**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 em in 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 heading element 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-section element. The second CSS rule will set the font size of all h1 elements inside of splash-section relative 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 (“&copy; Bana’s Travel Blog”) to 0.75em.

/\* Universal Styles \*/

body {

  background-color: white;

}

.image-container {

}

.image-container img {

}

/\* Banner Section \*/

#banner {

  background-image: url('camel-background.png');

  background-position: center;

  background-repeat: no-repeat;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

#banner h1 {

  font-size: 1.5em;

  font-family: 'Roboto', sans-serif;

  font-weight: 300;

  color: white;

}

/\* Blog Post \*/

#blog {

  margin: auto;

  display: flex;

  flex-direction: column;

  align-items: center;

}

#blog .post {

  display: flex;

  flex-direction: column;

  align-items: center;

  font-family: 'Merriweather', serif;

  font-weight: 300;

  font-size: 1rem;

  text-align: center;

  line-height: 1.8;

  color: #444444;

}

.post h2 {

  font-size: 1.75em;

}

.post h3 {

  font-size: 1.25em;

  color: #999999;

}

.post .opening-line {

  margin-top: 4.1875rem;

  margin-bottom: 1.5rem;

  color: black;

  font-weight: bold;

}

.post .image-container {

}

/\* Blog Images \*/

.images {

}

.images .image-container {

  display: inline-block;

}

/\* Footer \*/

footer {

  font-size: 0.75em;

  padding: 4rem 0;

  border-top: 1px solid #999999;

  font-family: 'Roboto', sans-serif;

  color: #999999;

  text-align: center;

}

# Rem

The second relative unit of measurement in CSS is the rem, coded as rem.

Rem stands for root em. It acts similar to em, but instead of checking parent elements to size font, it checks the root element. The root element is the <html> tag.

Most browsers set the font size of <html> to 16 pixels, so by default rem measurements will be compared to that value. To set a different font size for the root element, you can add a CSS rule.

html {

font-size: 20px;

}

h1 {

font-size: 2rem;

}

In the example above, the font size of the root element, <html>, is set to 20 pixels. All subsequent rem measurements will now be compared to that value and the size of h1 elements in the example will be 40 pixels.

One advantage of using rems is that all elements are compared to the same font size value, making it easy to predict how large or small font will appear. If you are interested in sizing elements consistently across an entire website, the rem measurement is the best unit for the job. If you’re interested in sizing elements in comparison to other elements nearby, then the em unit would be better suited for the job.

**Instructions**

**1.**

In **style.css**, add a new rule on line 3 that sets the font size of the root element to 16 pixels.

Hint

html {

font-size: 16px;

}

**2.**

Let’s update the font sizes you set in the previous exercise to use rem instead of em.

First, change the font size in #banner h1 to 3.75rem.

**3.**

Set the font size in .post h2 to 1.875rem.

**4.**

Set the font size in .post h3 to 1.125rem.

**5.**

Set the font size of the footer to 1.125rem.

/\* Universal Styles \*/

html {

  font-size: 16px;

}

body {

  background-color: white;

}

.image-container {

}

.image-container img {

}

/\* Banner Section \*/

#banner {

  background-image: url('camel-background.png');

  background-position: center;

  background-repeat: no-repeat;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

#banner h1 {

  font-size: 3.75rem;

  font-family: 'Roboto', sans-serif;

  font-weight: 300;

  color: white;

}

/\* Blog Post \*/

#blog {

  margin: auto;

  display: flex;

  flex-direction: column;

  align-items: center;

}

#blog .post {

  display: flex;

  flex-direction: column;

  align-items: center;

  font-family: 'Merriweather', serif;

  font-weight: 300;

  font-size: 1rem;

  text-align: center;

  line-height: 1.8;

  color: #444444;

}

.post h2 {

  font-size: 1.875em;

}

.post h3 {

  font-size: 1.125em;

  color: #999999;

}

.post .opening-line {

  margin-top: 4.1875rem;

  margin-bottom: 1.5rem;

  color: black;

  font-weight: bold;

}

.post .image-container {

}

/\* Blog Images \*/

.images {

}

.images .image-container {

  display: inline-block;

}

/\* Footer \*/

footer {

  font-size: 1.125em;

  padding: 4rem 0;

  border-top: 1px solid #999999;

  font-family: 'Roboto', sans-serif;

  color: #999999;

  text-align: center;

}

/\* Universal Styles \*/

html {

  font-size: 16px;

}

body {

  background-color: white;

}

.image-container {

}

.image-container img {

}

/\* Banner Section \*/

#banner {

  background-image: url('camel-background.png');

  background-position: center;

  background-repeat: no-repeat;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

#banner h1 {

  font-size: 3.75rem;

  font-family: 'Roboto', sans-serif;

  font-weight: 300;

  color: white;

}

/\* Blog Post \*/

#blog {

  margin: auto;

  display: flex;

  flex-direction: column;

  align-items: center;

}

#blog .post {

  display: flex;

  flex-direction: column;

  align-items: center;

  font-family: 'Merriweather', serif;

  font-weight: 300;

  font-size: 1rem;

  text-align: center;

  line-height: 1.8;

  color: #444444;

}

.post h2 {

  font-size: 1.875em;

}

.post h3 {

  font-size: 1.125em;

  color: #999999;

}

.post .opening-line {

  margin-top: 4.1875rem;

  margin-bottom: 1.5rem;

  color: black;

  font-weight: bold;

}

.post .image-container {

}

/\* Blog Images \*/

.images {

}

.images .image-container {

  display: inline-block;

}

/\* Footer \*/

footer {

  font-size: 1.125em;

  padding: 4rem 0;

  border-top: 1px solid #999999;

  font-family: 'Roboto', sans-serif;

  color: #999999;

  text-align: center;

}

# 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 .subsection each 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 .post to 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).

Hint

There are two CSS rules that make use of the .image-container selector. Be sure to add your code in the correct selector, like so:

.post .image-container {

width: 100%;

}

**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.

/\* Universal Styles \*/

html {

  font-size: 16px;

}

body {

  background-color: white;

}

.image-container {

}

.image-container img {

}

/\* Banner Section \*/

#banner {

  background-image: url('camel-background.png');

  background-position: center;

  background-repeat: no-repeat;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

#banner h1 {

  font-size: 3.75rem;

  font-family: 'Roboto', sans-serif;

  font-weight: 300;

  color: white;

}

/\* Blog Post \*/

#blog {

  width: 86%;

  margin: auto;

  display: flex;

  flex-direction: column;

  align-items: center;

}

#blog .post {

  width: 52%;

  display: flex;

  flex-direction: column;

  align-items: center;

  font-family: 'Merriweather', serif;

  font-weight: 300;

  font-size: 1rem;

  text-align: center;

  line-height: 1.8;

  color: #444444;

}

.post h2 {

  font-size: 1.875em;

}

.post h3 {

  font-size: 1.125em;

  color: #999999;

}

.post .opening-line {

  margin-top: 4.1875rem;

  margin-bottom: 1.5rem;

  color: black;

  font-weight: bold;

}

.post .image-container {

  width: 100%;

}

/\* Blog Images \*/

.images {

}

.images .image-container {

  display: inline-block;

  width: 50%;

}

/\* Footer \*/

footer {

  font-size: 1.125em;

  padding: 4rem 0;

  border-top: 1px solid #999999;

  font-family: 'Roboto', sans-serif;

  color: #999999;

  text-align: center;

}

**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-left is 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**, for the #banner ruleset, add a height property and assign it 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%.

/\* Universal Styles \*/

html {

  font-size: 16px;

}

body {

  background-color: white;

}

.image-container {

}

.image-container img {

}

/\* Banner Section \*/

#banner {

  height: 46rem;

  background-image: url('camel-background.png');

  background-position: center;

  background-repeat: no-repeat;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

#banner h1 {

  font-size: 3.75rem;

  font-family: 'Roboto', sans-serif;

  font-weight: 300;

  color: white;

}

/\* Blog Post \*/

#blog {

  width: 86%;

  margin: auto;

  display: flex;

  flex-direction: column;

  align-items: center;

}

#blog .post {

  width: 52%;

  margin-top: 12.5%;

  margin-bottom: 7.5%;

  display: flex;

  flex-direction: column;

  align-items: center;

  font-family: 'Merriweather', serif;

  font-weight: 300;

  font-size: 1rem;

  text-align: center;

  line-height: 1.8;

  color: #444444;

}

.post h2 {

  font-size: 1.875em;

}

.post h3 {

  font-size: 1.125em;

  color: #999999;

}

.post .opening-line {

  margin-top: 4.1875rem;

  margin-bottom: 1.5rem;

  color: black;

  font-weight: bold;

}

.post .image-container {

  width: 100%;

}

/\* Blog Images \*/

.images {

  margin-bottom: 20%;

}

.images .image-container {

  display: inline-block;

  width: 50%;

}

/\* Footer \*/

footer {

  font-size: 1.125em;

  padding: 4rem 0;

  border-top: 1px solid #999999;

  font-family: 'Roboto', sans-serif;

  color: #999999;

  text-align: center;

}

# 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.

/\* Universal Styles \*/

p{

  min-width: 200px;

}

html {

  font-size: 16px;

}

body {

  background-color: white;

}

.image-container {

}

.image-container img {

}

/\* Banner Section \*/

#banner {

  height: 46rem;

  background-image: url('camel-background.png');

  background-position: center;

  background-repeat: no-repeat;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

#banner h1 {

  font-size: 3.75rem;

  font-family: 'Roboto', sans-serif;

  font-weight: 300;

  color: white;

}

/\* Blog Post \*/

#blog {

  width: 86%;

  margin: auto;

  display: flex;

  flex-direction: column;

  align-items: center;

}

#blog .post {

  width: 52%;

  margin-top: 12.5%;

  margin-bottom: 7.5%;

  display: flex;

  flex-direction: column;

  align-items: center;

  font-family: 'Merriweather', serif;

  font-weight: 300;

  font-size: 1rem;

  text-align: center;

  line-height: 1.8;

  color: #444444;

}

.post h2 {

  font-size: 1.875em;

}

.post h3 {

  font-size: 1.125em;

  color: #999999;

}

.post .opening-line {

  margin-top: 4.1875rem;

  margin-bottom: 1.5rem;

  color: black;

  font-weight: bold;

}

.post .image-container {

  width: 100%;

}

/\* Blog Images \*/

.images {

  margin-bottom: 20%;

}

.images .image-container {

  display: inline-block;

  width: 50%;

}

/\* Footer \*/

footer {

  font-size: 1.125em;

  padding: 4rem 0;

  border-top: 1px solid #999999;

  font-family: 'Roboto', sans-serif;

  color: #999999;

  text-align: center;

}

# 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.

{

  min-width: 200px;

  min-height: 200px;

}

**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 height property 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 overflow property 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).

p{

  min-width: 200px;

  min-height: 200px;

}

html {

  font-size: 16px;

}

body {

  background-color: white;

}

.image-container {

  overflow: hidden;

}

.image-container img {

  max-width: 100%;

  height: auto;

  display: block;

}

/\* Banner Section \*/

#banner {

  height: 46rem;

  background-image: url('camel-background.png');

  background-position: center;

  background-repeat: no-repeat;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

#banner h1 {

  font-size: 3.75rem;

  font-family: 'Roboto', sans-serif;

  font-weight: 300;

  color: white;

}

/\* Blog Post \*/

#blog {

  width: 86%;

  margin: auto;

  display: flex;

  flex-direction: column;

  align-items: center;

}

#blog .post {

  width: 52%;

  margin-top: 12.5%;

  margin-bottom: 7.5%;

  display: flex;

  flex-direction: column;

  align-items: center;

  font-family: 'Merriweather', serif;

  font-weight: 300;

  font-size: 1rem;

  text-align: center;

  line-height: 1.8;

  color: #444444;

}

.post h2 {

  font-size: 1.875em;

}

.post h3 {

  font-size: 1.125em;

  color: #999999;

}

.post .opening-line {

  margin-top: 4.1875rem;

  margin-bottom: 1.5rem;

  color: black;

  font-weight: bold;

}

.post .image-container {

  width: 100%;

}

/\* Blog Images \*/

.images {

  margin-bottom: 20%;

}

.images .image-container {

  display: inline-block;

  width: 50%;

}

/\* Footer \*/

footer {

  font-size: 1.125em;

  padding: 4rem 0;

  border-top: 1px solid #999999;

  font-family: 'Roboto', sans-serif;

  color: #999999;

  text-align: center;

}

**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-size property in #banner to cover.

#banner {

  height: 46rem;

  background-image: url('camel-background.png');

  background-position: center;

  background-size: cover;

  background-repeat: no-repeat;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

# Review: 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, 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.

# Responsive Web Design

When someone visits a website, it’s possible they are viewing it on a phone, tablet, computer, or even a TV monitor. Because screen sizes can vary greatly across different devices, it’s important for websites to resize and reorganize their content to best fit screens of all sizes.

When a website doesn’t respond to different screen sizes, the website may look odd or become indecipherable on certain devices. This usually occurs on smaller screens, like phones. When a website responds to the size of the screen it’s viewed on, it’s called a responsive website.

Because websites can be displayed on thousands of different screen sizes, they must be able to respond to a change in screen size and adapt the content so that users can access it.

**Instructions**

Let’s take a look at a website that does not respond to different screen sizes.

1. First, visit [this site](https://s3.amazonaws.com/codecademy-content/courses/freelance-1/unit-5/globe-book-store/index.html).
2. Then resize the width of your browser to simulate a smaller screen size. Note how the content on the web page does not shrink as you resize the browser’s width.

In contrast, let’s take a look at a responsive website.

1. First, visit [this site](https://s3.amazonaws.com/codecademy-content/courses/freelance-1/unit-5/globe-book-store/index-responsive.html).
2. Again, resize the width of your browser and note how the content on the web page resizes and reorganizes itself.

# Media Queries

CSS uses media queries to adapt a website’s content to different screen sizes. With media queries, CSS can detect the size of the current screen and apply different CSS styles depending on the width of the screen.

@media only screen and (max-width: 480px) {

body {

font-size: 12px;

}

}

The example above demonstrates how a media query is applied. The media query defines a rule for screens smaller than 480 pixels (approximately the width of many smartphones in [landscape](https://en.wikipedia.org/wiki/Page_orientation) orientation).

Let’s break this example down into its parts:

1. @media — This keyword begins a media query rule and instructs the CSS compiler on how to parse the rest of the rule.
2. only screen — Indicates what types of devices should use this rule. In early attempts to target different devices, CSS incorporated different media types (screen, print, handheld). The rationale was that by knowing the media type, the proper CSS rules could be applied. However, “handheld” and “screen” devices began to occupy a much wider range of sizes and having only one CSS rule per media device was not sufficient. screen is the media type always used for displaying content, no matter the type of device. The only keyword is added to indicate that this rule only applies to one media type (screen).
3. and (max-width : 480px) — This part of the rule is called a media feature, and instructs the CSS compiler to apply the CSS styles to devices with a width of 480 pixels or smaller. Media features are the conditions that must be met in order to render the CSS within a media query.
4. CSS rules are nested inside of the media query’s curly braces. The rules will be applied when the media query is met. In the example above, the text in the body element is set to a font-size of 12px when the user’s screen is less than 480px.

**Instructions**

**1.**

At the bottom of **style.css**, write a media query for a max-width of 480px.

This will allow us to shrink the width of the .page-title element on smaller screens.

**2.**

When the screen is less than 480px wide, give the .page-title class a width of 270px.

This will make the .page-title element appear more clearly on small screens. Test your code by resizing the browser.

media only screen and (max-width: 480px) {

  .page-title {

    width: 270px;

  }

}

**Range**

Specific screen sizes can be targeted by setting multiple width and height media features. min-width and min-height are used to set the minimum width and minimum height, respectively. Conversely, max-width and max-height set the maximum width and maximum height, respectively.

By using multiple widths and heights, a range can be set for a media query.

@media only screen and (min-width: 320px) and (max-width: 480px) {

/\* ruleset for 320px - 480px \*/

}

The example above would apply its CSS rules only when the screen size is between 320 pixels and 480 pixels. Notice the use of a second and keyword after the min-width media feature. This allows us to chain two requirements together.

The example above can be written using two separate rules as well:

@media only screen and (min-width: 320px) {

/\* ruleset for >= 320px \*/

}

@media only screen and (min-width: 480px) {

/\* ruleset for >= 480px \*/

}

The first media query in the example above will apply CSS rules when the size of the screen meets or exceeds 320 pixels. The second media query will then apply CSS rules when the size of the screen meets or exceeds 480 pixels, meaning that it can override CSS rules present in the first media query or apply additional CSS rules that are not already present in the first.

Both examples above are valid, and it is likely that you will see both patterns used when reading another developer’s code.

**Instructions**

**1.**

Let’s make the gallery images appear larger when the screen size is small to medium size.

Write one @media query for screen sizes with a range between 320px and 480px. Use min-width and max-width to define the range.

**2.**

Inside the media query, select the thumbnails within the gallery with .gallery-item .thumbnail and give them a width of 95%.

You should notice that the gallery images appear wider when the screen size is between 320 and 480 pixels wide.

@media only screen and (min-width: 320px) and (max-width: 480px) {

  .gallery-item .thumbnail{

      width: 95%;

  }

# Dots Per Inch (DPI)

Another media feature we can target is screen resolution. Many times we will want to supply higher quality media (images, video, etc.) only to users with screens that can support high resolution media. Targeting screen resolution also helps users avoid downloading high resolution (large file size) images that their screen may not be able to properly display.

To target by resolution, we can use the min-resolution and max-resolution media features. These media features accept a resolution value in either dots per inch (dpi) or dots per centimeter (dpc). Learn more about resolution measurements [here](https://en.wikipedia.org/wiki/Dots_per_inch).

@media only screen and (min-resolution: 300dpi) {

/\* CSS for high resolution screens \*/

}

The media query in the example above targets high resolution screens by making sure the screen resolution is at least 300 dots per inch. If the screen resolution query is met, then we can use CSS to display high resolution images and other media.

**Instructions**

**1.**

Write a media query to make the logo higher quality if the visitor is looking at the Amazing Space website on a high resolution display.

A high resolution display may have a min-resolution of 150dpi.

**2.**

Inside of the media query, add this CSS property to the .logo class:

background-image: url("../img/spaceship@2x.png");

This code will exchange the existing logo with a higher resolution logo. To make the difference obvious, the higher resolution spaceship logo has blue wings.

If you are accessing Codecademy on a screen with a resolution greater than 150dpi, you will observe the spaceship change. If not, you can re-write the min-resolution media feature to a lower value to observe the change.

@media only screen and (min-resolution: 150dpi) {

    /\* CSS for high resolution screens \*/

    .logo{

      background-image: url("../img/spaceship@2x.png");

    }

}

**And Operator**

In previous exercises, we chained multiple media features of the same type in one media query by using the and operator. It allowed us to create a range by using min-width and max-width in the same media query.

The and operator can be used to require multiple media features. Therefore, we can use the and operator to require both a max-width of 480px *and* to have a min-resolution of 300dpi.

For example:

@media only screen and (max-width: 480px) and (min-resolution: 300dpi) {

/\* CSS ruleset \*/

}

By placing the and operator between the two media features, the browser will require both media features to be true before it renders the CSS within the media query. The and operator can be used to chain as many media features as necessary.

**Instructions**

**1.**

The website’s text needs to be larger for users who have small, low resolution screens.

Write a media query that applies when the max-resolution is 150dpi and the screen has a max-width of 480px.

**2.**

Inside the media query, make the font-size of the .page-description element 20px.

@media only screen and (max-resolution: 150dpi) and (max-width: 480px) {

    /\* CSS ruleset \*/

    .page-description{

      font-size: 20px;

    }

}

# Comma Separated List

If only one of multiple media features in a media query must be met, media features can be separated in a comma separated list.

For example, if we needed to apply a style when only one of the below is true:

* The screen is more than 480 pixels wide
* The screen is in landscape mode

We could write:

@media only screen and (min-width: 480px), (orientation: landscape) {

/\* CSS ruleset \*/

}

In the example above, we used a comma (,) to separate multiple rules. The example above requires only one of the media features to be true for its CSS to apply.

Note that the second media feature is orientation. The orientation media feature detects if the page has more width than height. If a page is wider, it’s considered landscape, and if a page is taller, it’s considered portrait.

**Instructions**

**1.**

Navigate to the first media query where you targeted screens with a min-width of 320px and a max-width of 480px.

Let’s also make the logo and text appear vertical if the screen is in a portrait orientation.

Add another media feature to the rule, using a comma (,) to separate rules. The second media feature should check if the screen’s orientation is portrait.

@media only screen and (min-width: 320px) and (max-width: 480px) , (orientation: portrait) {

  .gallery-item .thumbnail {

    width: 95%;

  }

}

# Breakpoints

We know how to use media queries to apply CSS rules based on screen size and resolution, but how do we determine what queries to set?

The points at which media queries are set are called breakpoints. Breakpoints are the screen sizes at which your web page does not appear properly. For example, if we want to target tablets that are in landscape orientation, we can create the following breakpoint:

@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: landscape) {

/\* CSS ruleset \*/

}

The example above creates a screen size range the size of a tablet in landscape mode and also identifies the orientation.

However, setting breakpoints for every device imaginable would be incredibly difficult because there are many devices of differing shapes and sizes. In addition, new devices are released with new screen sizes every year.

Rather than set breakpoints based on specific devices, the best practice is to resize your browser to view where the website naturally breaks based on its content. The dimensions at which the layout breaks or looks odd become your media query breakpoints. Within those breakpoints, we can adjust the CSS to make the page resize and reorganize.

By observing the dimensions at which a website naturally breaks, you can set media query breakpoints that create the best possible user experience on a project by project basis, rather than forcing every project to fit a certain screen size. Different projects have different needs, and creating a responsive design should be no different.

Check out [this](https://s3.amazonaws.com/codecademy-content/courses/freelance-1/unit-5/screen-sizes.png) list of breakpoints by device widths. Use it as a reference of screen widths to test your website to make certain it looks great across a variety of devices.

**Instructions**

**1.**

The last breakpoint we’d like to account for is a tablet in landscape orientation. The Amazing Space website should change its format to show the gallery pictures on the right, while having the logo and the description on the left.

Write a media query that meets the following requirements:

* The screen has a min-width of 768px
* The screen has a max-width of 1024px
* The screen has an orientation of landscape

**2.**

Inside of the media query, include this CSS:

.page-title, .page-description {

float: left;

width: 20em;

}

.page-description {

text-align: left;

}

This CSS will make the page title and description float to the left of the gallery images. Resize the browser to observe these changes at various screen widths.

@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: landscape){

  .page-title, .page-description {

  float: left;

    width: 20em;

}

.page-description {

    text-align: left;

}

}

# Review: Media Queries

Incredible work! You learned how to change the way a website appears on different screens with media queries and breakpoints

Throughout this lesson, you learned:

* When a website responds to the size of the screen it’s viewed on, it’s called a responsive website.
* You can write media queries to help with different screen sizes.
* Media queries require media features. Media features are the conditions that must be met to render the CSS within a media query.
* Media features can detect many aspects of a user’s browser, including the screen’s width, height, resolution, orientation, and more.
* The and operator requires multiple media features to be true at once.
* A comma separated list of media features only requires one media feature to be true for the code within to be applied.
* The best practice for identifying where media queries should be set is by resizing the browser to determine where the content naturally breaks. Natural breakpoints are found by resizing the browser.

With your knowledge of media queries and CSS, you can make websites that look great on any device, from a small phone to a huge television. By making your websites responsive, you’ll make it possible for any of your users to have a great experience.