

www.agilityio.com

Responsive web design Notes

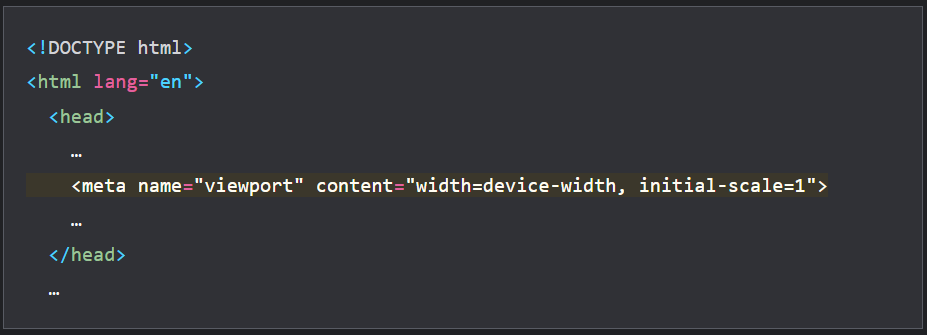
[**Docs: Responsive web design basics**](https://web.dev/responsive-web-design-basics/#viewport)

**Chapter 0: Introduction**

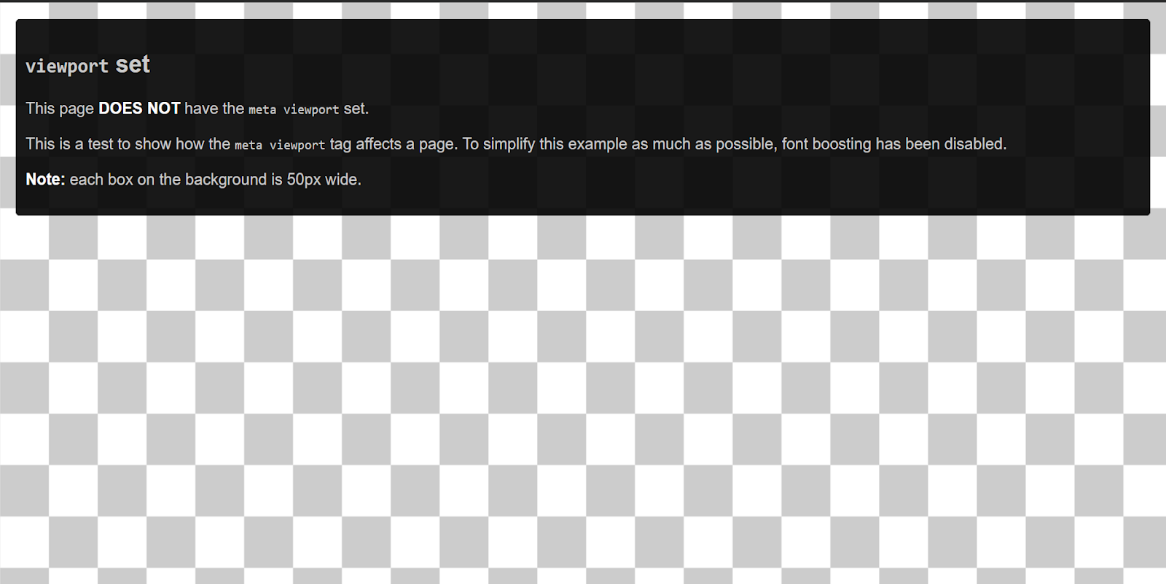
* **The reason**
* The use of mobile devices to surf the web continues to grow at an astronomical pace, and these devices are often constrained by display size and require a different approach to how content is laid out on the screen.
* **The definition and effect of responsive web design**
* Responsive web design, originally defined by [Ethan Marcotte in A List Apart](http://alistapart.com/article/responsive-web-design/), 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
* Screen sizes are always changing, so it's important that your site can adapt to any screen size, today or in the future. The modern responsive design considers all of these things to optimize the experience for everyone. Body, Head, and Title elements

**Chapter 1: Set the viewport**

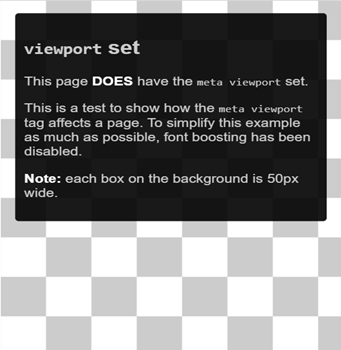
* **The definition**
* Pages optimized for a variety of devices must include a meta viewport tag in the head of the document. A meta viewport tag gives the browser instructions on how to control the page's dimensions and scaling.



* 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 is a representation of a single pixel, which may on a high-density screen consist of many physical pixels.
* **Example:**
* An example of how the page loads in a device without the viewport meta tag



* An example of how the page loads in a device with the viewport meta tag.

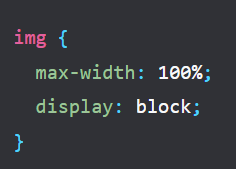


Some browsers keep the page's width constant when rotating to landscape mode, and zoom rather than reflow to fill the screen. Adding the value initial-scale = 1 instructs browsers to establish a 1:1 relationship between CSS pixels and device-independent pixels regardless of device orientation, and allows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

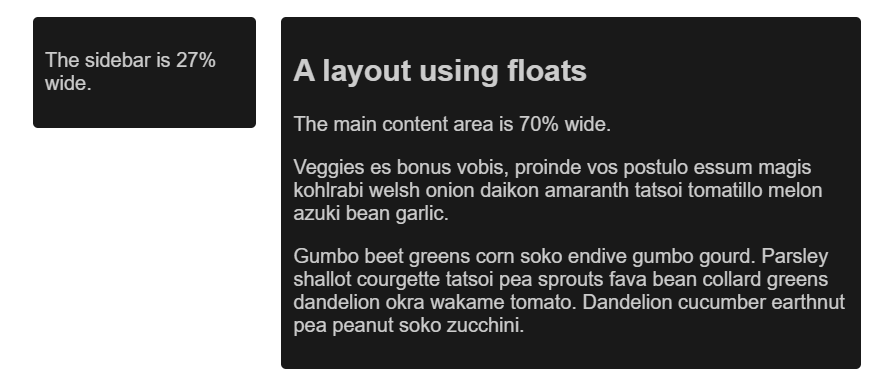
* The does not have a tag with width or initial-scale Lighthouse audit can help you automate the process of making sure that your HTML documents are using the viewport meta tag correctly.
* **NOTE:** To ensure that older browsers can properly parse the attributes, use a comma to separate the attributes.
* 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.

**Chapter 2: Size content to viewport**

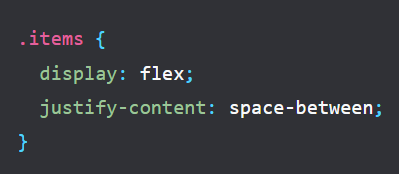
* **The reason**
* When developing a mobile site with a meta viewport tag, it's easy to accidentally create page content that doesn't quite fit within the specified viewport.
* For example, an image that is displayed at a width wider than the viewport can cause the viewport to scroll horizontally. You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



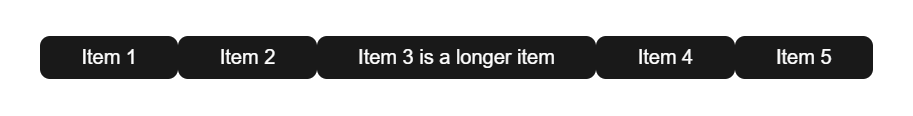
* When using max-width: 100% you are overriding the natural dimensions of the image. However, you should still use the width and height attributes on your <img> tag. This is because modern browsers will use this information to reserve space for the image before it loads in, this will help to avoid layout shifts as content loads.
* **Layout**
* Since screen dimensions and width in CSS pixels vary widely between devices, content should not rely on a particular viewport width to render well.
* 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.



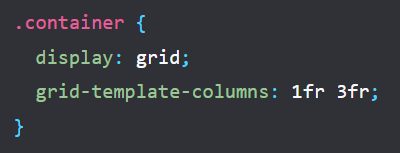
* Modern CSS layout techniques such as Flexbox, Grid Layout, and Multicol make the creation of these flexible grids much easier.
* **Flexbox**
* This layout method is ideal when you have a set of items of different sizes and you would like them to fit comfortably in a row or rows, with smaller items taking less space and larger ones getting more space.

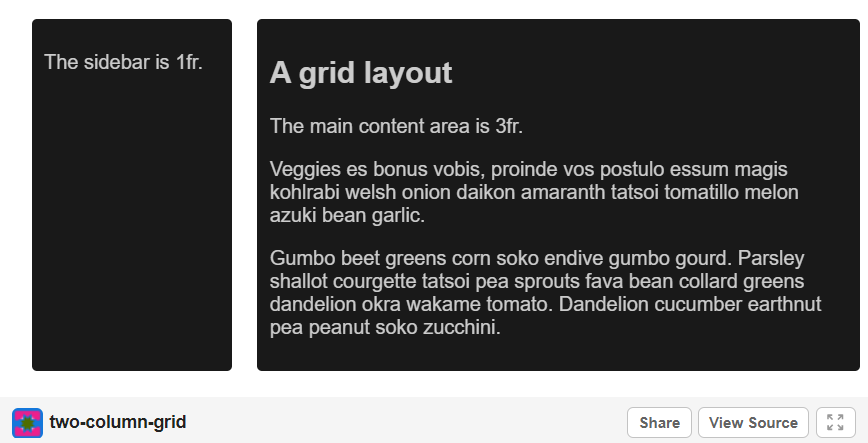


* In a responsive design, you can use Flexbox to display items as a single row, or wrapped onto multiple rows as the available space decreases.



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

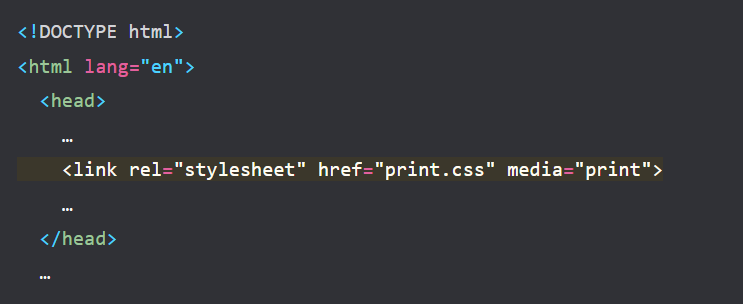




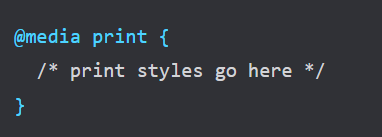
* The grid can also be used to create regular grid layouts, with as many items as will fit. The number of available tracks will be reduced as the screen size shrinks.
* **Multiple-column layout**
* For some types of layout, you can use a Multiple-column Layout (Multicol), which can create responsive numbers of columns with the column-width property

**Chapter 3: Use CSS media queries for responsiveness**

* **The reasons**
* 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.
* 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:



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



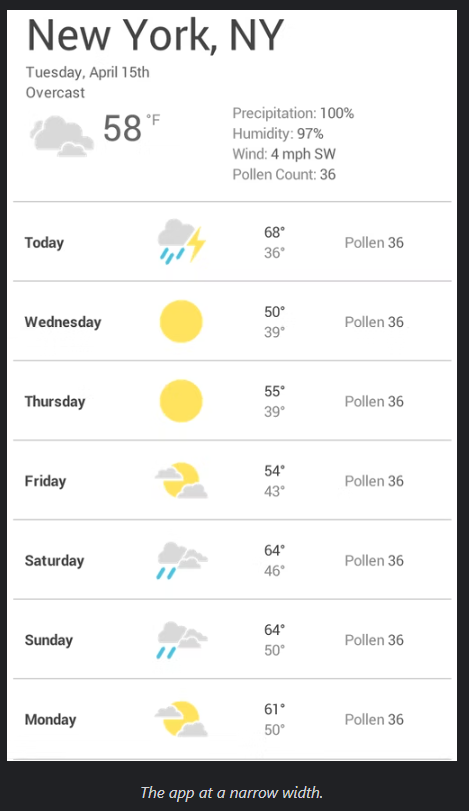
* **Note:**

It is also possible to include separate stylesheets in your main CSS file using the @import syntax, @import url(print.css) print; however, this use is not recommended for performance reasons

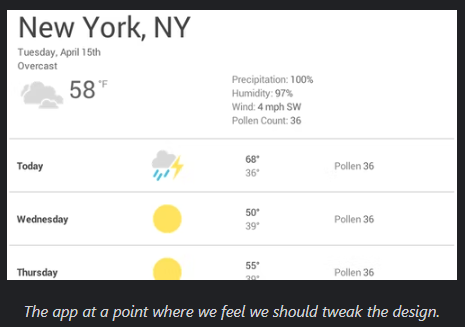
* **Media queries based on viewport size**
* Media queries enable us to create a responsive experience where specific styles are applied to small screens, large screens, and anywhere in between. The feature we are detecting here is therefore screen size, and we can test for the following things.
* width (min-width, max-width)
* height (min-height, max-height)
* orientation
* aspect-ratio
* **Media queries based on device capability**
* 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

**Chapter 4: How to choose breakpoints**

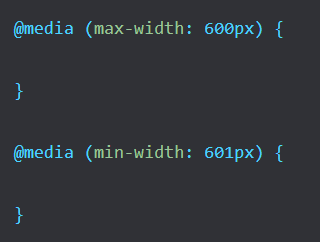
* **How to define**
* Don't define breakpoints based on device classes. Defining breakpoints based on specific devices, products, brand names, or operating systems that are in use today can result in a maintenance nightmare. Instead, the content itself should determine how the layout adjusts to its container
* **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.
* Example:
* The weather forecast
* The first step is to make the forecast look good on a small screen.



* 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.



* 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.



* 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
* **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.
* **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

**Chapter 5: View media query breakpoints in Chrome DevTools**

* **How to view media query breakpoints in Chrome DevTools**
* Chrome DevTools has a built-in feature that makes it easy to see how a page looks under different breakpoints.