*` properties and the `fr` unit.

Usage % of all users ?

Global 95.2% + 0.8% = 96%

unprefixed: 95.2%

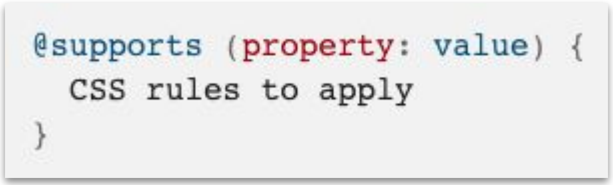
Current aligned | Usage relative | Date relative | Filtered | All

IE	Edge	Firefox	Chrome	Safari	Opera	Safari on iOS	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-39	4-28													
		[3] 40-51	[1] 29-56		10-27											
6-9	[2] 12-15	[4] 52-53	[4] 57	3.1-10	[1] 28-43	3.2-10.2							4-5.4			
[2] 10	16-92	54-91	58-93	10.1-14	44-78	10.3-14.7		2.1-4.4.4	12-12.1				6.2-13.0			
[2] 11	93	92	94	14.1	79	15	all	93	64	93	92	12.12	14.0	10.4	7.12	2.5
		93-94	95-97	15-TP												

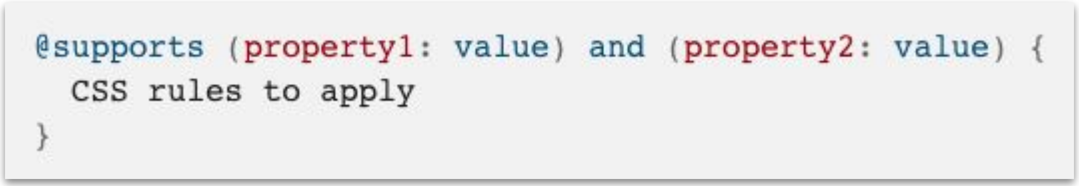
Feature Query Syntax

Syntax for feature queries is similar to syntax for media queries: You will include the @supports rule, and inside of the parentheses you add the CSS property and value you want to check for support of. Rules to use when that property is supported will go inside the curly brackets

You can also use logical operators in these to check for multiple features that work together



```
@supports (property: value) {
  CSS rules to apply
}
```

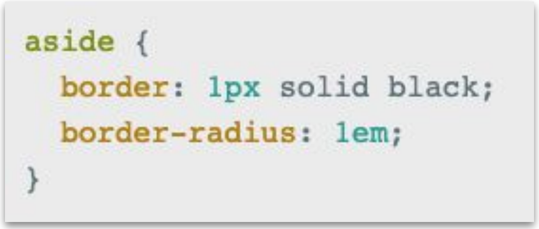


```
@supports (property1: value) and (property2: value) {
  CSS rules to apply
}
```

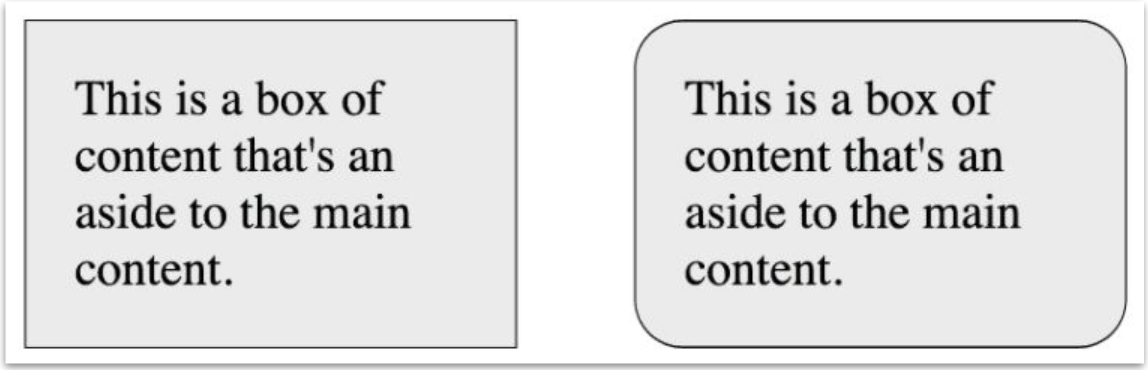
Using Feature Queries for Progressive Enhancement

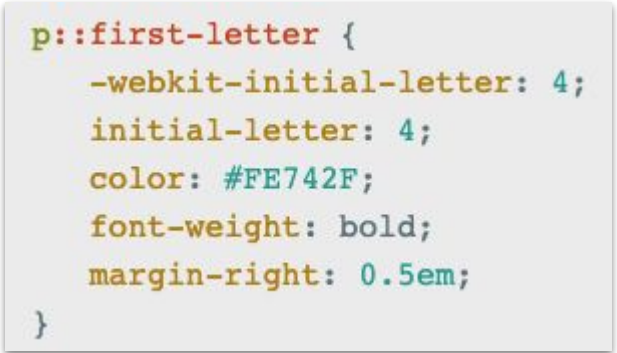
You don’t need to test for every feature used in your CSS unless it does something to break the display

Look at these examples of what happens when border-radius isn’t supported, and then look at what happens when the ::first-letter selector isn’t supported on the next slide

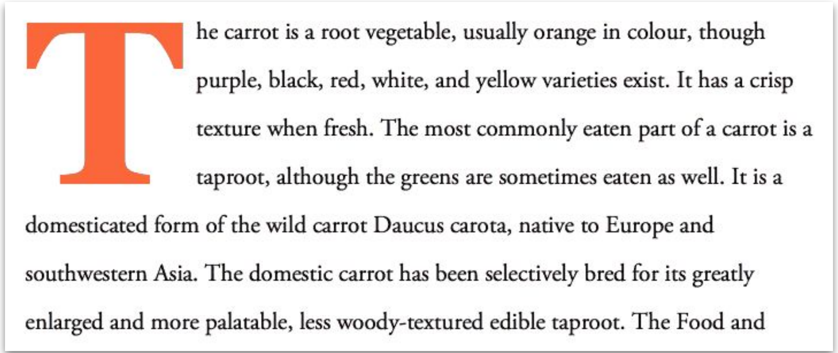


```
aside {
  border: 1px solid black;
  border-radius: 1em;
}
```

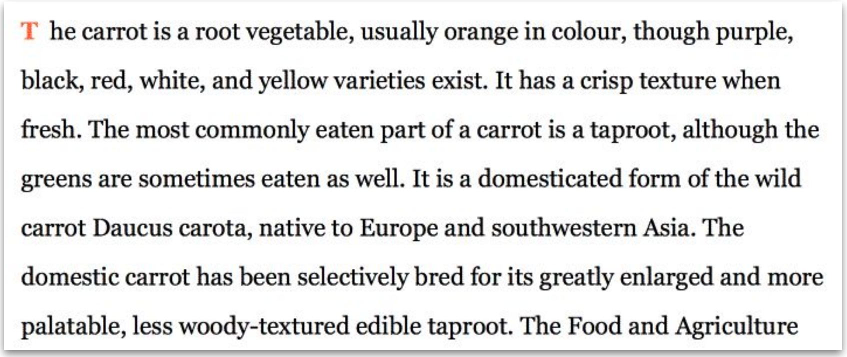

This is a box of content that's an aside to the main content.
This is a box of content that's an aside to the main content.



```
p::first-letter {
   -webkit-initial-letter: 4;
   initial-letter: 4;
   color: #FE742F;
   font-weight: bold;
   margin-right: 0.5em;
}
```



The carrot is a root vegetable, usually orange in colour, though purple, black, red, white, and yellow varieties exist. It has a crisp texture when fresh. The most commonly eaten part of a carrot is a taproot, although the greens are sometimes eaten as well. It is a domesticated form of the wild carrot Daucus carota, native to Europe and southwestern Asia. The domestic carrot has been selectively bred for its greatly enlarged and more palatable, less woody-textured edible taproot. The Food and



The carrot is a root vegetable, usually orange in colour, though purple, black, red, white, and yellow varieties exist. It has a crisp texture when fresh. The most commonly eaten part of a carrot is a taproot, although the greens are sometimes eaten as well. It is a domesticated form of the wild carrot Daucus carota, native to Europe and southwestern Asia. The domestic carrot has been selectively bred for its greatly enlarged and more palatable, less woody-textured edible taproot. The Food and Agriculture

Additional Resources

Additional Resources

Using Feature Queries

@supports

How @support Works

@media

Prefers-color-scheme

Viewport Concepts

How to Use CSS Breakpoints for Responsive Design

Beyond Media Queries: Using Newer HTML and CSS Features for Responsive Design