**Relative Measurements**

Modern technology allows users to browse the Internet via multiple devices, such as desktop monitors, mobile phones, tablets, and more. Devices of different screen sizes, however, pose a problem for web developers: how can we ensure that a website is readable and visually appealing across all devices, regardless of screen size?

The answer: *responsive design*! Responsive design refers to the ability of a website to resize and reorganize its content based on:

1. The size of other content on the website.
2. The size of the screen the website is being viewed on.

In this lesson, we'll size HTML content *relative*to other content on a website.

You've probably noticed the unit of [pixels](https://en.wikipedia.org/wiki/Pixel), or px, used in websites. Pixels are used to size content to *exact* dimensions. For example, if you want a div to be exactly 500 pixels wide and 100 pixels tall, then the unit of px can be used. Pixels, however, are fixed, [hard coded](https://www.google.com/search?q=hard+coding) values. When a screen size changes (like switching from landscape to portrait view on a phone), elements sized with pixels can appear too small, overflow the screen, or become completely illegible.

With CSS, you can avoid hard coded measurements and use *relative measurements*instead. Relative measurements offer an advantage over hard coded measurements, as they allow for the proportions of a website to remain intact regardless of screen size or layout.

**Instructions**

The website in the browser to the right uses many relative measurements. Resize the page by increasing and decreasing the browser's width repeatedly. What do you notice about how the content on the page responds to the changes in width?

When you're done resizing the browser component, proceed to the next exercise.

**Em**

Incorporating relative sizing starts by using units other than pixels. One unit of measurement you can use in CSS to create relatively-sized content is the *em*, written as emin CSS.

Historically, the em represented the width of a capital letter M in the typeface and size being used. That is no longer the case.

Today, the em represents the size of the base font being used. For example, if the base font of a browser is 16 pixels (which is normally the default size of text in a browser), then 1 em is equal to 16 pixels. 2 ems would equal 32 pixels, and so on.

Let's take a look at two examples that show how em can be used in CSS.

.heading { font-size: 2em; }

In the example above, no base font has been specified, therefore the font size of the headingelement will be set relative to the default font size of the browser. Assuming the default font size is 16 pixels, then the font size of the heading element will be 32 pixels.

.splash-section { font-size: 18px; } .splash-section h1 { font-size: 1.5em; }

The example above shows how to use ems without relying on the default font size of the browser. Instead, a base font size (18px) is defined for all text within the splash-sectionelement. The second CSS rule will set the font size of all h1 elements inside of splash-sectionrelative to the base font of splash-section (18 pixels). The resulting font size of h1 elements will be 27 pixels.

**Instructions**

**1.**

In **style.css**, set the font size in #banner h1 ("Bana's Travel Blog") to 1.5em.

**2.**

Set the font size in .post h2 ("Saturday Market") to 1.75em.

**3.**

Set the font size in .post h3 to 1.25em.

**4.**

Set the font size in the footer ("© Bana's Travel Blog") to 0.75em.

**Percentages: Height & Width**

To size non-text HTML elements relative to their parent elements on the page you can use *percentages*.

Percentages are often used to size box-model values, like width and height, padding, border, and margins. They can also be used to set positioning properties (top, bottom, left, right).

To start, let's size the height and width of an element using percentages.

.main { height: 300px; width: 500px; } .main .subsection { height: 50%; width: 50%; }

In the example above, .main and .subsectioneach represent divs. The .subsection div is nested within the .main div. Note that the dimensions of the parent div (.main) have been set to a height of 300 pixels and a width of 500 pixels.

When percentages are used, elements are sized relative to the dimensions of their parent element (also known as a container). Therefore, the dimensions of the .subsection div will be 150 pixels tall and 250 pixels wide. Be careful, a child element's dimensions may be set erroneously if the dimensions of its parent element aren't set first.

**Note:** Because the box model includes padding, borders, and margins, setting an element's width to 100% may cause content to overflow its parent container. While tempting, 100%should only be used when content will not have padding, border, or margin.

**Instructions**

**1.**

Currently, the blog takes up the full width of the body. Let's modify this so that it doesn't extend to fill the full width.

In **style.css**, set the width in #blog to 86%.

This will responsively set the entire blog's container to 86% of the full width of the body.

**2.**

Great! Resize the browser's width again. Notice that the blog's text becomes illegible at smaller widths.

To fix this, set the width in #blog .postto 52%.

This will ensure that the text fills only 52% of its container's (#blog) width. Resize the browser now and notice how the text remains legible.

**3.**

Now set the width in .post .image-container to 100%. This will make sure the image's container is always the full width of the blog post (.post).

Stuck? Get a hint

**4.**

Scroll to the bottom of the web page. Notice that there are two images. We'd like to display these images next to each other on the page, with equal width.

Set the width in .images .image-container to 50%. This will give each image in .images equal width.

Don't worry if the images still look distorted at the moment. You'll improve their appearance in a later exercise.

**Percentages: Padding & Margin**

Percentages can also be used to set the padding and margin of elements.

When height and width are set using percentages, you learned that the dimensions of child elements are calculated based on the dimensions of the parent element.

When percentages are used to set padding and margin, however, they are calculated based only on the *width* of the parent element.

For example, when a property like margin-leftis set using a percentage (say 50%), the element will be moved halfway to the right in the parent container (as opposed to the child element receiving a margin half of its parent's margin).

Vertical padding and margin are also calculated based on the width of the parent. Why? Consider the following scenario:

1. A container div is defined, but its height is not set (meaning it's flat).
2. The container then has a child element added within. The child element *does* have a set height. This causes the height of its parent container to stretch to that height.
3. The child element requires a change, and its height is modified. This causes the parent container's height to also stretch to the new height. This cycle occurs endlessly whenever the child element's height is changed!

In the scenario above, an unset height (the parent's) results in a constantly changing height due to changes to the child element. This is why vertical padding and margin are based on the width of the parent, and not the height.

**Note:** When using relative sizing, ems and rems should be used to size text and dimensions on the page related to text size (i.e. padding around text). This creates a consistent layout based on text size. Otherwise, percentages should be used.

**Instructions**

**1.**

Let's size the height of the banner relative to the root element's font size.

In **style.css**, set the height in #banner to 46rem.

**Note:** The root element's font size is 16 pixels, meaning that 46rem will result in a height of 736 pixels.

**2.**

Set the top margin in #blog .post to 12.5%.

**3.**

Set the bottom margin in #blog .post to 7.5%.

**4.**

Set the bottom margin in .images to 20%.

**Width: Minimum & Maximum**

Although relative measurements provide consistent layouts across devices of different screen sizes, elements on a website can lose their integrity when they become too small or large. You can limit how wide an element becomes with the following properties:

1. min-width — ensures a minimum width for an element.
2. max-width — ensures a maximum width for an element.

p { min-width: 300px; max-width: 600px; }

In the example above, when the browser is resized, the width of paragraph elements will not fall below 300 pixels, nor will their width exceed 600 pixels.

When a browser window is narrowed or widened, text can become either very compressed or very spread out, making it difficult to read. These two properties ensure that content is legible by limiting the minimum and maximum widths.

**Note**: The unit of pixels is used to ensure hard limits on the dimensions of the element(s).

**Instructions**

**1.**

Resize the browser to the right. Notice that the text on the web page can become difficult to read. Let's limit the text's width to keep it legible.

In **style.css**, create a CSS rule that sets the minimum width of all paragraphs to 200 pixels and run your code.

Now, resize your browser (make it narrower) and notice that the text no longer compresses as much as it did before.

**Height: Minimum & Maximum**

You can also limit the minimum and maximum *height* of an element.

1. min-height — ensures a minimum height for an element's box.
2. max-height — ensures a maximum height for an element's box.

p { min-height: 150px; max-height: 300px; }

In the example above, the height of all paragraphs will not shrink below 150 pixels and the height will not exceed 300 pixels.

What will happen to the contents of an element if the max-height property is set too low for that element? It's possible that content will overflow outside of the element, resulting in content that is not legible.

**Instructions**

**1.**

Again, resize your browser (stretch it out). Notice that the paragraph's text can become overly spread out (i.e. a small height). Let's limit the height to keep the text legible.

In **style.css**, set the minimum height of all paragraphs to 200 pixels.

Resize your browser once more and notice that the text no longer spreads out as much as it did before.

**Scaling Images and Videos**

Many websites contain a variety of different media, like images and videos. When a website contains such media, it's important to make sure that it is scaled proportionally so that users can correctly view it.

.container { width: 50%; height: 200px; overflow: hidden; } .container img { max-width: 100%; height: auto; display: block; }

In the example above, .container represents a container div. It is set to a width of 50% (half of the browser's width, in this example) and a height of 200 pixels. Setting overflow to hidden ensures that any content with dimensions larger than the container will be hidden from view.

The second CSS rule ensures that images scale with the width of the container. The heightproperty is set to auto, meaning an image's height will *automatically* scale proportionally with the width. Finally, the last line will display images as block level elements (rather than inline-block, their default state). This will prevent images from attempting to align with other content on the page (like text), which can add unintended margin to the images.

It's worth memorizing the entire example above. It represents a *very common* design pattern used to scale images and videos proportionally.

**Note:** The example above scales the width of an image (or video) to the width of a container. If the image is larger than the container, the vertical portion of the image will overflow and will not display. To swap this behavior, you can set max-height to 100% and width to auto(essentially swapping the values). This will scale the *height* of the image with the height of the container instead. If the image is larger than the container, the horizontal portion of the image will overflow and not display.

**Instructions**

**1.**

Take a look at the images on the web page. Notice that they currently display incorrectly (too large). Let's fix that.

First, in **style.css**, set the overflowproperty in .image-container to hidden. Run your code.

Take a look at the images once more. At this point, the images partially display. In reality, what we've done is constrain them to the dimensions of their container (.image-container). Any part of the image that overflows out of the container will be hidden from view. This will set us up to scale them proportionally.

**2.**

Resize the width of the browser back and forth. Notice that the images expand and contract to show more (or less) of the image. Instead, let's display the full image at all times.

In **style.css**, set the maximum width in .image-container img to 100%. This will ensure the full image is always displayed.

**3.**

Great work! Take a look at the images on the web page again — they have been greatly improved!

Next, we'll want to make sure the images automatically remain in proportion when the browser is resized.

In .image-container img, set the height to auto.

**4.**

Finally, within the same CSS rule, set the display to block. This will instruct the images to behave as block-level elements and facilitate scaling (as opposed to their default inline behavior).

**Scaling Background Images**

Background images of HTML elements can also be scaled responsively using CSS properties.

body { background-image: url('#'); background-repeat: no-repeat; background-position: center; background-size: cover; }

In the example above, the first CSS declaration sets the background image (# is a placeholder for an image URL in this example). The second declaration instructs the CSS compiler to not repeat the image (by default, images will repeat). The third declaration centers the image within the element.

The final declaration, however, is the focus of the example above. It's what scales the background image. The image will *cover* the entire background of the element, all while keeping the image in proportion. If the dimensions of the image exceed the dimensions of the container then only a portion of the image will display.

**Instructions**

**1.**

In **style.css**, set the background-sizeproperty in #banner to cover.

**eview: Relative Measurements**

Great work! You learned how to size elements on a website relative to other elements on the page.

Let's review what you learned:

* Content on a website can be sized relative to other elements on the page using *relative measurements*.
* The unit of em sizes font relative to the font size of a parent element.
* The unit of rem sizes font relative to the font size of a root element. That root element is the <html> element.
* Percentages are commonly used to size box-model features, like the width, height, borders, padding, or margin of an element.
* When percentages are used to size width and height, child elements will be sized relative to the dimensions of their parent (remember that parent dimensions must first be set).
* Percentages can be used to set padding and margin. Horizontal and vertical padding and margin are set relative to the width of a parent element.
* The minimum and maximum width of elements can be set using min-width and max-width.
* The minimum and maximum height of elements can be set using min-height and max-height.
* When the height of an image or video is set, then its width can be set to auto so that the media scales proportionally. Reversing these two properties and values will also achieve the same result.
* A background image of an HTML element will scale proportionally when its background-size property is set to cover.

Relative units of measurement are a first step towards incorporating responsive design in a website. When combined with more advanced responsive techniques, you can create a seamless user experience regardless of a device's screen size.

**Instructions**

Take some time to experiment with your new knowledge of relative measurements in **style.css**. When you're done, proceed to the next unit.