Chapter 5

Color and Images Manipulation

Colors and images are a very import feature in designing an app user interface, UI. Colors help express hierarchy, establish brand presence, give meaning, and indicate element states. For example, when an element's color contrasts with its surroundings, that element stands out, so users can tell it's important. In this chapter, students will learn how to choose and manipulate color and image in an app.

5.1. Colors

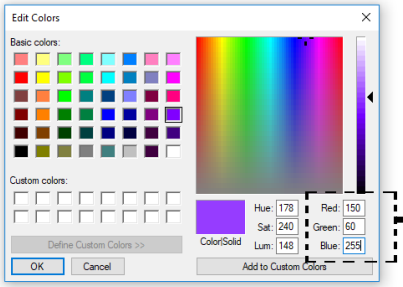
The color property allows us to specify the color of text inside an element. We can specify any color in CSS in one of three ways:

Color Names

There are 147 predefined color names that are recognized by browsers. For example: `darkcyan`

RGB Values

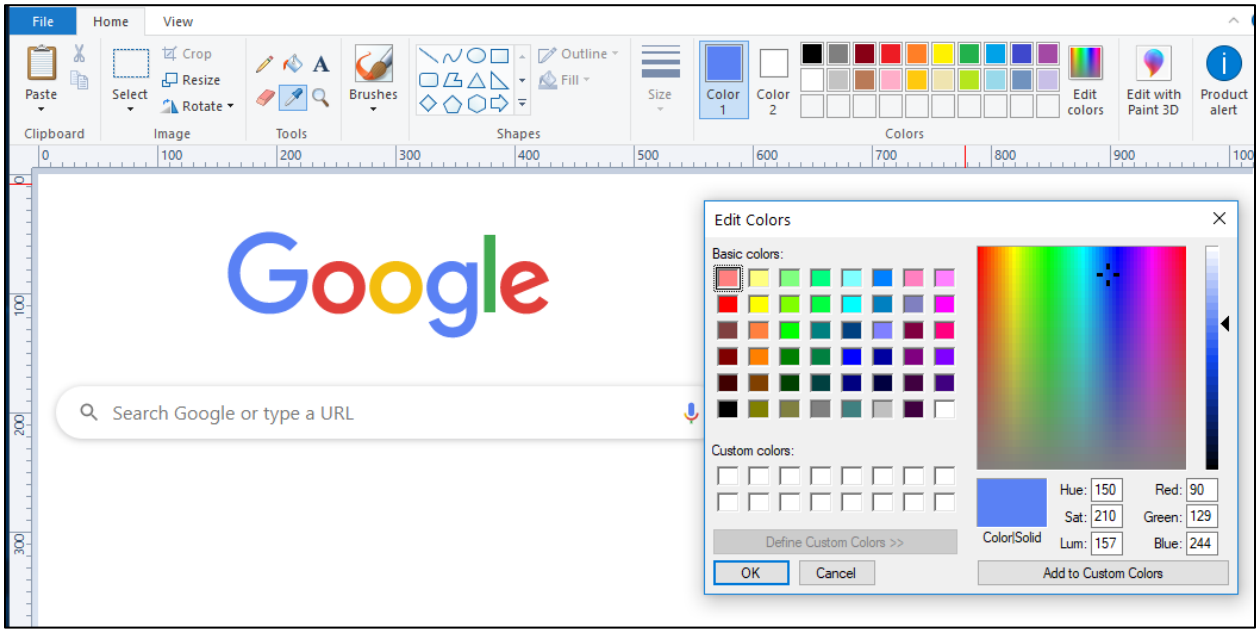
These express colors in terms of how much red, green and blue are used to make it up. Every color on a computer screen is created by mixing amounts of red, green, and blue. To find the color you want, we can use a color picker.



`rgb(150, 60, 255)`

Activity) Prepare a cordova project folder (chapter 1). Open a text editor and save a HTML file as *index*.html in the www folder within the cordova project folder.

Now, use the color value of the Google logo as a background color of a paragraph. We can use Paint, color picker, to find the rgb values.

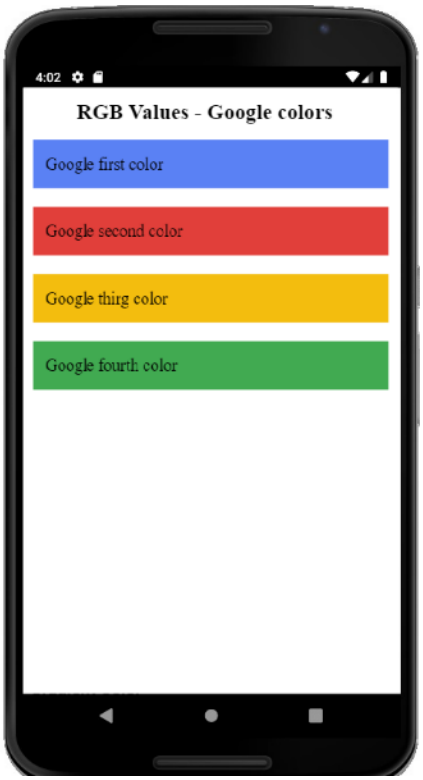


HTML

```
<h3>RGB Values - Google colors</h3>
<p class="rgb1"> Google first color</p>
<p class="rgb2"> Google second color</p>
<p class="rgb3"> Google thirg color</p>
<p class="rgb4"> Google fourth color</p>
```

CSS

```
.rgb1{
  background-color: rgb(66,133,244);
  padding: 12px;
}
.rgb2{
  background-color: rgb(234,67,53);
  padding: 12px;
}
.rgb3{
  background-color: rgb(251,188,5);
  padding: 12px;
}
.rgb4{
  background-color: rgb(52,168,83);
  padding: 12px;
}
```



HEX Codes

These are six-digit codes that represent the amount of red, green and blue in a color, preceded by a pound or hash # sign. For example: `#ee3e80`

A **primary color** is the color displayed most frequently across your app's screens and components. If you don't have a secondary color, your primary color can also be used to accent elements.

You can make a color theme for your app using your primary color, as well as dark and light primary variants.

A **secondary color** provides more ways to accent and distinguish your product. Having a secondary color is optional, and should be applied sparingly to accent select parts of your UI.

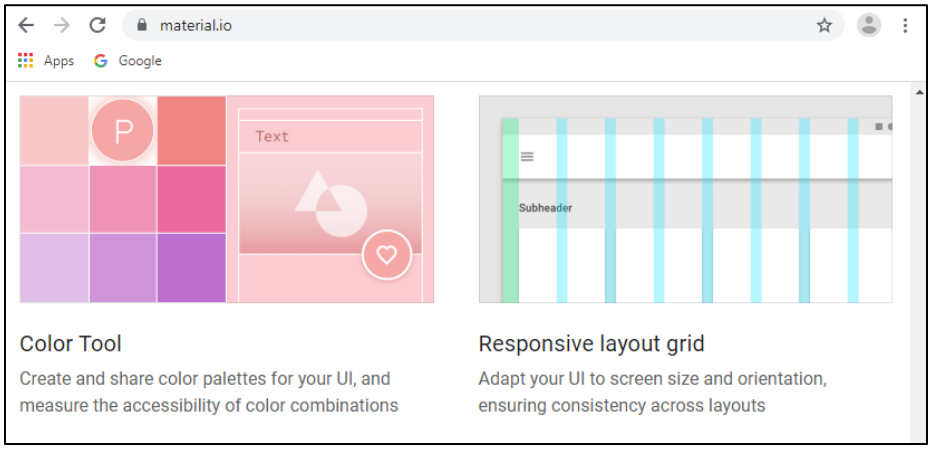
Secondary colors are best for:

- Floating action buttons
- Selection controls, like sliders and switches
- Highlighting selected text
- Progress bars
- Links and headlines

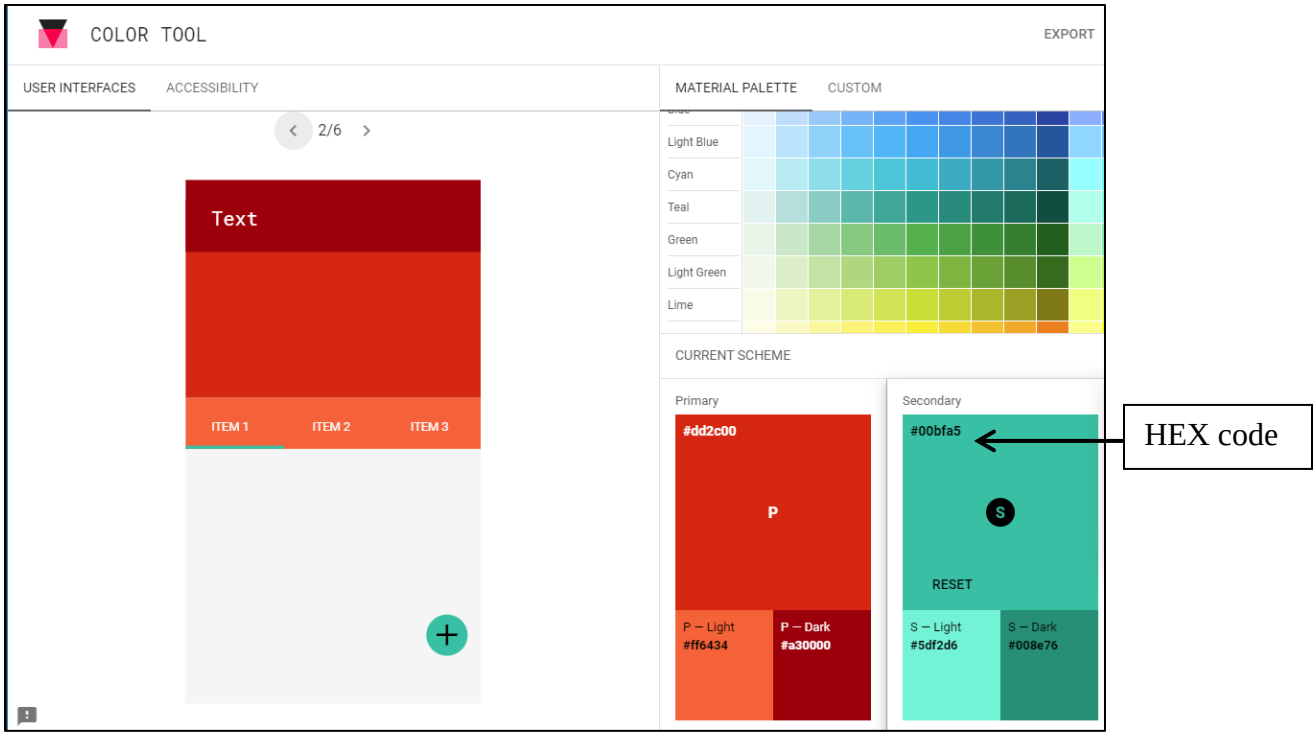
Just like the primary color, your secondary color can have dark and light variants. You can make a color theme by using your primary color, secondary color, and dark and light variants of each color.

READ MORE

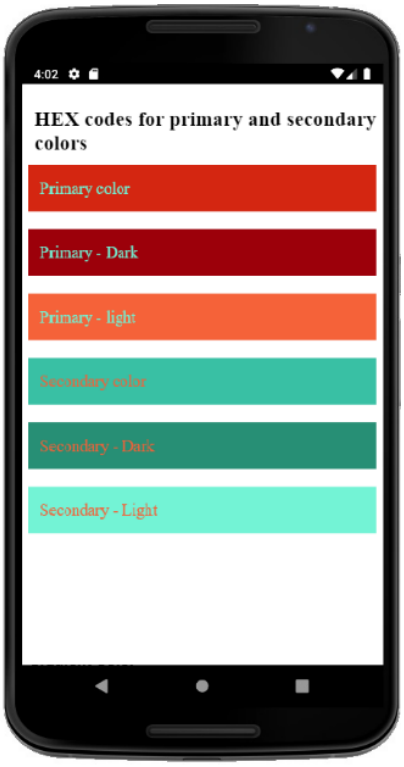
Activity) Go to material.io website, scroll down, and click on **Color Tool**



In the Color Tool website, select a primary and secondary color. On top of the color is the HEX code.



Use the primary and secondary colors, including the light and dark color, as the background-color of a paragraph:



HTML

```
<h3>HEX codes for primary and secondary colors</h3>
    <p class="primary">Primary color</p>
    <p class="primary_dark">Primary - Dark</p>
    <p class="primary_light">Primary - light</p>
    <p class="sec">Secondary color</p>
    <p class="sec_dark">Secondary - Dark</p>
    <p class="sec_light">Secondary - Light</p>
```

CSS

```
.primary{
  background-color: #dd2c00; padding:12px; color:#5df2d6;
}
.primary_dark{
  background-color: #a30000; padding:12px; color:#5df2d6;
}
.primary_light{
  background-color: #ff6434; padding:12px; color:#5df2d6;
}
.sec{
  background-color: #00bfa5; padding:12px; color:#ff6434;
}
.sec_dark{
  background-color: #008e76; padding:12px; color:#ff6434;
}
.sec_light{
  background-color: #5df2d6; padding:12px; color:#ff6434;
}
```

rgba( ) color

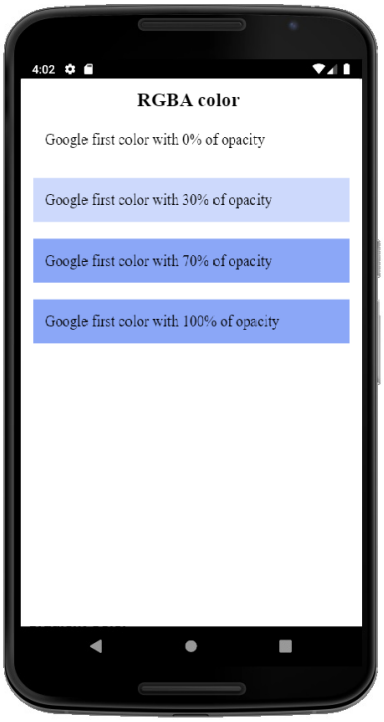
The `rgba( )` function define colors using the red-green-blue-alpha (RGBA) model where alpha specifies the opacity of the color. Alpha color goes from 0 to 1, 0 means fully opaque (0% of opacity) and 1 means fully opaque (100% of opacity). The syntax of the code is:

```
rgba(red, green, blue, alpha)
```

For example, for color **`rgb(150, 60, 255)`**with 35% of opacity, we can write the code as:

`rgba(150, 60, 255,0.35)`

Activity) Using the color of the first letter of the Google logo as a background-color of a paragraph. Apply 0%, 30%, 70%, and 100% of opacity.



HTML

```
<h3>RGBA color</h3>
<p class="rgba1">Google first color with 0% of opacity</p>
<p class="rgba2">Google first color with 30% of opacity</p>
<p class="rgba3">Google first color with 70% of opacity</p>
<p class="rgba3">Google first color with 100% of opacity</p>
```

CSS

```
.rgba1{
  background-color: rgba(66,133,244,0); padding: 12px;
}
.rgba2{
  background-color: rgba(66,133,244,0.3); padding: 12px;
}
.rgba3{
  background-color: rgba(66,133,244,0.7); padding: 12px;
}
.rgba4{
  background-color: rgba(66,133,244,1); padding: 12px;
}
```

5.2. CSS linear-gradient

CSS gradient display smooth transition between two or more specified colors. Those colors are the colors that we want to render smooth transition among the gradient space. For example, we can create a **`linear-gradient`** of the red and yellow color from top to bottom as the following:

HTML

```
<div class="gradient1"></div>
```

CSS

```
.grading1 {
  height: 200px;
  background-image: linear-gradient(red, yellow);
}
```

`linear-gradient` can set the color from left to right. For this, we can add **`to right`** value to the `background-image` property as the following:

`background-image: linear-gradient(to right, red, yellow);`

HTML

```
<div class="gradient2"></div>
```

CSS

```
.grading2 {
  height: 200px;
  background-image: linear-gradient(to right, red, yellow);
}
```

We can also set the linear-gradient from bottom-left to top-right:

`background-image: linear-gradient(to top right, red, yellow);`

HTML

```
<div class="gradient3"></div>
```

CSS

```
.grading3 {
  height: 200px;
  background-image: linear-gradient(to top left, red,
yellow);
}
```

5.3. Position Property

One importation property of CSS layout is the `position` property. The `position` property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

position: static;

HTML elements are positioned static by default.
Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page.

position: relative;

An element with position: relative; is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

position: absolute;

An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: A "positioned" element is one whose position is anything except static.

Background image has `position:relative;` in CSS

Message has `position:absolute;` in CSS

Activity) Create a container with a background image and a text message in front of it. The text message has to have opacity.

We can create a container using ** ** element

HTML

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="index.css" rel="stylesheet" type="text/css"/>
    <title>Activity 5 by Huixin Wu</title>
  </head>
  <body>
    <h3> Position relative and absolute </h3>
    <div class="imageContainer">
      <img src="img/image.jpg">
    </div>
  </body>
</html>
```

CSS

```
img{ width: 100%; height: 100%;}
.imageContainer{
  width: 80%;
  height: 160px;
  margin: auto;
  position: relative;
}
```

After it, we can add the text message using another ** ** or **<p>** in the HTML file, and then use **position:absolute;** in CSS.

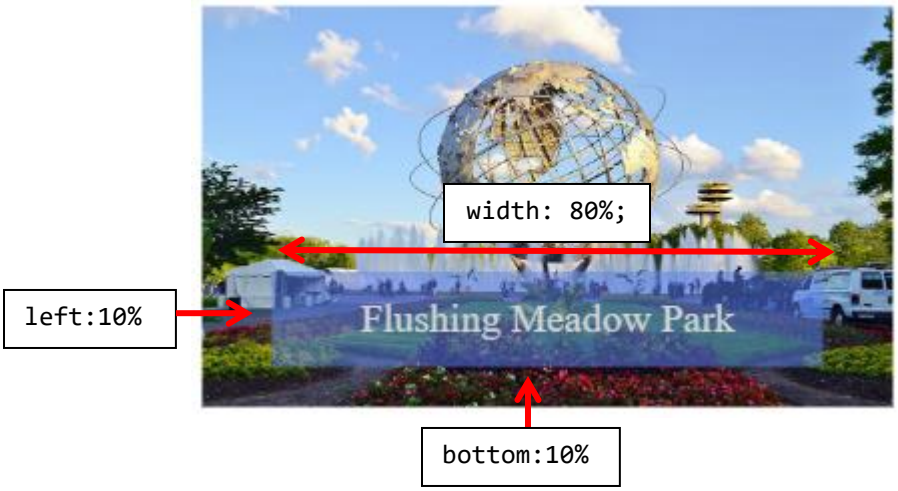
According to the text message position, the text has a is center in the bottom

HTML

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="index.css" rel="stylesheet" type="text/css"/>
    <title>Activity 5 by Huixin Wu</title>
  </head>
  <body>
    <h3> Position relative and absolute </h3>

    <div class="imageContainer">
      <img src="image.jpg">
      <div class="message"> Creating Smartphone Apps</div>
    </div>
  </body>
</html>
```

css

```
.message{
  position: absolute;
  bottom:10%;
  left: 10%;
  width: 80%;
  background-color: rgba(66,133,244,0.5);
  text-align: center;
  padding: 10px;
  color: lightgray;}
```



position: sticky;

An element with `position: sticky;` is positioned based on the user's scroll position.

A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like `position:fixed`).

Activity) Create a message box using `position:sticky;`

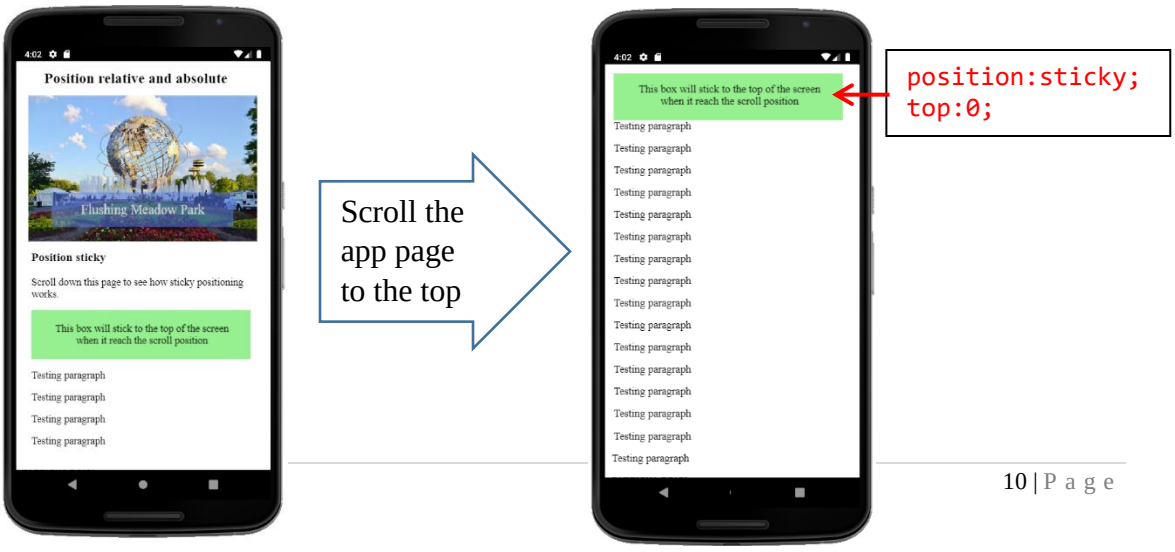
We can create a container for the sticky message using ** ** element with `position:sticky;` and the position to stick to, in this case, we can set the message to stick on the top of the screen using `top:0;`

HTML

```
<h3>Position sticky </h3>
<p>Scroll down this page to see how  sticky positioning works.</p>

<div class="stickyBox">This box will stick to the top of the screen when it reach
the scroll position</div>

<p>Testing paragraph </p> <p>Testing paragraph </p>
<p>Testing paragraph </p> <p>Testing paragraph </p>
<p>Testing paragraph </p> <p>Testing paragraph </p>
<p>Testing paragraph </p> <p>Testing paragraph </p>
<p>Testing paragraph </p> <p>Testing paragraph </p>
<p>Testing paragraph </p> <p>Testing paragraph </p>
<p>Testing paragraph </p> <p>Testing paragraph </p>
<p>Testing paragraph </p> <p>Testing paragraph </p>
```

css

```
.stickyBox{
  font-size: 20px;
  text-align: center;
  padding: 20px;
  background-color: lightgreen;
  position: sticky;
  top:0;
}
```



5.4. Hero image

A hero image is a large image with text places on the top of it. For this, we will use **`linear-gradient`** and the background image will be inserted from the CSS file instead of the HTML file.

Activity) The following hero image:

First, we create a container with a background image:

HTML

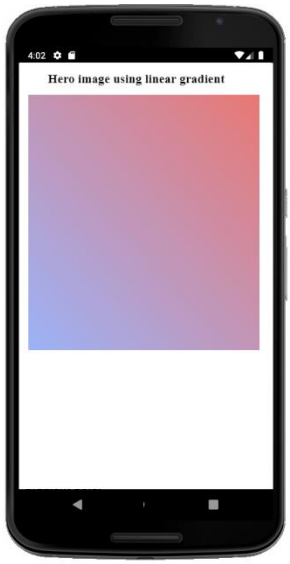
```
<h3>Hero Image </h3>
  <div class="heroContainer">

  </div>
```

Now in CSS file, we are going to create a linear-gradient with two different colors:

CSS

```
.heroContainer{
  height: 380px;
  background-image:linear-gradient(to top right, rgba(66,133,244,0.6),
     rgba(234,67,53,0.7));
  position: relative;
}
```

The app view should look as:

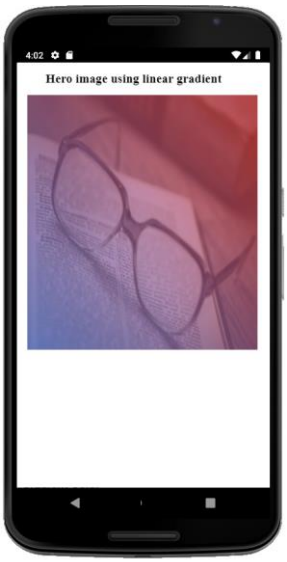


After the linear-gradient, we can insert a background image to the `background-image` property. Also, after we insert the background image, we adjust the width and height using **`background-size: 100% 100%;`** which means that background image with 100% fit the width and the height.

CSS

```
.heroContainer{
  height: 380px;
  background-image:linear- gradient(to top right, rgba(66,133,244,0.6),
rgba(234,67,53,0.7)), url("../img/book.jpg");
  background-size: 100% 100%;
  position: relative;
}
```

The app view should look as the following:



Once we have the container and the background image set, we can create the text that will sit on top of the background image. We can name the text container **`heroText`**

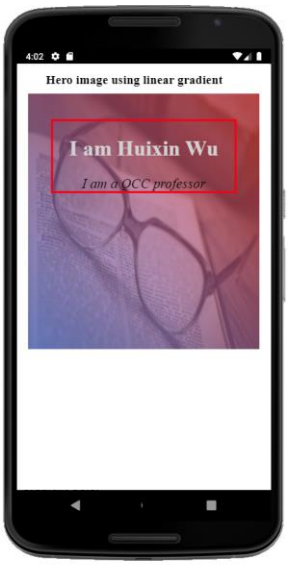
HTML

```
<h3>Hero Image </h3>
    <div class="heroContainer">
      <div class="heroText">
          <h1>I am Huixin Wu</h1>
          <i>I am a QCC professor</i>
          <br>
      </div>
    </div>
```

Now in CSS file:

CSS

```
.heroText{
  position: absolute;
  top: 10%;
  width: 80%;
  left: 10%;
  text-align: center;
}
.heroText h1{color:lightgray; }
.heroText i{ font-size: 1.3em; }
```



Remember that the border in **.heroText** is for reference, once the **.heroText** is done, we can remove it.

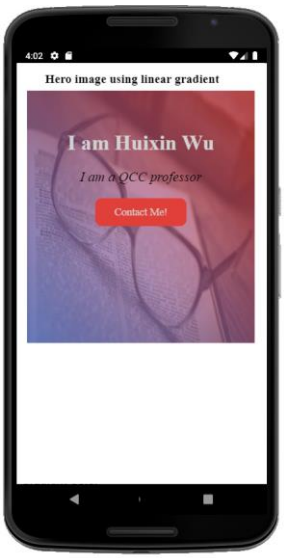
After it, we can add a Contact Me! hyperlink, using **<a>** element, in the **.heroText**

HTML

```
<div class="heroContainer">
      <div class="heroText">
          <h1>I am Huixin Wu</h1>
          <i>I am a QCC professor</i>
          <br>
          <a href="mailto:hwu@qcc.cuny.edu">Contact Me!</a>
      </div>
</div>
```

CSS

```
.heroText a{
  display: block;
  text-decoration: none;
  color: lightgray;
  background-color: rgb(234,67,53);
  padding: 12px;
  width: 50%;
  margin: auto;
  margin-top: 20px;
  border-radius: 10px;
}
.heroText a:hover{
  background-color: rgb(66,133,244);
  font-weight: 600;
}
```



Once the **.heroText** is set, we can remove the **border**

5.5. Filter images

Filter property adds visual effects to an element. Some of the filter attributes are: blur, contrast, sepia, grayscale, and invert.

blur(px)

`blur` applies a blur effect to an element. The value, in pixels *px*, between the parenthesis defines the blurriness of the element, a larger value will create more blur.

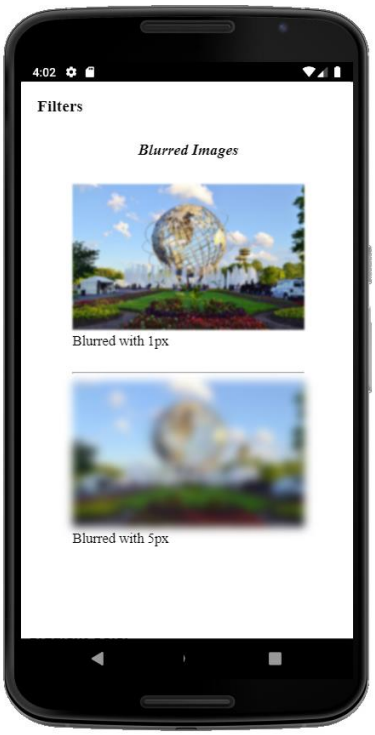
Activity) Create four different blur effects to an image with 1px, 5px, 10px, and 15px.

HTML

```
<figure>
      <figcaption><b>Blurred Images</b></figcaption>
      <br>
      <img src="img/park.jpg" class="blur1">Blurred with 1px <br><br><hr>
      <img src="img/park.jpg" class="blur2">Blurred with 5px<br><br><hr>
      <img src="img/park.jpg" class="blur3">Blurred with 10px<br><br><hr>
      <img src="img/park.jpg" class="blur4">Blurred with 15px<br><br><hr>
</figure>
```

CSS

```
.blur1{filter: blur(1px); }
.blur2{filter: blur(5px); }
.blur3{filter: blur(10px);}
.blur4{filter: blur(15px);}
```



contrast(%)

contrast adjusts the contrast of an element. If the element is used as the container for an image, then contrast will adjust the contrast of the image.

contrast(0%) will make the image completely black and **contrast(100%),** which is by default, will show the original image, and values over 100% will provide results with more contrast:

HTML

```
<figure>
      <figcaption><b>Images with contrast</b></figcaption>
      <br>
      <img src="img/park.jpg" class="contrast1">0% of contrast<br><br><hr>
      <img src="img/park.jpg" class="contrast2">30% of contrast<br><br><hr>
      <img src="img/park.jpg" class="contrast3">60% of contrast<br><br><hr>
      <img src="img/park.jpg" class="contrast4">90% of contrast<br><br><hr>
    </figure>
```

CSS

```
.contrast1{filter: contrast(0%);}
.contrast2{filter: contrast(30%);}
.contrast3{filter: contrast(60%);}
.contrast4{filter: contrast(90%);}
```

invert(%)

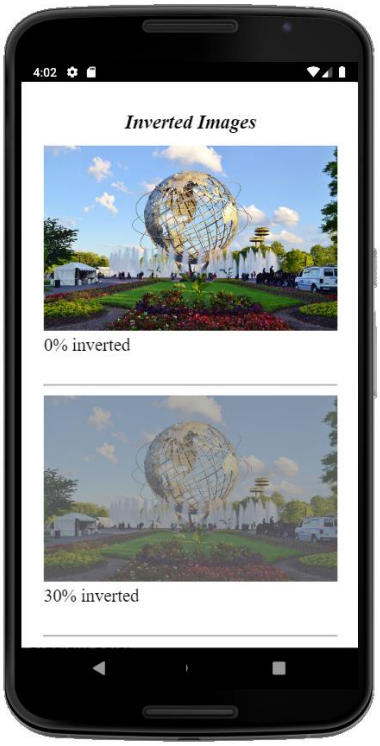
`invert( )` inverts the sample color of the element. **`invert(0%)`** is default and means that there isn't any color inversion in the element. **`invert(100%)`** will make the image will invert completely its sample color.

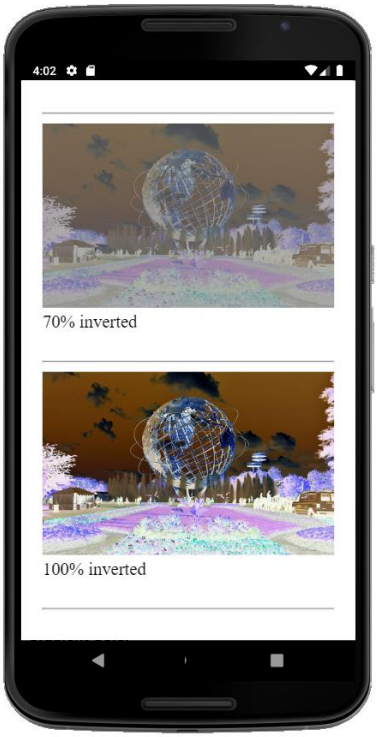
HTML

```
<figure>
      <figcaption><b>Inverted Images</b></figcaption>
      <img src="img/park.jpg" class="invert1">0% inverted<br><br><hr>
      <img src="img/park.jpg" class="invert2">30% inverted<br><br><hr>
      <img src="img/park.jpg" class="invert3">70% inverted<br><br><hr>
      <img src="img/park.jpg" class="invert4">100% inverted<br><br><hr>
</figure>
```

CSS

```
.invert1{filter: invert(0%);}
.invert2{filter: invert(30%);}
.invert3{filter: invert(70%);}
.invert4{filter: invert(100%);}
```





Bibliography

Duckett, J. (2016). *HTML and CSS Design and build websites.* Indianapolis: John Wiley and Sons Inc.

Duckett, J. (2016). *JavaScript and JQuery: interactive front-end developer.* Indianapolis: John Wiley and Sons Inc.

HTML, CSS, JavaScript, JQuery, and Bootstrap. (2017, June). Retrieved from w3schools: www.w3schools.com