

[web.dev](#)

Responsive web design basics

Pete LePage [Twitter](#) [GitHub](#) [Glitch](#) [Homepage](#)

14-18 minutos

- [Home](#)
- [All articles](#)

How to create sites which respond to the needs and capabilities of the device they are viewed on.

Feb 12, 2019 — Updated May 14, 2020



- [Set the viewport](#)
- [Size content to the viewport](#)
- [Use CSS media queries for responsiveness](#)
- [How to choose breakpoints](#)
- [View media query breakpoints in Chrome DevTools](#)

The use of mobile devices to surf the web continues to grow at an astronomical pace, and these devices are often constrained by display size and require a different approach to how content is laid out on the screen.

Responsive web design, originally defined by [Ethan Marcotte in A List Apart](#), responds to the needs of the users and the devices they're using. The layout changes based on the size and

capabilities of the device. For example, on a phone users would see content shown in a single column view; a tablet might show the same content in two columns.



A multitude of different screen sizes exist across phones, "phablets," tablets, desktops, game consoles, TVs, and even wearables. Screen sizes are always changing, so it's important that your site can adapt to any screen size, today or in the future. In addition, devices have different features with which we interact with them. For example some of your visitors will be using a touchscreen. Modern responsive design considers all of these things to optimize the experience for everyone.

Set the viewport <#>

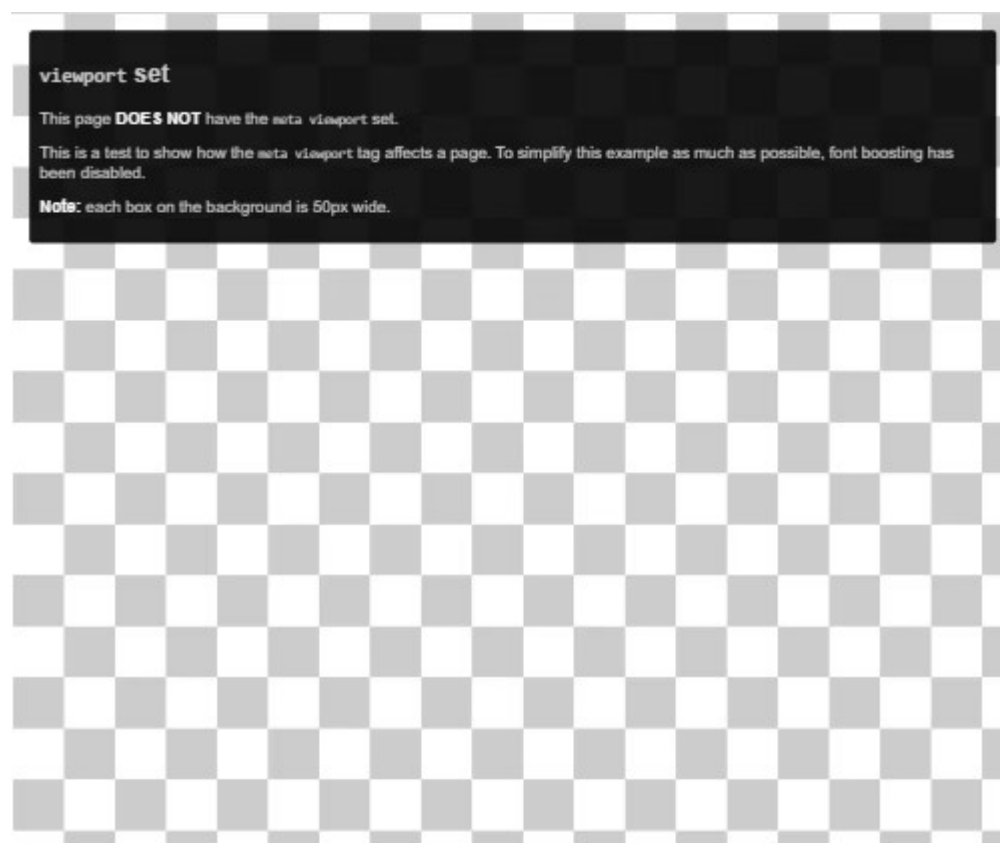
Pages optimized for a variety of devices must include a meta viewport tag in the head of the document. A meta viewport tag gives the browser instructions on how to control the page's dimensions and scaling.

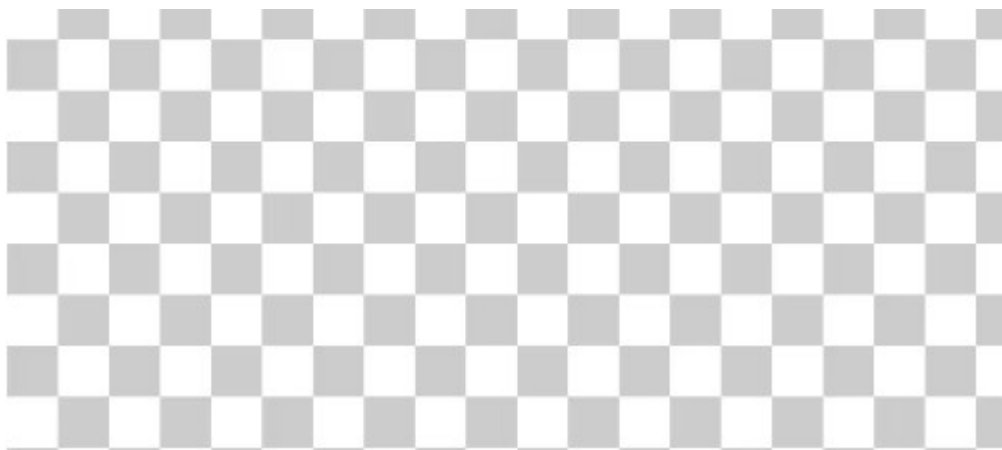
To attempt to provide the best experience, mobile browsers render the page at a desktop screen width (usually about 980px, though this varies across devices), and then try to make the content look better by increasing font sizes and scaling the content to fit the screen. This means that font sizes may appear

inconsistent to users, who may have to double-tap or pinch-to-zoom in order to see and interact with the content.

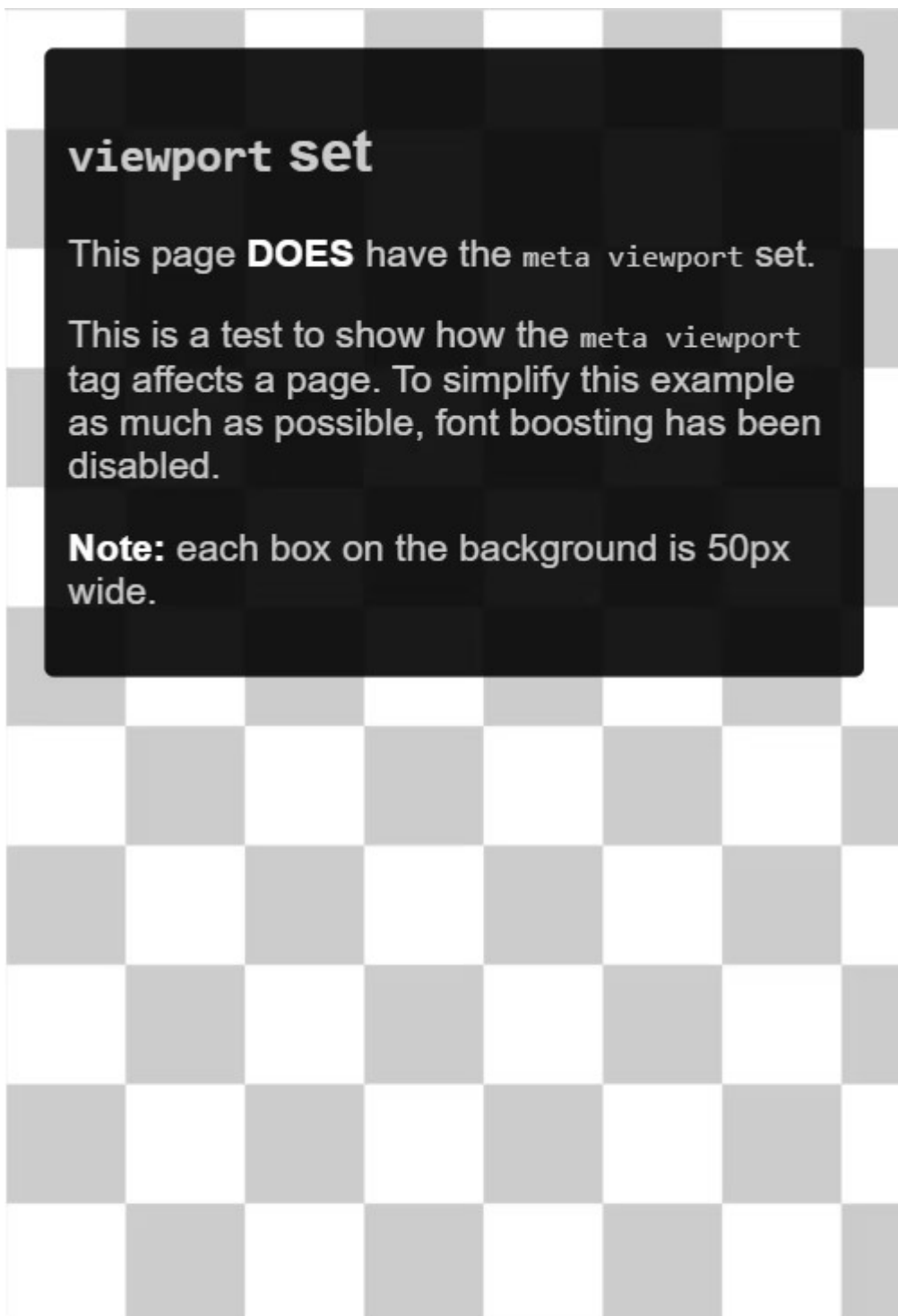
```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
    <meta name="viewport"
content="width=device-width, initial-scale=1">
    ...
  </head>
  ...
```

Using the meta viewport value `width=device-width` instructs the page to match the screen's width in device-independent pixels. A device (or density) independent pixel being a representation of a single pixel, which may on a high density screen consist of many physical pixels. This allows the page to reflow content to match different screen sizes, whether rendered on a small mobile phone or a large desktop monitor.





An example of how the page loads in a device without the viewport meta tag. [See this example on Glitch.](#)





An example of how the page loads in a device with the viewport meta tag. [See this example on Glitch.](#)

[Some browsers](#) keep the page's width constant when rotating to landscape mode, and zoom rather than reflow to fill the screen. Adding the value `initial-scale=1` instructs browsers to establish a 1:1 relationship between CSS pixels and device-independent pixels regardless of device orientation, and allows the page to take advantage of the full landscape width.

The [Does not have a `<meta name="viewport">` tag with `width` or `initial-scale`](#) Lighthouse audit can help you automate the process of making sure that your HTML documents are using the viewport meta tag correctly.

Ensure an accessible viewport <#>

In addition to setting an `initial-scale`, you can also set the following attributes on the viewport:

- `minimum-scale`
- `maximum-scale`
- `user-scalable`

When set, these can disable the user's ability to zoom the viewport, potentially causing accessibility issues. Therefore we would not recommend using these attributes.

Size content to the viewport <#>

On both desktop and mobile devices, users are used to scrolling websites vertically but not horizontally; forcing the user to scroll horizontally or to zoom out in order to see the whole page results in a poor user experience.

When developing a mobile site with a meta viewport tag, it's easy to accidentally create page content that doesn't quite fit within the specified viewport. For example, an image that is displayed at a width wider than the viewport can cause the viewport to scroll horizontally. You should adjust this content to fit within the width of the viewport, so that the user does not need to scroll horizontally.

The [Content is not sized correctly for the viewport](#) Lighthouse audit can help you automate the process of detecting overflowing content.

Images <#>

An image has fixed dimensions and if it is larger than the viewport will cause a scrollbar. A common way to deal with this problem is to give all images a `max-width` of `100%`. This will cause the image to shrink to fit the space it has, should the viewport size be smaller than the image. However because the `max-width`, rather than the `width` is `100%`, the image will not stretch larger than its natural size. It is generally safe to add the following to your stylesheet so that you will never have a problem with images causing a scrollbar.

```
img {  
  max-width: 100%;  
  display: block;  
}
```

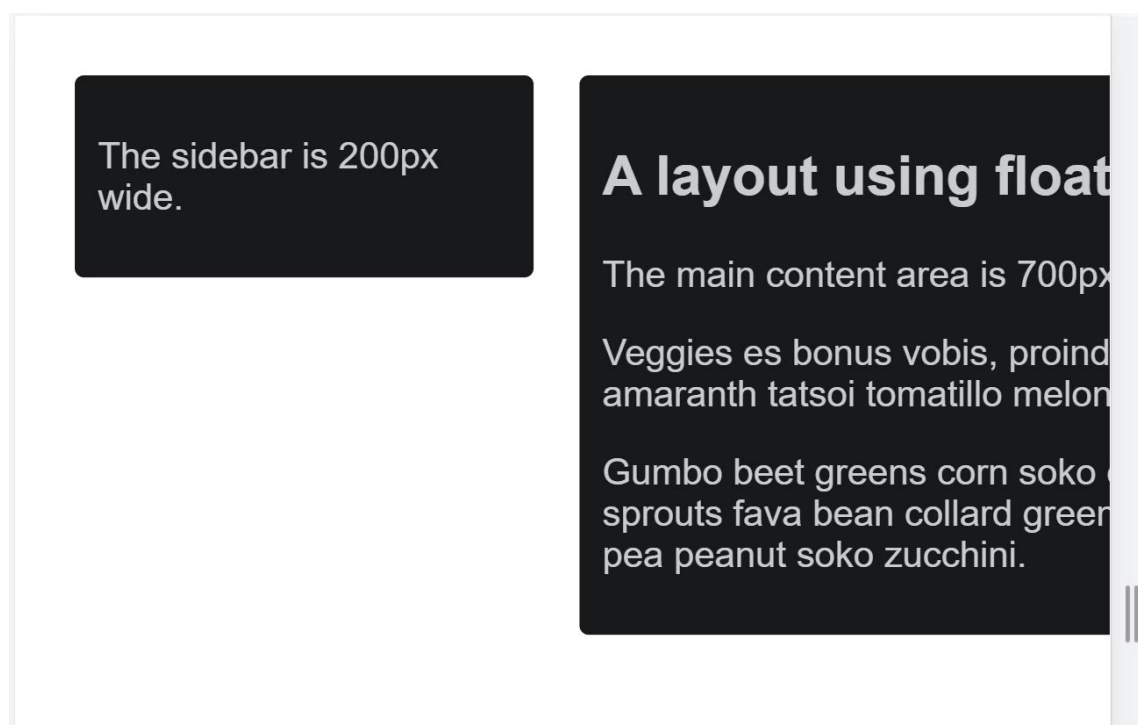
Add the dimensions of the image to the `img` element <#>

When using `max-width: 100%` you are overriding the natural dimensions of the image, however you should still use the `width` and `height` attributes on your `` tag. This is because modern browsers will use this information to reserve space for the image before it loads in, this will help to avoid [layout shifts](#) as content loads.

Layout <#>

Since screen dimensions and width in CSS pixels vary widely between devices (for example, between phones and tablets, and even between different phones), content should not rely on a particular viewport width to render well.

In the past, this required setting elements used to create layout in percentages. In the example below, you can see a two-column layout with floated elements, sized using pixels. Once the viewport becomes smaller than the total width of the columns, we have to scroll horizontally to see the content.



A floated layout using pixels. [See this example on Glitch.](#)

By using percentages for the widths, the columns always remain a certain percentage of the container. This means that the columns become narrower, rather than creating a scrollbar.

Modern CSS layout techniques such as Flexbox, Grid Layout, and Multicol make the creation of these flexible grids much easier.

Flexbox <#>

This layout method is ideal when you have a set of items of different sizes and you would like them to fit comfortably in a row or rows, with smaller items taking less space and larger ones getting more space.

```
.items {  
  display: flex;  
  justify-content: space-between;  
}
```

In a responsive design, you can use Flexbox to display items as a single row, or wrapped onto multiple rows as the available space decreases.

[Read more about Flexbox.](#)

CSS Grid Layout <#>

CSS Grid Layout allows for the straightforward creation of flexible grids. If we consider the earlier floated example, rather than creating our columns with percentages, we could use grid layout and the `fr` unit, which represents a portion of the available space in the container.

```
.container {  
  display: grid;
```



```
    grid-template-columns: 1fr 3fr;  
}
```

Grid can also be used to create regular grid layouts, with as many items as will fit. The number of available tracks will be reduced as the screen size shrinks. In the below demo, we have as many cards as will fit on each row, with a minimum size of 200px.

[Read more about CSS Grid Layout](#)

Multiple-column layout <#>

For some types of layout you can use Multiple-column Layout (Multicol), which can create responsive numbers of columns with the `column-width` property. In the demo below, you can see that columns are added if there is room for another 200px column.

[Read more about Multicol](#)

Sometimes you will need to make more extensive changes to your layout to support a certain screen size than the techniques shown above will allow. This is where media queries become useful.

Media queries are simple filters that can be applied to CSS styles. They make it easy to change styles based on the types of device rendering the content, or the features of that device, for example width, height, orientation, ability to hover, and whether the device is being used as a touchscreen.

To provide different styles for printing, you need to target a *type* of output so you could include a stylesheet with print styles as follows:

```
<!DOCTYPE html>
```

```
<html lang="en">
  <head>
    ...
    <link rel="stylesheet" href="print.css"
media="print">
    ...
  </head>
  ...
```

Alternatively, you could include print styles within your main stylesheet using a media query:

```
@media print {
  /* print styles go here */
}
```

For responsive web design, we are typically querying the *features* of the device in order to provide a different layout for smaller screens, or when we detect that our visitor is using a touchscreen.

Media queries based on viewport size <#>

Media queries enable us to create a responsive experience where specific styles are applied to small screens, large screens, and anywhere in between. The feature we are detecting here is therefore screen size, and we can test for the following things.

- width (min-width, max-width)
- height (min-height, max-height)
- orientation
- aspect-ratio

All of these features have excellent browser support, for more

details including browser support information see [width](#), [height](#), [orientation](#), and [aspect-ratio](#) on MDN.

Media queries based on device capability

Given the range of devices available, we cannot make the assumption that every large device is a regular desktop or laptop computer, or that people are only using a touchscreen on a small device. With some newer additions to the media queries specification we can test for features such as the type of pointer used to interact with the device and whether the user can hover over elements.

- `hover`
- `pointer`
- `any-hover`
- `any-pointer`

Try viewing this demo on different devices, such as a regular desktop computer and a phone or tablet.

These newer features have good support in all modern browsers. Find out more on the MDN pages for [hover](#), [any-hover](#), [pointer](#), [any-pointer](#).

Using `any-hover` and `any-pointer`

The features `any-hover` and `any-pointer` test if the user has the capability to hover, or use that type of pointer even if it is not the primary way they are interacting with their device. Be very careful when using these. Forcing a user to switch to a mouse when they are using their touchscreen is not very friendly! However, `any-hover` and `any-pointer` may be

useful if it is important to work out what kind of device a user has. For example, a laptop with a touchscreen and trackpad should match coarse and fine pointers, in addition to the ability to hover.

How to choose breakpoints <#>

Don't define breakpoints based on device classes. Defining breakpoints based on specific devices, products, brand names, or operating systems that are in use today can result in a maintenance nightmare. Instead, the content itself should determine how the layout adjusts to its container.

Pick major breakpoints by starting small, then working up <#>

Design the content to fit on a small screen size first, then expand the screen until a breakpoint becomes necessary. This allows you to optimize breakpoints based on content and maintain the least number of breakpoints possible.

Let's work through the example we saw at the beginning: the weather forecast. The first step is to make the forecast look good on a small screen.

New York, NY

Tuesday, April 15th

Overcast



58°F

Precipitation: 100%

Humidity: 97%

Wind: 4 mph SW

Pollen Count: 36

Today



68°
36°

Pollen 36

Wednesday



50°
39°






Pollen 36

Thursday



55°

Pollen 36

Thursday		39°	Pollen 36
Friday		54° 43°	Pollen 36
Saturday		64° 46°	Pollen 36
Sunday		64° 50°	Pollen 36
Monday		61° 50°	Pollen 36

The app at a narrow width.

Next, resize the browser until there is too much white space between the elements, and the forecast simply doesn't look as good. The decision is somewhat subjective, but above 600px is certainly too wide.

<div>New York, NY</div> <div>Tuesday, April 15th Overcast</div> <div> 58°F</div> <div>Precipitation: 100% Humidity: 97% Wind: 4 mph SW Pollen Count: 36</div>			
Today		68° 36°	Pollen 36
Wednesday		50° 39°	Pollen 36
Thursday		55° 20°	Pollen 36

The app at a point where we feel we should tweak the design.

To insert a breakpoint at 600px, create two media queries at the end of your CSS for the component, one to use when the browser is 600px and below, and one for when it is wider than 600px.

```
@media (max-width: 600px) {  
}
```

```
@media (min-width: 601px) {  
  
}
```

Finally, refactor the CSS. Inside the media query for a `max-width` of `600px`, add the CSS which is only for small screens. Inside the media query for a `min-width` of `601px` add CSS for larger screens.

Pick minor breakpoints when necessary <#>

In addition to choosing major breakpoints when layout changes significantly, it is also helpful to adjust for minor changes. For example, between major breakpoints it may be helpful to adjust the margins or padding on an element, or increase the font size to make it feel more natural in the layout.

Let's start by optimizing the small screen layout. In this case, let's boost the font when the viewport width is greater than `360px`. Second, when there is enough space, we can separate the high and low temperatures so that they're on the same line instead of on top of each other. And let's also make the weather icons a bit larger.

```
@media (min-width: 360px) {  
  body {  
    font-size: 1.0em;  
  }  
}  
  
@media (min-width: 500px) {  
  .seven-day-fc .temp-low,  
  .seven-day-fc .temp-high {  
    display: inline-block;  
    width: 45%;  
  }  
}
```

```
}

.seven-day-fc .seven-day-temp {
  margin-left: 5%;
}

.seven-day-fc .icon {
  width: 64px;
  height: 64px;
}
}
```

Similarly, for the large screens it's best to limit to maximum width of the forecast panel so it doesn't consume the whole screen width.

```
@media (min-width: 700px) {
  .weather-forecast {
    width: 700px;
  }
}
```

Optimize text for reading <#>

Classic readability theory suggests that an ideal column should contain 70 to 80 characters per line (about 8 to 10 words in English). Thus, each time the width of a text block grows past about 10 words, consider adding a breakpoint.

Bacon Ipsum

Bacon ipsum dolor sit amet pancetta salami beef ribs ribeye ham. Ribeye strip steak boudin t-bone chuck tri-tip kielbasa sirloin frankfurter shankle ball tip ham short loin hamburger pork chop. Doner jowl tail, shoulder chicken beef ribs ribeye leberkas. Tongue corned beef fatback turkey pork chop

corned beef fatback turkey pork chop
bacon. Pork meatloaf andouille doner.
Swine biltong ham pork chop tongue,
shankle shank sausage jowl landjaeger
bresaola boudin pork. Porchetta shank
beef tongue cow, pork flank shankle
rump pig.

Ground round shankle tongue chicken
kevin fatback pancetta boudin strip
steak doner shank drumstick pork loin
ball tip short ribs. Kevin rump chicken
ribeye salami tail andouille. Tenderloin
shoulder drumstick biltong jowl short

The text as read on a mobile device.

Bacon Ipsum

Bacon ipsum dolor sit amet pancetta salami beef ribs ribeye ham. Ribeye strip steak boudin t-bone chuck tri-tip kielbasa sirloin frankfurter shankle ball tip ham short loin hamburger pork chop. Doner jowl tail, shoulder chicken beef ribs ribeye leberkas. Tongue corned beef fatback turkey pork chop bacon. Pork meatloaf andouille doner. Swine biltong ham pork chop tongue, shankle shank sausage jowl landjaeger bresaola boudin pork. Porchetta shank beef tongue cow, pork flank shankle rump pig.

Ground round shankle tongue chicken kevin fatback pancetta boudin strip steak doner shank drumstick pork loin ball tip short ribs. Kevin rump chicken ribeye salami tail andouille. Tenderloin shoulder drumstick biltong jowl short ribs capicola. Ham hock pastrami spare ribs tail doner frankfurter.

Pancetta ground round hamburger, tenderloin tail porchetta filet mignon prosciutto bacon. Prosciutto rump spare ribs boudin meatloaf doner flank chuck. Prosciutto leberkas ground round, sirloin tail flank filet mignon spare ribs ham turducken jowl swine pork loin shankle. Tri-tip rump doner andouille tail, bacon ham jerky. Rump shank pastrami, sausage brisket chicken ribeye frankfurter andouille pork chop pork pancetta. Jowl tongue filet mignon jerky, prosciutto strip steak pastrami rump.

Bresaola pork kielbasa ham hock tail cow short ribs sirloin kevin fatback

The text as read on a desktop browser with a breakpoint added to constrain the line length.

Let's take a deeper look at the above blog post example. On smaller screens, the Roboto font at 1em works perfectly giving 10 words per line, but larger screens require a breakpoint. In this case, if the browser width is greater than 575px, the ideal content width is 550px.

```
@media (min-width: 575px) {
  article {
    width: 550px;
    margin-left: auto;
    margin-right: auto;
  }
}
```

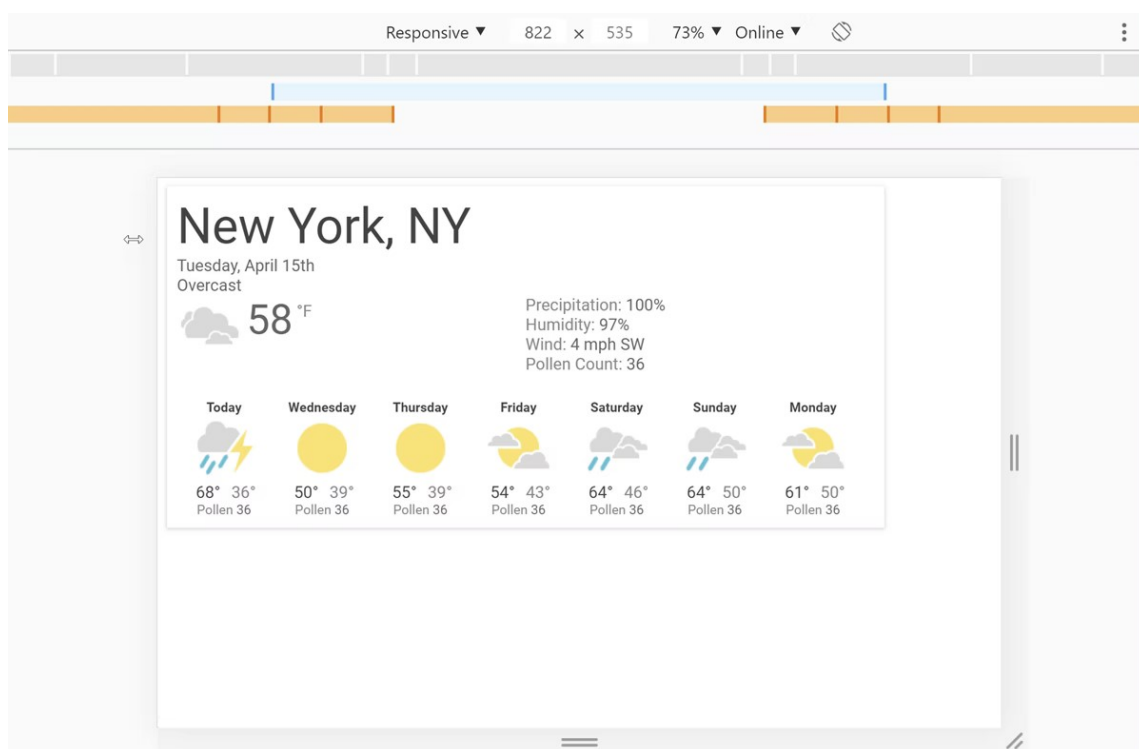


```
}
```

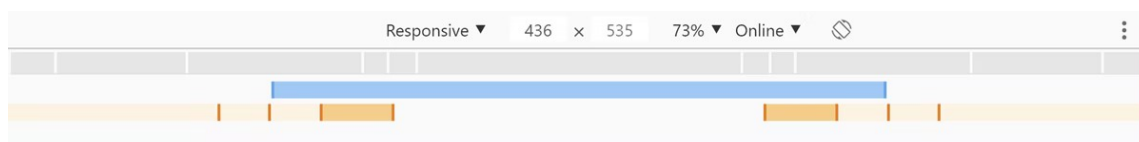
Avoid simply hiding content

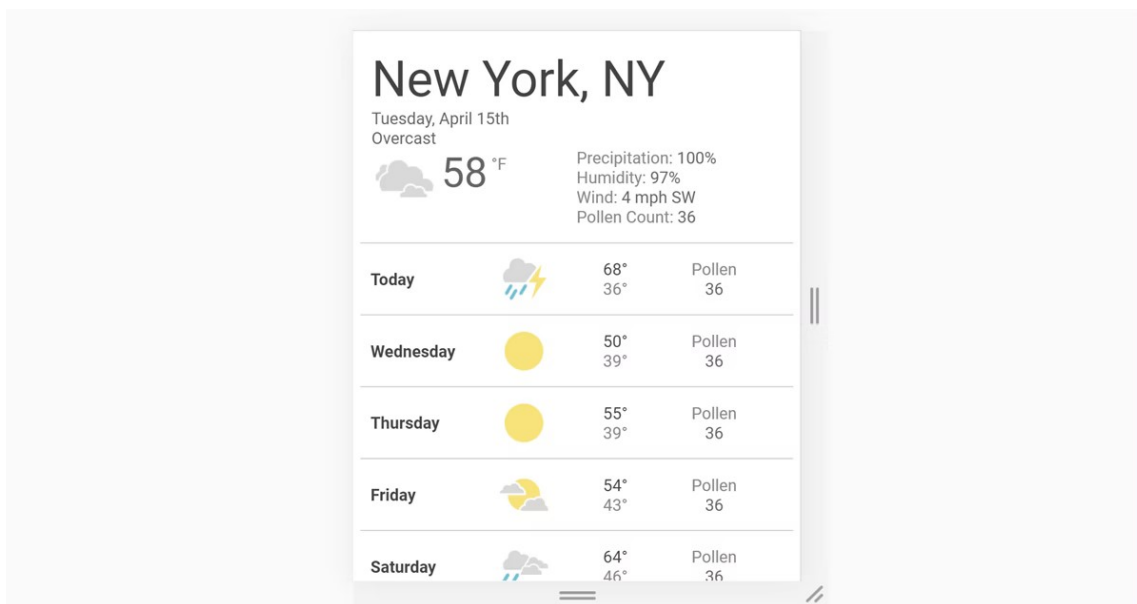
Be careful when choosing what content to hide or show depending on screen size. Don't simply hide content just because you can't fit it on the screen. Screen size is not a definitive indication of what a user may want. For example, eliminating the pollen count from the weather forecast could be a serious issue for spring-time allergy sufferers who need the information to determine if they can go outside or not.

Once you've got your media query breakpoints set up, you'll want to see how your site looks with them. You could resize your browser window to trigger the breakpoints, but Chrome DevTools has a built-in feature that makes it easy to see how a page looks under different breakpoints.



DevTools showing the weather app as it looks at a wider viewport size.





DevTools showing the weather app as it looks at a narrower viewport size.

To view your page under different breakpoints:

[Open DevTools](#) and then turn on [Device Mode](#). This opens in [responsive mode](#) by default.

To see your media queries, open the Device Mode menu and select [Show media queries](#) to display your breakpoints as colored bars above your page.

Click on one of the bars to view your page while that media query is active. Right-click on a bar to jump to the media query's definition.

Last updated: May 14, 2020 — [Improve article](#)

← [Return to all articles](#)