girls who code

AT HOME

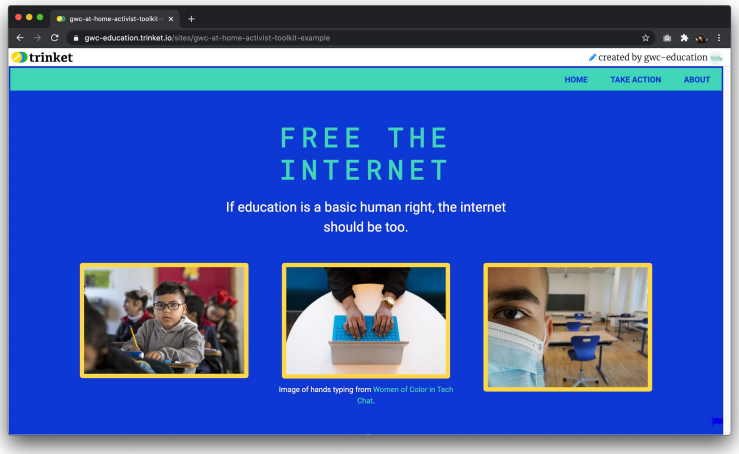
Activist Toolkit Part 5

Building: Make it Responsive

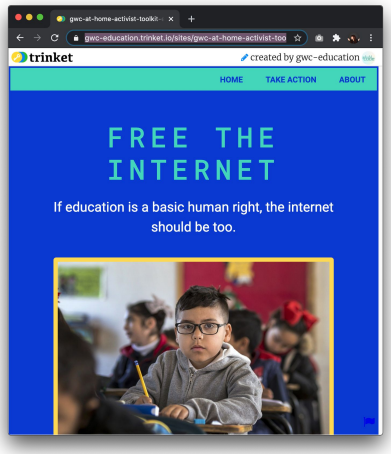
Updated: May 2025

Activity Overview

Your website is looking great with the CSS you added in the last activity! For this part of the series, we will explore how to make sure your website looks great on different screen sizes like laptop, tablet, and mobile devices. This is called responsive design.



Screen size on a laptop



Screen size on a mobile device

Materials

- Computer
- Trinket or the text editor of your choice
- Code you wrote during Part 4
- Final Project Example Code

*Note: If you **did not** complete Part 4, you can access Parts 1, 2, and 3 here. If you have some experience with HTML, you can remix and use the Part 4 Example Code that stops at the end of Part 4.*

PART 1 PLANNING	PART 2 PLANNING	PART 3 BUILDING	PART 4 BUILDING	PART 5 BUILDING
Research & Feedback	**Wireframe & Mockup**	**Intro to HTML**	**Intro to CSS**	**Make it Responsive with CSS**
Identify and research a cause.	Layout the skeleton of your website, then add in your design elements.	Build the framework of your site using HTML.	Style the HTML elements on your site using CSS.	Add styling that adapts your site to different screen sizes.

GWC Tech Spotlight: Gitanjali Rao

Image Source: Wikipedia

At age 10, Gitanjali Rao was deeply moved by the Flint water crisis. Concerned that children her age couldn't access clean water, she invented Tethys, a device using carbon nanotubes to detect lead in water. The device was affordable, easy to use, and won her the Discovery Education 3M Young Scientist Challenge at 11 years old. Gitanjali's passion for solving real problems continued with Epione, a tool for early detection of opioid addiction, and Kindly, an AI app that prevents cyberbullying. Her achievements earned her recognition as TIME Magazine's first-ever "Kid of the Year" in 2020.

Currently a student at MIT, Gitanjali is also committed to inspiring other young innovators. She's conducted STEM workshops reaching over 85,000 students across 46 countries and published a book outlining her problem-solving methodology. "My goal has shifted not only from creating my own devices to solve the world's problems, but inspiring others to do the same as well," she says. "Science is cool, innovating is cool, and anybody can be an innovator."

Watch Gitanjali's TED Talk to learn more about how mentorship shaped her approach to problem-solving. Want to learn more about Gitanjali? Check out TIME's 2020 Kid of the Year article featuring her and her work!

Reflect

Being a computer scientist is more than just being great at coding. Take some time to reflect on how Gitanjali and her work relates to the strengths that great computer scientists focus on building - bravery, resilience, creativity, and purpose.

PURPOSE

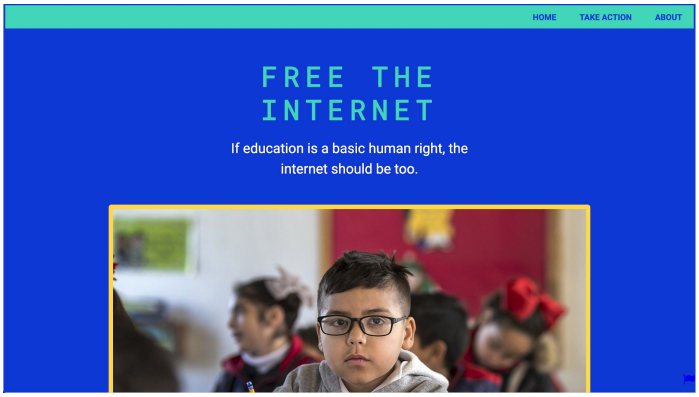
Gitanjali has an interest in the medical field, but still has learned to code and build technology. What other fields are you interested in that you could possibly use computer science for?

Share your responses with a family member or friend. Encourage others to read more about Gitanjali to join in the discussion!

Step 1: Learn About Responsive Design (2-4 mins)

The big question we want to answer with this activity is: **How do I design my website for many different devices?**

Right now, your website only works well on a computer screen that is the size of a laptop. Open your project website and compare how it looks in the smaller *Result* window (to the right of the editor) with the way it looks in full screen mode (as a reminder, click **Share** > **Publish** > **Site URL**).

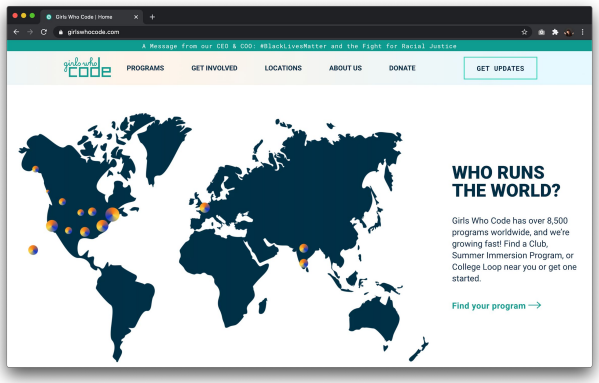


In the smaller size screen on the right, the heading text is not aligned properly, there is too much space on the sides, and the images are too small to see.

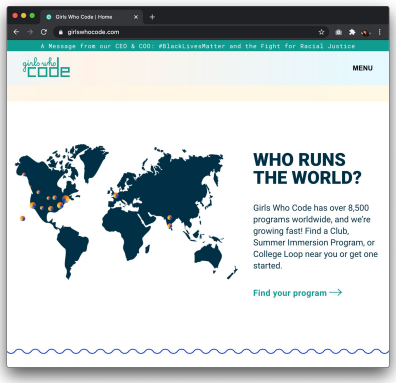
The smaller screen probably breaks your styling. That is, it breaks the styling rules of your current CSS and doesn't appear right. If you look at your project on a smartphone, it will also likely break.

In order to design for multiple size screens, we need to use **responsive design**. This is a way of styling your website so the layout adapts based on the size of the screen.

Let's look at the Girls Who Code website as an example:



Laptop
769px to 1024px



Tablet
481px to 768px

Mobile
320px to 480px

The laptop screen displays the full navigation bar and positions the map image next to the map text. The tablet screen condenses the nav bar to a drop down and reduces the map image size. The mobile screen wraps the map text above and below the map image. All of these changes increase the website's usability across screen sizes.

Responsive websites are designed so that the elements on the page can resize and adjust their layout to fit different screen sizes. These days more and more people are accessing websites on phones and tablets, so it has become even more important to design and build websites that work for multiple screen sizes. You will learn about two techniques you can use to make your site responsive: flexbox and media queries. They allow designers to achieve popular layouts more simply and with cleaner code. Read about them below.

FLEXBOX

Flexbox allows you to arrange and position groups of visual elements on a website more easily than the box model. It is particularly helpful for laying out images.

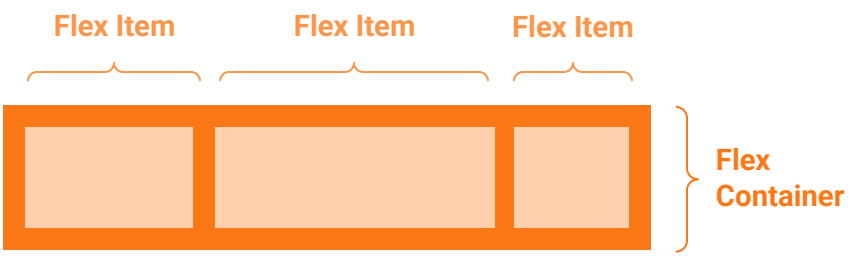
MEDIA QUERIES

Media queries allow you to create CSS rule sets that are only implemented if the screen is a certain size.

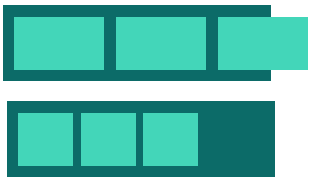
Step 2: Meet Flexbox (2-4 mins)

In the last activity, you learned about the box model as a tool to layout elements on your site. It's great for styling some parts of your site, but it gets complex quickly when you have a collection of elements that you want to size and align in a particular way. Flexbox is great at this. It can easily position a set of three images horizontally from left to right or center an element vertically and horizontally inside another element or style nav bars. Check out this page for more on uses of flexbox.

Flexbox is short for the **CSS Flexible Box Layout Module**, which is a part of the current version of CSS. Flexbox allows HTML elements to change their size and position based on the size of the screen. These elements live in a special flex container that has a defined size. The **flex container** allows elements to shrink, expand, or wrap onto new lines to fill the free space inside the flex container. This means that none of the items will overflow outside the container.



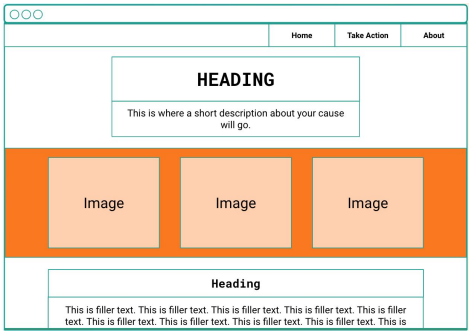
Examples of challenges flexbox helps us solve:



The elements included in the flex container are considered **flex items**. Styling on a flex container will affect all items inside the container, but you can also add properties to the individual flex items for more customization in layout.

Step 3: Add the Flex Container (2 mins)

To get started you'll need to be able to define and select the flex container. You can do this by adding a `<div>` with a class attribute around the elements that you want to layout with flexbox. In this case, the elements we want to layout with flexbox are the images and any corresponding image attributions. If you remember all the way back to Part 3, we added a `<div>` tag around the row of images in our wireframe. We will turn this `<div>` tag into our flex container.



Let's add this to our project now:

- Go into your **index.html** file.
- Add a class attribute to the `<div>` tag that contains *all* the images you want to include. Give it a recognizable name such as "`flex-container`".
- Now we need to create the associated CSS rule set. Go to your **style.css** file.
- Find the `/*Images*/` comment under the `/*HOMEPAGE*/` comment. Since our flexbox styling is for images, we are including it here. You can add another comment to remind yourself that these are flexbox rule sets if you like.
- Under the comment, use the class name you just added to the `<div>` tag in your **index.html** file to define the class CSS selector for your flexbox container. Remember that class selectors begin with a `.` symbol.

index.html

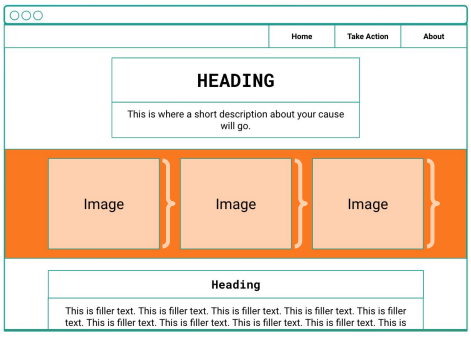
```
<main>
  <!--Images-->
  <div class="flex-container">
    <div>
      <img src="classroom.jpg" alt="Elementary scho
    </div>

    <div>
      <img src="hands.jpg" alt="Two hands of a woma
      <!--WOCinTEch asks for image attribution-->
      <p class="attribution"> Image of hands typing
    </div>
```

style.css

```
/* Images */
.flex-container {


}
```

Step 4: Add the Flex Items (2 mins)

Now that we have our container in place, we need to be able to define and select the flex items. We will follow a very similar process to creating our flex container: assign a class attribute to each `<div>` tag around the elements you want to include as a single flex item. In this case, our flex items are the individual images and any associated image attributions. Since we already added `<div>` tags around each image in Part 3, all we have to do is add the class attribute to the `<div>` tag and create the corresponding rule set in our CSS file.




```
<div class="flex-items">
```

Let's add this to our project now:

- Go to **index.html**.
- Add a class attribute to the first `<div>` tag containing an image (and text if you have it) that you want to include as a flex item. Give it a recognizable name such as "`flex-item`".
- Add a class attribute to the second `<div>` tag containing an image (and text if you have it) that you want to include as a flex item. Give it the **same class attribute name** as the first flex item.
- Repeat the last step for the third image.
- Now we need to create the associated CSS rule set. Go to your **style.css** file.
- Under the rule set you created for the flex container, use the class name you added to the `<div>` tags above to define the class CSS selector for your flex items. Remember that class selectors begin with a `.` symbol.

index.html

```
<main>
  <!--Images-->

  <div class="flex-container">

    <div class="flex-item">

      <img src="classroom.jpg" alt="Elementary schoo
    </div>

    <div class="flex-item">

      <img src="hands.jpg" alt="Two hands of a woma
      <!--WOCinTEch asks for image attribution-->
      <p class="attribution"> Image of hands typing
    </div>

    <div class="flex-item">

      <img src="mask.jpg" alt="Close up of a young
    </div>
  </div>
```

style.css

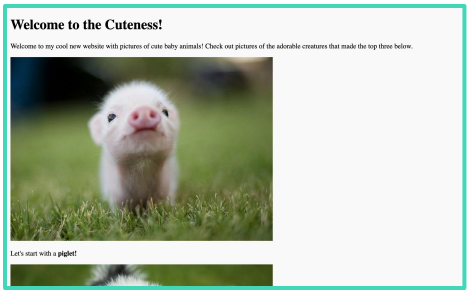
```
/* Images */
.flex-container {
}

.flex-items {
}
```

In the next step, we will explore a few key flexbox styling properties to activate your flex container and items. These are the properties that you will use in your own layout.

Step 5: Examine Flexbox Styling Properties (10 mins)

Let's return to our cute animal website from Part 3 to explore how flexbox styling properties work. Since this site is very simple, we will be able to get a better idea of how these flexbox properties work. After we explore these properties, we will apply them to your project. In this section, you don't have to write any code, but you can remix the project if you like and tinker with the values to see how they work.

First, let's see how the project currently displays:



All of the following elements are stacked on top of each other: a header, introduction text, an image of a piglet, the piglet subtitle, an image of a baby skunk, the baby skunk subtitle, an image of a baby hedgehog, the baby hedgehog subtitle, and image attribution text.

The only CSS rule sets we have now are below. Let's look at those first:

```
img {
   width: 100%;
}
```

We used the `width` property to define the width of our image. Later on, we will want each image to fit the full width of the flex item container. Instead of using pixels, we used a percentage value so the images can adapt to the size of the screen.

```
.subtitle {
   text-align: center;
}
```

In our **index.html file,** we created a class attribute with the name subtitle for the text under our images: `<p class="subtitle"></p>`. In our **style.css** file, we created a ruleset using the `.subtitle` class selector and added a `text-align` property to center the text.

Flexbox Styling Goals

Before we look at any flex properties, let's break down our styling goals:

- **Goal 1:** Layout three cute animal images in a row.
- **Goal 2:** Keep all the images on one line.
- **Goal 3:** If the screen size changes, the images should grow or shrink proportionally.
- **Goal 4:** The image widths should be equal.
- **Goal 5:** There should be a small amount of space around each image so they are not bunched up next to each other.

Based on what we have learned so far, flexbox is a perfect solution for this challenge! To learn about flexbox properties, we will add properties one at a time to see what they do. We'll begin with the flex container.

Flex Container Properties

We can achieve many of our goals with the three properties below. We will only apply these properties to the flex container, since we want them to apply to all of our flex items - in this case, the flex items are our images.

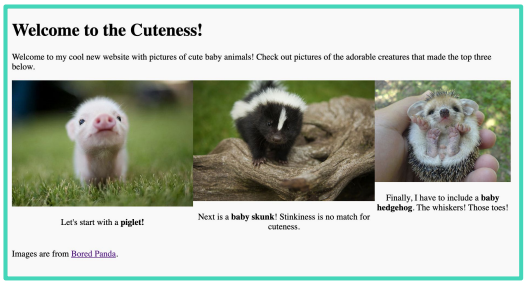
`display`

The first property we need is the `display` property. To create our flex container, we need to set the value of `display` to `flex`. This is the magic that turns our `<div>` into a flex container! Let's add it to the flex container rule set and see what happens.

CSS

```
.flex-container {
   display: flex;
}
```

RESULT



GOAL UNLOCKED

Goal 1: Layout three cute animal images in a row.

`flex-direction`

Right now, our images are already in a row. However, we still need to specify to our browser that we want our flex items to be in a row. We can use the `flex-direction` property and set it to row to achieve this. The `flex-direction` property tells the browser which direction the flex items should go in: left to right, right to left, top to bottom, bottom to top. Basically you can think of it as telling the browser to create a row or column. The `flex-direction` property can be set to row, row-reverse, column, or column-reverse.

CSS

```
.flex-container {
   display: flex;
   flex-direction: row;
}
```

RESULT

GOAL UNLOCKED

Goal 1: Layout three cute animal images in a row.

flex-wrap

Flex items will try to fit on one line by default. Sometimes you may want to change this and allow items to wrap and position items on the next line. According to Goal 2, we only want our images on one line, so we want to define that property and set the value to `nowrap`. This means our images won't go onto more than one line. The `flex-wrap` property can be set to `nowrap`, `wrap`, or `wrap-reverse`.

CSS

```
.flex-container {
    display: flex;
    flex-direction: row;
    flex-wrap: nowrap;
}
```

RESULT

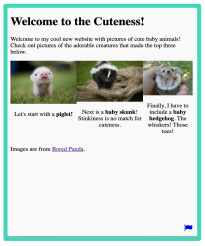
Note: Setting the value to `wrap` would place items on a new line.

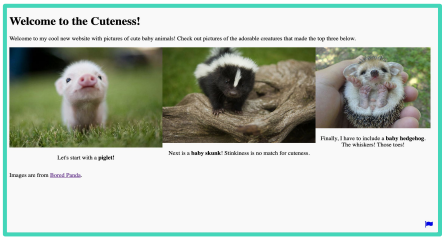
GOAL UNLOCKED

Goal 2: Keep all the images on one line.

Those are the three main properties we need. Before we completely close that curly bracket, let's test in to check in on **Goal 3** - changing the image size proportionally to the screen size.

Drag the right side of your screen as far as you can to the left. You should see the images change as the size of the browser window changes!





Flexbox Item Properties

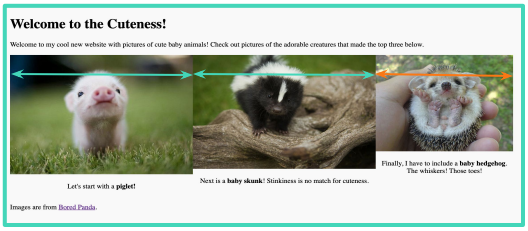
We can add properties to flex items as well. There are special flex item properties you can apply in addition to the regular CSS properties you learned about in Part 4. In this project, we only need to use one special flex item property: `flex-basis`. We will also include a property you are already familiar with: `margin`. We include both of these properties in the `.flex-item` rule set.

Note: We do not need to define the `display` property for flex items, only for flex containers.

flex-basis

This `flex-basis` property sets the initial size of the flex item before space is distributed according to the rules in the flex container. We are going to use this property to define the width of our elements. By using percentage values, the images will grow or shrink proportionally.

Because we set our `flex-wrap` property to `nowrap`, all of the images will show up on one line. Right now, they are **not** equal width because the image sizes are all different, so we need to define the width we want our items to have. We set `flex-basis` to `100%` to tell the flex container that each element inside of it should display at equal widths.

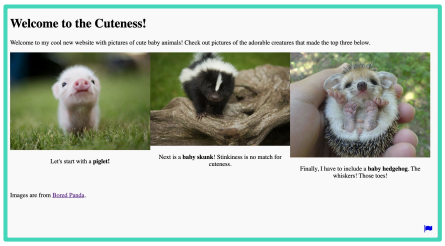


The baby hedgehog is not the same width.

CSS

```
.flex-item {
    flex-basis: 100%;
}
```

RESULT



The baby hedgehog is the same width!

GOAL UNLOCKED

Goal 4: The image widths should be equal.

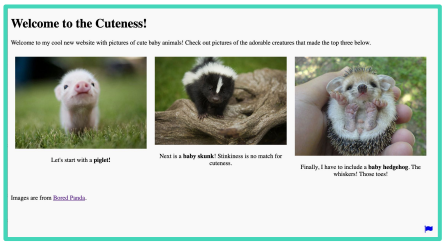
margin

We want some space around our flex items so the images are not touching each other. One solution is to use the `margin` property to set a margin around the flex items. If we set our `margin` value to 1%, here is the result:

CSS

```
.flex-item {
    flex-basis: 100%;
    margin: 1%;
}
```

RESULT



GOAL UNLOCKED

Goal 5: There should be a small amount of space around each image so they are not bunched up next to each other.

That's it! We accomplished all of our goals for our cute animal website. In the next section, we will practice applying these concepts to your Activist Toolkit project.

TAKE IT FURTHER

There are so many more flex container and flex item properties, but these are the only ones we will discuss. Flexbox might feel confusing at first, so try different properties out one by one. To learn more about the flexbox properties and their values take a look at this CSS Tricks' A Complete Guide to Flexbox. We also recommend playing Flexbox Froggy for some fun practice with Flexbox properties.

Step 6: Add Flexbox Properties to Your Project (5-7 mins)

Now that we have learned more about the flex container and flex item properties we can use to style our row of images, let's add them to your Activist Toolkit project.

Add the Flex Container Properties

Include the following properties inside your flex container rule set. Our class selector name is `.flex-container`, but you might have chosen a different name. Remember: It should be the same name as the class attribute in your **index.html** file.

CSS

```
.flex-container {
   display: flex;
   flex-direction: row;
   flex-wrap: nowrap;
}
```

DESCRIPTION

- `.flex-container:` Add the class selector.
- `display:` Create the flex container by setting this property to `flex`.
- `flex-direction:` Display flex items in a row or column.
- `flex-wrap:` Keep all the flex items on one line.

Add the Flex Item Properties

Include the following properties inside your flex item rule set. Our class selector name is `.flex-item`, but you might have chosen a different name. Remember: It should be the same name as the class attribute in your **index.html** file.

CSS

```
.flex-item {
   flex-basis: 100%;
   margin: 1%;
}
```

DESCRIPTION

- `.flex-item:` Add the class selector.
- `flex-basis:` Set the size of each item before any extra space is distributed. We want all the items to fit on the line equally, so we won't have any extra space.
- `margin:` Add some space around the flex items. Try different values until you are satisfied with how it displays.

Test It

Let's see how it displays:

- Save your project.
- Open your project in a new tab if it isn't already. (You can do this by clicking **Share** > **Publish** > **Site URL**).
- Reload the tab with your full website so it updates with the changes.
- Drag the right side of your browser window to the left so the screen size decreases. You should see the images change in size as the screen gets smaller.

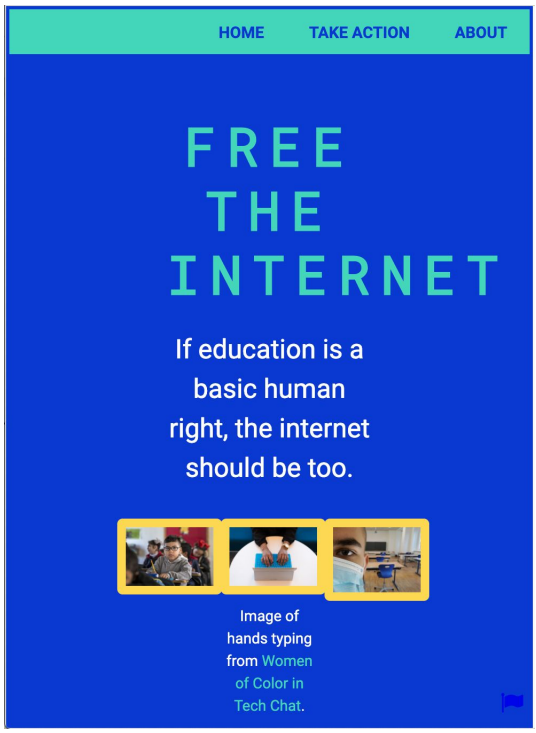
If it's not working the way it's supposed to, go back and make sure that:

- There are no typos in the properties and values.
- There is a semicolon `;` at the end of each property.
- All your curly brackets `{}` are closed.
- The CSS selector for your flex container and flex item rule sets corresponds to the `<div>` class attribute in your **index.html** file.

Step 7: Meet Media Queries (2-4 mins)

If you remember back to the top of this section, we defined responsive web design as a way of styling your website so it changes or adapts based on the size of the screen. Flexbox is one tool you can use for responsive design, but there are times when you might want to change the styling of an element based on the screen size.

For example, if you resize your browser window as small as it will go, you might notice that it is quite difficult to see your row of images. Flexbox will only take us so far. We also need to use **media queries**.



Media queries are a second tool that allows you to tailor your CSS based on the device's screen size or a media type. They can be used to check all sorts of possible scenarios:

- Width and height of the visible area of a webpage (also called the viewport)
- Width and height of a device
- Orientation (i.e. landscape or portrait mode)
- Screen resolution

We will only focus on screen size media queries, but you can learn more about other media types from W3Schools.

Now let's take a deep dive into the syntax.

`@media`

To create a media query, we use `@media` rules to define the type of media we want to apply the styles to and any conditions (like screen size) that the browser should test for. If the condition set up by this rule is met, the browser applies the CSS rule sets inside the curly brackets.

```css
@media mediatype and (media feature) {
/* CSS rule sets */
}
```

Here is an explanation of the syntax:

- `@media`: The keyword to indicate there is a media query.
- `mediatype`: Specifies the type of media that the styles will apply to. These values can be screen, print, speech, or all. We are only covering screen.
- `and`: This keyword adds the first condition (the mediatype) to the second condition (the media feature).
- `()`: We use parentheses to indicate we are setting a condition using one or more media features.
- `media feature`: These features let us test for certain conditions of the media query, such as whether or not the screen is greater than a certain size. You can combine multiple features together using and, or, not, etc. See W3School's full list of media features.
- `{ }`: All rule sets that will be applied if the condition is true must be in curly brackets.
- `/* CSS Rule Sets */`: Add all CSS rule sets you want to apply if the condition is met. You can add as many rule sets as you like.

Step 8: Add a Breakpoint to Your Site (5 mins)

Breakpoint is a term you will hear a lot in responsive design. A breakpoint is the point when a media query is introduced in order to prevent the style from breaking. You have probably seen a website "break" when it's not designed for mobile. The images or font size might get too small to see, the text might move outside of a box, a button gets squished to the side, etc.

Try resizing your browser window to see how your elements change based on the size of the screen. What do you think the size of the screen is when it “breaks”?

Here are some common breakpoints you can use as a reference:

- 320px to 480px for small devices (like smart phones)
- 481px to 768px for medium-sized devices (like tablets)
- 769px to 1024px for medium to large devices (like laptops)
- 1025px to 1200px for large devices (like desktops)
- 1201px and more for extra large devices (like TVs)

Add a Breakpoint

Let's create a first breakpoint using the `@media` rule to fix it. Right now, if our screen is roughly less than 850 pixels, the flex items (i.e. images) become extremely small. Instead of being in a row on only one line, it would be better if they were in a column on multiple lines. This sounds like the `flex-wrap` property we learn about earlier! We can just tell the images to wrap (i.e. move) onto the next line when the screen reaches a certain size.

Let's add this to our project now:

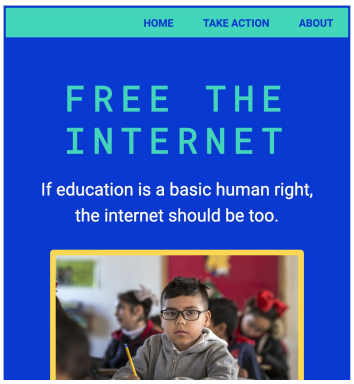
- Go to the **styles.css** file. Find the comment block that says `/* CHANGE CSS BASED ON SCREEN SIZE*/`.
- Under the `/*Breakpoint 1*/` comment, add an `@media` rule that checks to see if the screen has a maximum width of 850 pixels.
- Now we need to add the CSS rule sets we want to change when this condition is met. Inside the curly brackets of your media query, create another rule set for the class selector of your flex container. Our selector is `.flex-container` but yours may be different.
- Add the `flex-wrap` property and set its value to `wrap`.
- If you want to include other rule sets, you can add them below `.flex-container`.

```
/*Breakpoint 1*/
@media screen and (max-width: 850px){

    /*Wrap images onto new lines*/
    .flex-container {
        flex-wrap: wrap;
    }

    /*Add more rule sets if you want*/
    header, main {
        padding-left: 50px;
        padding-right: 50px;
    }

}
```

This property tells the browser to wrap the flex items onto another line if the screen is less than 850 pixels. Let's test it. Save your project and reload or open it in a full size browser window, then drag the side of the browser window to make it smaller.

It works! When the screen width reaches 850 pixels, the images wrap onto new lines. If you have other changes you want to make based on screen size, follow this same process. You can make as many changes as you want.

You can also add more breakpoints if you want. If you want to learn more about responsive web design, check out these resources from Mozilla and Free Code Camp.



Step 9: Extensions (5-15 mins)

In this activity series, we only scratched the surface of what you can do with HTML and CSS. You can continue adding more content to your site with HTML or keep styling it with CSS selectors and rule sets.

Here are a few other ideas you could try out:

Extension 1: Finish Customizing Your CSS

If you haven't finished adding your CSS or want to experiment with some new styling properties, try this extension!

Extension 2: Add Another Page to Your Site

You can add a resources page for your audience to get more information or a page focused on another topic related to your cause.

Extension 3: Make Your Website More Accessible

Read about the principles from the Web Accessibility Initiative, then try to implement one or more on your site.

Extension 4: Add More Media

Media like images, audio, and videos can help you communicate your message in ways text cannot. Try adding one or more below:

- Embedding YouTube videos with `<iframe>` is a fun way to make your website interactive. Get started with this tutorial from W3Schools.
- An image gallery is a great option if you want to give your audience a set of visuals related to your cause.
- If you collected audio stories and want to add them to your website, you can upload the files to your asset folder, then use the `<audio>` tag to add them to your site. Here is a tutorial from W3Schools to get you started.

Once you've finished your site, share it with your audience! Remember: The web is a powerful way to communicate a message. Coding is just one way you can take action for causes that are important to you.