

3. Cascading Style Sheets (CSS)

3.1. Introduction to CSS

3.1.1 CSS Basics

CSS (Cascading Style Sheets) allows you to create rules that specify how the content of an element should appear. For example, you can specify that the background of the page is cream, all paragraphs should appear in gray using the Arial typeface, or that all level one headings should be in a blue, italic, Times typeface.

CSS works by associating rules with HTML elements. These rules govern how the content of specified elements should be displayed. A CSS rule contains two parts: a **selector** and a **declaration**.

Selectors indicate which element the rule applies to. The same rule can apply to more than one element if you separate the element names with commas.

Declarations indicate how the elements referred to in the selector should be styled. Declarations are placed between brackets.

Declarations are split into two parts (a property and a value) are separated by a colon.

selector
{
p {color: red;}
}
declaration

Within declaration

The declaration should specify the property and value of element of the selector.

Properties indicate the aspects of the element you want to change. For example, color, font, width, height and border. Each declaration ends with a semicolon. You can always refer to w3schools.com to check the list of all properties that is available in CSS.

Values specify the settings you want to use for the chosen properties. For example, if you want to specify a color property then the value is the color you want the text in these elements to be.

p { color: red; }
property value

Where should I place my CSS instructions?

Externally: Your CSS code can be written in a text editor and saved with file extension .css. Then the .css can be found using the link element.

Internally: You can also include CSS rules within an HTML page by placing them inside a **<style>** element, which usually sits inside the <head> element of the page.

Using external CSS

The **<link>** element can be used in an HTML document to tell the browser where to find the CSS file used to style the page. It is an empty element (meaning it does not need a closing tag), and it lives inside the <head> element. It should use three attributes:

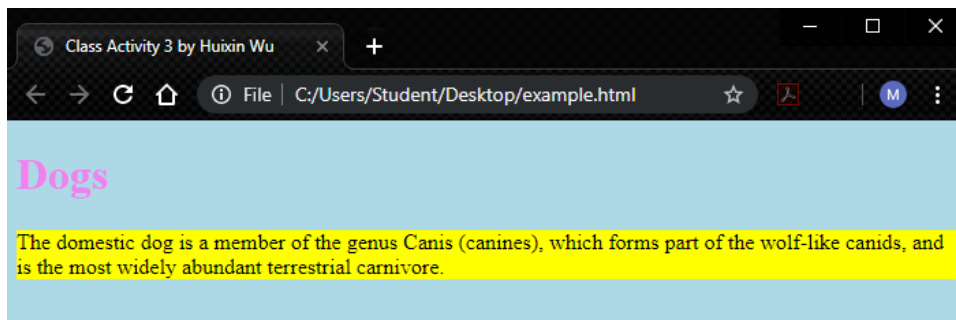
href: This specifies the path to the CSS file (which is often placed in a folder called css or styles).

type: This attribute specifies the type of document being linked to. The value should be text/css.

rel: This specifies the relationship between the HTML page and the file it is linked to. The value should be stylesheet when linking to a CSS file.

An HTML page can use more than one CSS style sheet. To do this it could have a <link> element for every CSS file it uses. For example, some authors use one CSS file to control the presentation (such as fonts and colors) and a second to control the layout.

Exercise) Create a simple app with **<h1>** and **<p>** element. Save an external .css file as **style.css**



```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Class Activity 3 by Huixin Wu</title>
    <link href="style.css" type="text/css" rel="stylesheet" />
  </head>
  <body> <h1>Dogs</h1>
  <p> The domestic dog is a member of the genus Canis (canines), which forms part of
the wolf-like canids, and is the most widely abundant terrestrial carnivore.</p>
</body>
</html>
```

HTML

```
body {
background-color: lightblue;
}
h1 {color:violet;}
p {background-color: yellow;}
```

style.css

Using Internal CSS

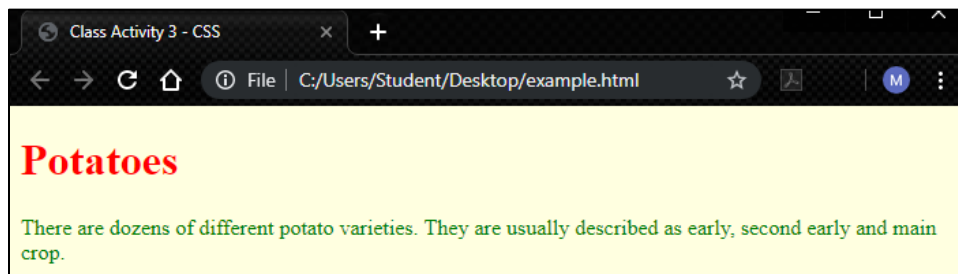
You can also include CSS rules within an HTML page by placing them inside a **<style>** element, which usually sits inside the <head> element of the page.

The **<style>** element should use the type attribute to indicate that the styles are specified in CSS. The value should be text/ css.

When building a site with more than one page, you should use an external CSS style sheet. This:

- Allows all pages to use the same style rules (rather than repeating them in each page).
- Keeps the content separate from how the page looks.
- Means you can change the styles used across all pages by altering just one file (rather than each individual page).

Exercise) Create the following text with font color.



style.css

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Class Activity 3 - CSS</title>
    <style type="text/css">
      body { background-color: lightyellow;}
      h1 { color: red;}
      p{ color: green;}
    </style>
  </head>
  <body> <h1>Potatoes</h1><p> There are dozens of different potato
varieties. They are usually described as
early, second early and main crop.</p>
  </body>
</html>
```

3.2. Text in CSS

The formatting of your text can have a significant effect on how readable your pages are.

Text Properties

The properties that allow you to control the appearance of text can be split into two groups:

- Those that directly affect the font and its appearance (including the typeface, whether it is regular, bold or italic, and the size of the text)
- Those that would have the same effect on text no matter what font you were using (including the color of text and the spacing between words and letters)

When choosing a typeface, it is important to understand that a browser will usually only display it if it's installed on that user's computer.

Typeface Terminology

SERIF

Serif fonts have extra details on the ends of the main strokes of the letters. These details are known as serifs.

In print, serif fonts were traditionally used for long passages of text because they were considered easier to read.

Examples: Georgia, Times New Roman

im

SANS-SERIF

Sans-serif fonts have straight ends to letters, and therefore have a much cleaner design.

Screens have a lower resolution than print. So, if the text is small, sans-serif fonts can be clearer to read.

Examples: Arial , Verdana, Helvetica

im

MONOSPACE

Every letter in a monospace (or fixed-width) font is the same width. (Non-monospace fonts have different widths.)

Monospace fonts are commonly used for code because they align nicely, making the text easier to follow.

Example: Courier

im

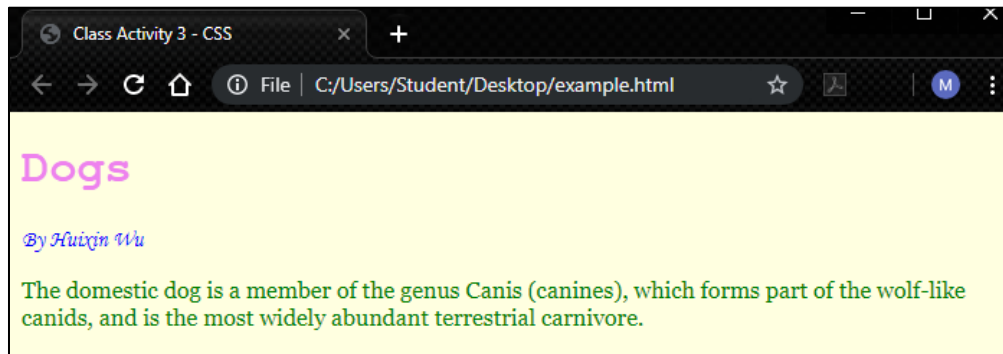
CURSIVE

Cursive fonts either have joining strokes or other cursive characteristics, such as handwriting styles.

Example: *Monotype Corsiva*

im

Exercise) Using the previous html code, create a new paragraph and name it as **author** using **class** property.



```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Class Activity 3 - CSS</title>
    <link href="css/styles.css" type="text/css" rel="stylesheet" />
  </head>
  <body> <h1>Dogs</h1>
    <p class="author">By <i>Huixin Wu</i> </p>
    <p class="intro"> The domestic dog is a member of the genus Canis (canines),
    which forms part of the wolf-like canids, and is the most widely abundant
    terrestrial carnivore.</p>
  </body>
</html>
```

html file

Add new
paragraph
with class
name
"author"

```
body {
background-color: lightyellow;
}
h1 {
color:violet;font-family: courier;
}
.intro{
color: green; font-family: georgia;
}
.author{
color: blue;font-family: "monotype corsiva";
}
```

style.css

Text Characteristics

font-weight

The font weight not only adds emphasis but can also affect the amount of white space and contrast on a page.

Examples:

Light Medium **Bold** Black

font-style

Italic fonts have a cursive aspect to some of the lettering. Oblique font styles take the normal style and put it on an angle

Examples:

Normal *Italic* *Oblique*

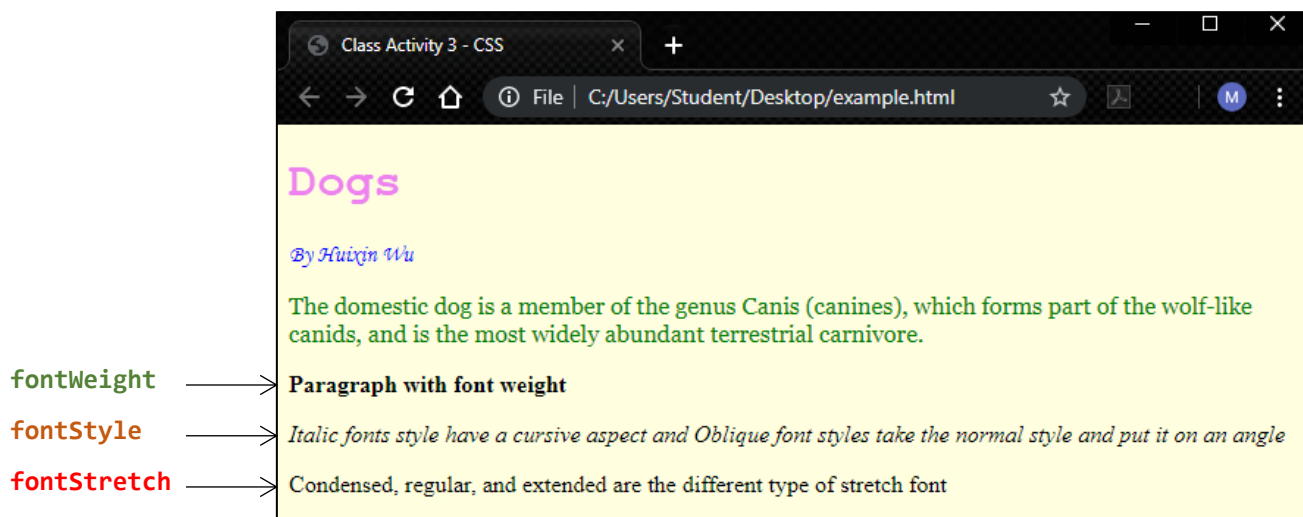
font-stretch

In condensed (or narrow) versions of the font, letters are thinner and closer together. In expanded versions they are thicker and further apart.

Examples:

Condensed Regular EXTENDED

Exercise) Using the previous code, add three more paragraphs with class name **fontWeight**, **fontStyle**, and **fontStretch**



```

<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Class Activity 3 - CSS</title>
  <link href="css/styles.css" type="text/css" rel="stylesheet" />
</head>
<body> <h1>Dogs</h1>
  <p class="author">By <i>Huixin Wu</i></p>
  <p class="intro"> The domestic dog is a member of the genus Canis (canines), which
forms part of the wolf-like canids, and is the most widely abundant terrestrial
carnivore.</p>
  <p class="fontWeight">Paragraph with font weight</p>
  <p class="fontStyle">Italic fonts style have a cursive aspect and Oblique font
styles take the normal style and put it on an angle</p>
  <p class="fontStretch">Condensed, regular, and extended are the different type of
stretch font</p>
</body>
</html>

```

Add the
following
paragraphs

```

.fontWeight{
  font-weight: bold;
}
.fontStyle{
  font-style: oblique;
}
.fontStretch{
  font-stretch: condensed;
}

```

style.css

Units of Text Type Size

The default size of text in a browser is 16 pixels.

Text can be written as pixels, percentages, and ems.

Pixels	Percentages	ems
Setting font size in pixels is the best way to ensure that the type appears at the size you intended (because percentages and ems are more likely to vary if a user has changed the default size of text in their browser)	The default size of text in a web browser is 16 pixels. Using percentages of this amount, you can create a scale where the default text size is 12 pixels, and headings are sized in relation to this.	Ems allow you to change the size of text relative to the size of the text in the parent element. Since the default size of text in web browsers is 16 pixels, you can use similar rules to those shown for percentages.
Pixels	Percentages	Ems (Ephemeral Unit Scalable)

body → 16px

p → 16px

h1 → 32px

h2 → 24px

h3 → 18px

body → 100%

p → 100%

h1 → 200%

h2 → 150%

h3 → 112.5%

body → 100%

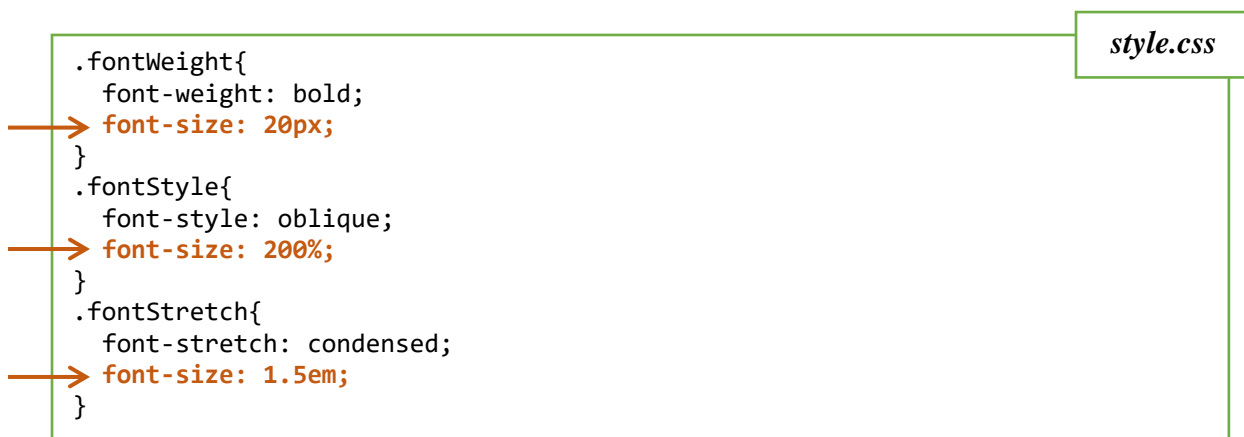
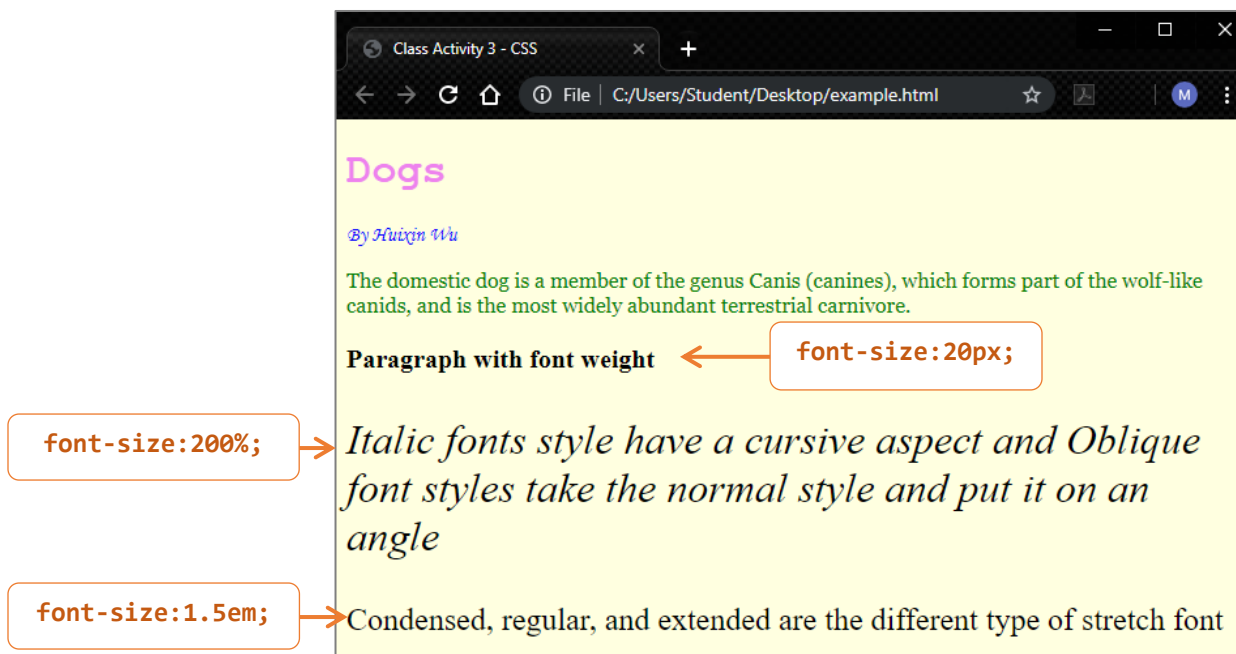
p → 1em

h1 → 2em

h2 → 1.5em

h3 → 1.125em

Example) Using the previous CSS file, use different values of **font-size** to class fontWeight, fontStyle, and fontStretch

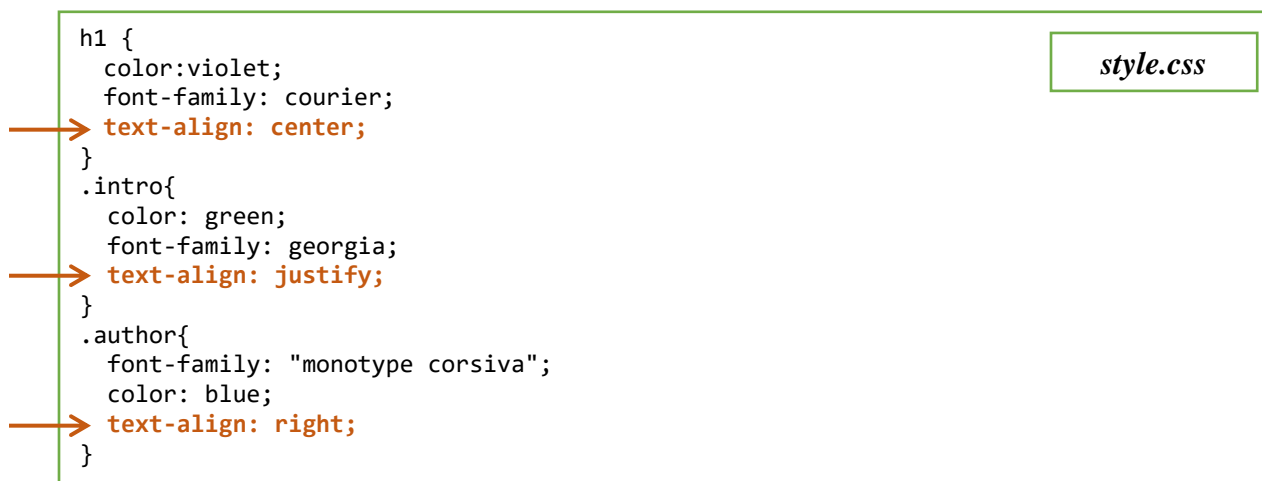
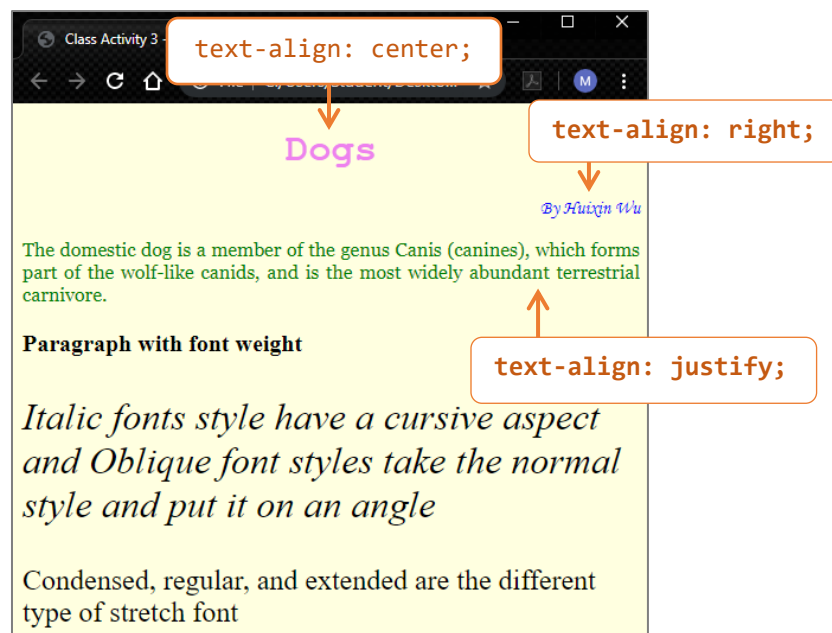


Text Alignment (horizontal)

The **text-align** property allows you to control the alignment of text. The property can take one of four values:

- **left**: This indicates that the text should be left-aligned.
- **right**: This indicates that the text should be right-aligned.
- **center**: This allows you to center text.
- **justify**: This indicates that every line in a paragraph, except the last line, should be set to take up the full width of the containing box.

Example) Using the previous css code, add different **text-align** properties to **h1**, **.intro**, and **.author**



Text-Shadow

The **text-shadow** property is used to create a drop shadow, which is a dark version of the word just behind it and slightly offset. It can also be used to create an embossed effect by adding a shadow that is slightly lighter than the text.

text-shadow can use three lengths and a color for the drop shadow.

The first length indicates how far to the left or right the shadow should fall.

The second value indicates the distance to the top or bottom that the shadow should fall.

The third value is optional and specifies the amount of blur that should be applied to the drop shadow.

The fourth value is the color of the drop shadow.

```
text-shadow: 5px 10px 20px black;
```

Example) Create different **text-shadow** effect.

Text Shadow

html

```
<h1 class="title1">Text Shadow</h1>
```

style.css

```
.title1{  
  text-shadow: 10px 0px 0px black;  
}
```

Text Shadow

html

```
<h1 class="title4">Text Shadow</h1>
```

style.css

```
.title4{  
  text-shadow: 5px 10px 5px black;  
}
```

Text Shadow

html

```
<h1 class="title2">Text Shadow</h1>
```

style.css

```
.title2{  
  text-shadow: 0px 10px 0px black;  
}
```

Text Shadow

html

```
<h1 class="title5">Text Shadow</h1>
```

style.css

```
.title5{  
  text-shadow: 5px 10px 30px black;  
}
```

Text Shadow

html

```
<h1 class="title3">Text Shadow</h1>
```

style.css

```
.title3{  
  text-shadow: 5px 10px 0px black;  
}
```

Text Border, Margin & Padding

Every box has three available properties that can be adjusted to control its appearance:

border

Every box has a border (even if it is not visible or is specified to be 0 pixels wide). The border separates the edge of one box from another.

Border Style

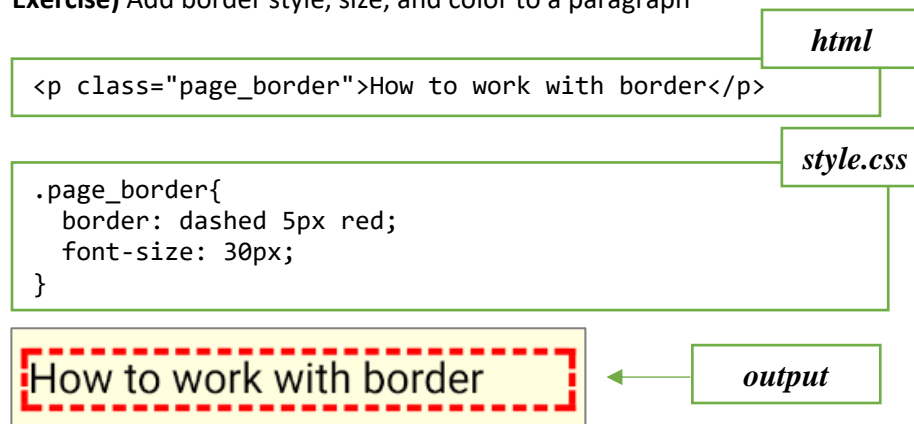
You can control the style of a border using the border-style property. This property can take the following values:

- **solid** a single solid line.
- **dotted** a series of square dots (if your border is 2px wide, then the dots are 2px squared with a 2px gap between each dot)
- **dashed** a series of short lines
- **double** two solid lines (the value of the border-width property creates the sum of the two lines)
- **groove** appears to be carved into the page
- **ridge** appears to stick out from the page
- **inset** appears embedded into the page
- **outset** looks like it is coming out of the screen
- **hidden / none** no border is shown

border values can be written in one line: border style, border size, and border color:

border: dashed 5px red;

Exercise) Add border style, size, and color to a paragraph



margin

Margins sit outside the edge of the border. You can set the width of a margin to create a gap between the borders of two adjacent boxes.

The margin property controls the gap between boxes. Its value is commonly given in pixels, although you may also use percentages or ems.

Specify margin

You can specify values for each side of a box using: margin-top, margin-right, margin-bottom, margin-left.

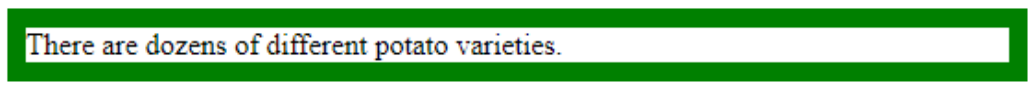
Example) Add right margin to a paragraph

```
<p class="m1">There are dozens of different potato varieties.</p>
```

html

```
.m1{margin-right:100px; border: solid 10px green;}
```

css



Margin with one value

A margin with one value means that the value is added to all sides (top, right, bottom, left) of the element.

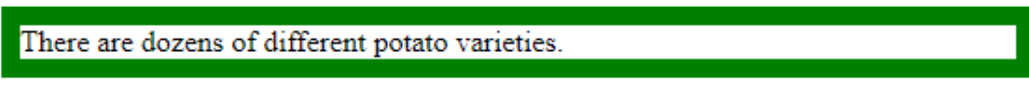
Example) Add the same margin, 50px, to all sides of a paragraph

```
<p class="m2">There are dozens of different potato varieties. </p>
```

html

```
.m2{margin: 50px; border: solid 10px green;}
```

css



Margin with two values

A shorthand to add margin is by given two values. The first value is for the top and bottom margin, and the second value is for the left and right margin.

Example) Add 50px to the top and bottom of a paragraph and 100px to the right and left of the paragraph

```
<p class="m3">There are dozens of different potato varieties. </p>
```

html

```
.m3{margin: 50px 100px; border: solid 10px green;}
```

css

There are dozens of different potato varieties.

Margin with four values

You can also use another shorthand with four values, where the values are in clockwise order: top, right, bottom, left

```
margin: 10px 20px 30px 5px;
```

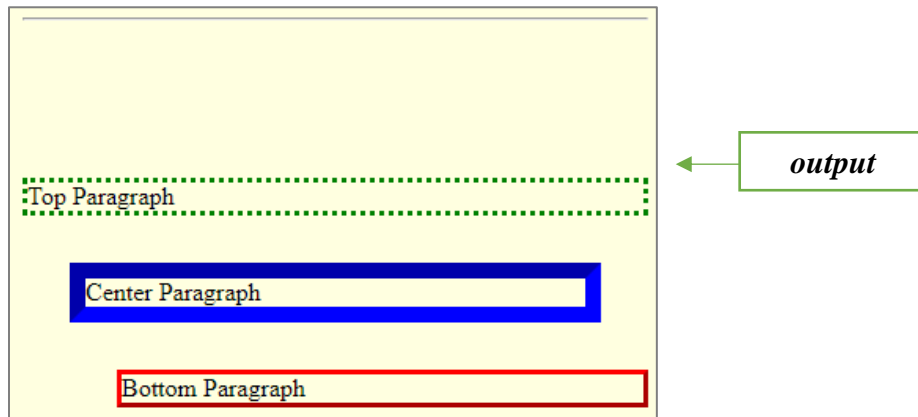
Example) Create three paragraphs and apply different margin and border values to each paragraph

```
<p class="margin1">Top Paragraph </p>
<p class="margin2">Center Paragraph</p>
<p class="margin3">Bottom Paragraph</p>
```

html

style.css

```
.margin1{
  margin-top: 100px;
  border: dotted green 3px;
}
.margin2{
  margin: 30px;
  border:inset blue 10px;
}
.margin3{
  margin-left: 60px;
  border: outset red;
}
```



padding

Padding is the space between the border of a box and any content contained within it. Adding padding can increase the readability of its contents.

The padding property allows you to specify how much space should appear between the content of an element and its border.

The value of this property is most often specified in pixels (although it is also possible to use percentages or ems). If a percentage is used, the padding is a percentage of the browser window (or of the containing box if it is inside another box).

Specify padding

You can specify different values for each side of an element by using padding-top; padding-right; padding-bottom; padding-left.

Example) add 30px of left padding to a paragraph

```
<p class="padding1">There are dozens of different potato varieties.  
They are usually described as early, second and main crop.</p>
```

html

```
.padding1{padding-left: 30px; border: 10px inset orange; }
```

css

There are dozens of different potato varieties. They are usually described as early, second and main crop.

Padding with one value

A padding with one value means that the value is added to all sides (top, right, bottom, left) of the element.

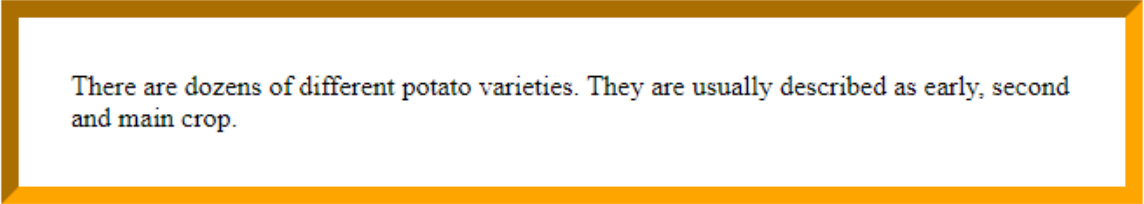
Example) apply 50px of padding to all sides of an element

```
<p class="padding2">There are dozens of different potato varieties.  
They are usually described as early, second and main crop.</p>
```

html

```
.padding2{padding: 30px; border: 10px inset orange; }
```

css



There are dozens of different potato varieties. They are usually described as early, second and main crop.

Padding with two values

A shorthand to add padding is by given two values. The first value is for the top and bottom padding, and the second value is for the left and right padding.

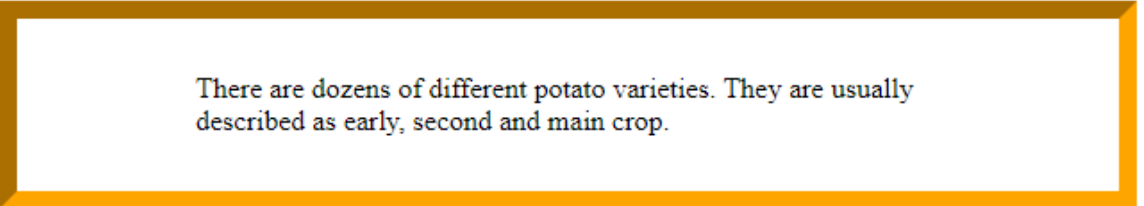
Example) apply 30px of padding to the top and bottom, and 100px to the left and right of an element

```
<p class="padding3">There are dozens of different potato varieties.  
They are usually described as early, second and main crop.</p>
```

html

```
.padding3{padding: 30px 100px; border: 10px inset orange; }
```

css



There are dozens of different potato varieties. They are usually described as early, second and main crop.

Padding with four values

Another shorthand to add padding is by given four values, where the values are in clockwise order: top, right, bottom, left

Example) apply 10px to top padding, 5px to right padding, 30px to bottom padding, and 100px to left padding

```
<p class="padding4">There are dozens of different potato varieties.  
They are usually described as early, second and main crop.</p>
```

html

```
.padding4{padding: 10px 5px 30px 100px ; border: 10px inset orange;}
```

css

There are dozens of different potato varieties. They are usually described as early, second and main crop.

Example) Create three paragraphs and apply three different padding to each of them.

```
<p class="one">Without Padding</p>  
<p class="two">padding to 20px</p>  
<p class="three">padding top by 50px</p>
```

html file

```
.one{ border: ridge green;}  
  
.two{  
    padding:20px;  
    border: solid gray 2px;  
}  
  
.three{  
    padding-top: 50px;  
    border: dotted black;  
}
```

style.css

Without Padding

padding to 20px

padding top by 50px

output

2.3. CSS Layout Properties and images

2.3.1. Basic CSS Layout Design

In this chapter we are going to look at how to control where each element sits on a page and how to create attractive page layouts.

Screen size

Different visitors to your site will have different sized screens that show different amounts of information, so your design needs to be able to work on a range of different sized screens.



Dell XPS 13 Laptop
Display: **13.3 inches across screen**
Display resolution: **1920 × 1080 pixels**



Apple MacBook
Display: **12 inches across screen**
Display resolution: **2304 × 1440 pixels**

Building blocks

CSS treats each HTML element as if it is in its own box. This box will either be a block-level box or an inline box.

If one block-level element sits inside another block-level element then the outer box is known as the containing or parent element.

It is common to group a number of elements together inside a `<div>` (or other block-level) element.

Static Layouts: Fixed Width

Fixed width layout designs do not change size as the user increases or decreases the size of their app window. Measurements tend to be given in pixels.

Advantages

- Pixel values are accurate at controlling size and positioning of elements.

- The designer has far greater control over the appearance and position of items on the page than with liquid layouts.
- You can control the lengths of lines of text regardless of the size of the user's window.
- The size of an image will always remain the same relative to the rest of the page.

Disadvantages

You can end up with big gaps around the edge of a page.

- If the user's screen is a much higher resolution than the designer's screen, the page can look smaller and text can be harder to read.
- If a user increases font sizes, text might not fit into the allotted spaces.
- The design works best on devices that have a size or resolution similar to that of desktop or laptop computers.
- The page will often take up more vertical space than a liquid layout with the same content.

Dynamic Layouts: Liquid Layout

The liquid layout uses percentages to specify the width of each box so that the design will stretch to fit the size of the screen.

Advantages

- Pages expand to fill the entire app window so there are no spaces around the page on a large screen.
- If the user has a small window, the page can contract to fit it without the user having to scroll to the side.
- The design is tolerant of users setting font sizes larger than the designer intended (because the page can stretch).

Disadvantages

- If you do not control the width of sections of the page then the design can look very different than you intended, with unexpected gaps around certain elements or items squashed together.
- If the user has a wide app window, lines of text can become very long, which makes them harder to read.
- If the user has a very narrow app window, words may be squashed and you can end up with few words on each line.
- If a fixed width item (such as an image) is in a box that is too small to hold it (because the user has made the window smaller) the image can overflow over the text.

Elements use to design layouts

There are multiple ways and different types of elements and properties that we can design a static or dynamic layout. We can look at the different types of elements first and then the different CSS properties that can help us to design a static and dynamic layout.

CSS elements in the design process

CSS offers a variety of elements that we can use to design a webpage layout. For example, earlier years of internet, web designer used non-semantic elements such as `<div>`, a block element, and ``, an inline element. As internet technology advances, now webpages are designed using semantic elements, which are elements that describe their meaning to both the browser and the developer. Examples of semantic elements are `<header>`, `<nav>`, `<section>`, `<footer>`, etc

Floating Elements. float property in CSS

The **float** property allows you to take an element in normal flow and place it as far to the left or right of the containing element as possible.

Anything else that sits inside the containing element will flow around the element that is floated.

When you use the float property, you should also use the width property to indicate how wide the floated element should be. If you do not, results can be inconsistent but the box is likely to take up the full width of the containing element (just like it would in normal flow).

The CSS **float** property specifies how an element should float. The **float** property can have one of the following values:

- **left** - The element floats to the left of its container
- **right** - The element floats to the right of its container
- **none** - The element does not float (will be displayed just where it occurs in the text). This is default
- **inherit** - The element inherits the float value of its parent

box-sizing property

The **box-sizing** property can make building CSS layouts easier and a lot more intuitive. **box-sizing: border-box;**, we can change the **box** model to what was once the "quirky" way, where an element's specified width and height aren't affected by padding or borders.

Example) Create a layout using to display a title and four images: a title at the first row and two images at the second and third row.



To create the layout, first we create a main container to hold the header and the four images. We can create this container using `<section>` element and assign a class name container

```
<section class="container">  
  
</section>
```

Once the container is set, within the `<section>` we are going to use a `<header>` element for the title and two `<figure>` containers, each `<figure>` element will hold two images.

```
<section class="container">  
  <header class="header">NEW YORK CITY</header>  
  <figure class="images"><img class="image1"><img class="image2"></figure>  
  <figure class="images"><img class="image3"><img class="image4"></figure>  
</section>
```

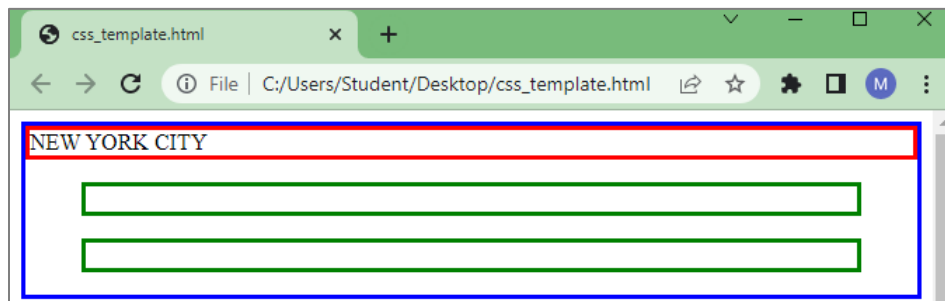
A complete HTML file looks as the following:

html file

```
<!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>Layout Design</title>
  </head>
  <body>
    <section class="container">
      <header class="header">NEW YORK CITY</header>
      <figure class="images"><img class="img1"><img class="img2"></figure>
      <figure class="images"><img class="img3"><img class="img4"></figure>
    </section>
  </body>
</html>
```

We can add CSS border to each element to see the layout:

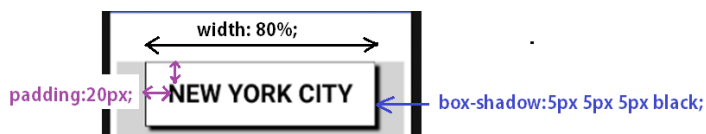
```
.container{border: solid blue;}
.header{border: solid red;}
.images{border:solid green;}
```



Now in the CSS file, we can set a background-color, width, and height to **.container**:

```
.container{
  background-color: lightgray;
  width: 100%;
  height: auto;
  padding-bottom: 5%;
}
```

After it, we can set the CSS to the **.header** with a 80% of width, text padding of 20px for all sides, center the title element, background-color to white, font size set to 30px, bolder text, and a box-shadow:

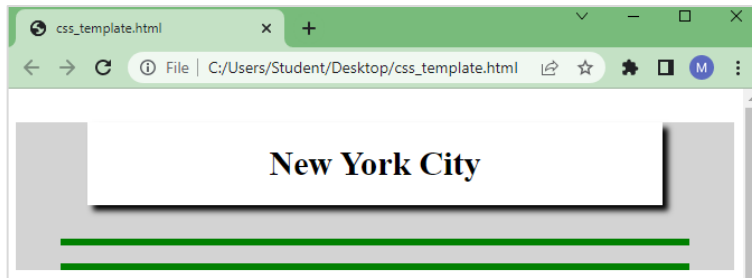


```
.header{
  width: 80%;
  margin: auto;
  box-shadow: 5px 5px 5px black;
  padding: 20px;
  font-size: 30px;
  font-weight: 600;
  text-align: center;
  background-color: white;
}
```

Once the header is set, we can set the CSS to the **.images** container. For this, we can add a border to the images container to use it as reference. This border can be erased after the layout is set. We also can set the height of **.images** element to *auto*, which will automatically adjust to the height of the picture:

```
.images{
  border:solid green;
  height: auto;
}
```

The layout looks as following:



The complete CSS file looks as the following:

```
*{ box-sizing: border-box; }
.container{
  background-color: lightgray;
  width: 100%;
  height: auto;
  padding-bottom: 5%;
}
.header{
  width: 80%;
  margin: 1em auto;
  box-shadow: 5px 5px 5px black;
  padding: 20px;
  font-size: 30px;
  font-weight: 600;
  text-align: center;
  background-color: white;
}
.images{
  border: solid green;
  height: auto;
}
```

index.css

4.3.2 CSS styling images

Continuing working on the previous example, the images can fit two images in a **<figure>** container. If we are applying two images in one **<figure>** element, we can set the width of the each image to 40% and the height to fit 100% of the **<figure>** element.

Now, let us Insert four different images to the previous code using **<figure>** element.

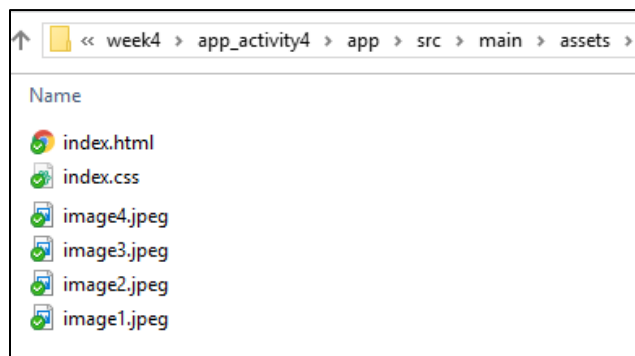
We can download four images and save them in assets folder. If we want to organize our assets folder, we can create a new folder, which we can name it *images*, inside our project folder place all the images:



If the images are inside of the *images* folder, the code in HTML should indicate the location of the folder as the following:

```
<figure class="images">
  
  
</figure>
```

If the images are in the same folder that the HTML file:



The code in HTML should just indicate the image name:

```
<figure class="images">
  
  
</figure>
```

For this activity, we are going to organize all images inside the *images* folder. The complete HTML file

```
<!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>Layout Design</title>
  </head>
  <body>
    <div class="container">
      <header class="header">NEW YORK CITY</header>
      <figure class="images">
        
        
      </figure>
      <figure class="images">
        
        
      </figure>
    </div>
  </body>
</html>
```

html file

Now in CSS, for the images to sit within the `<figure>` element, we have to write a CSS code that specifies the **width** of the images to **40%** with respect to the width of the `<figure>` element, and the **height** to fill automatically the height of the `<figure>` container. We can also add 5% of margin to the left and right of the image:

```
.images img{
  width: 40%;
  height: auto;
  margin-left:5%;
  margin-right:5%;
}
```

The complete CSS file looks as the following:

```
*{ box-sizing: border-box; }
.container{
  background-color: lightgray; width: 100%; height: auto;
}
.header{
  width: 80%; margin: 1em auto; box-shadow: 5px 5px 5px black; padding: 20px;
  font-size: 30px; font-weight: 600; text-align: center; background-color: white;
}
.images{
  border:solid green; height: auto;
}
.images img{
  width: 40%; height: auto; margin-left: 5%; margin-right: 5%;
}
```

index.css

The images should look as the following:

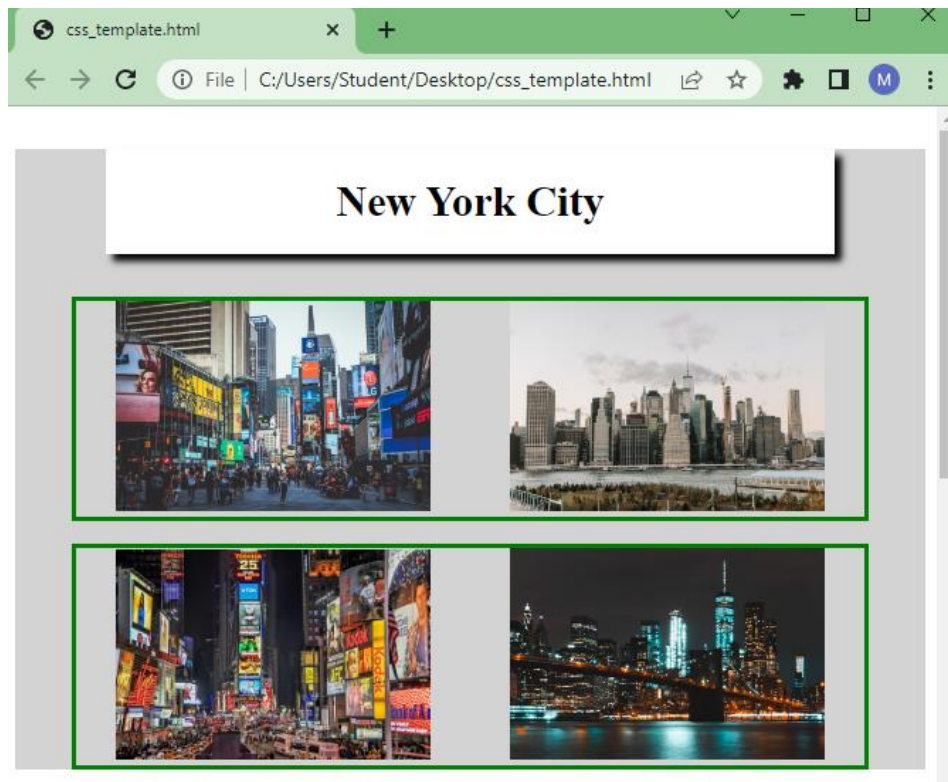


Image format

To create rounded and circled images, we can use the `border-radius` property.

Rounded Images



```
img1 {  
  border-radius: 60px;  
}
```

Circled or oval image



```
img2 {  
  border-radius: 50%;  
}
```

If the height and width is the same values, applying `border-radius: 50%;` results in a circled shape. Otherwise, if the height and width has different values, applying `border-radius: 50%;` results in an oval shape

To create a border or shadow to an image, we can use the `border` property to create thumbnail images.

Border image



```
img3 {  
  border: 1px solid green;  
  border-radius: 10px;  
  padding: 20px;  
}
```

Shadow image



```
img4 {  
  box-shadow: 0 0 5px 20px lightblue;  
}
```

Example) Using the previous code, add CSS code to round or oval the first image, make the second image to have rounded borders, add border with padding to the third image, and shadow the fourth image.

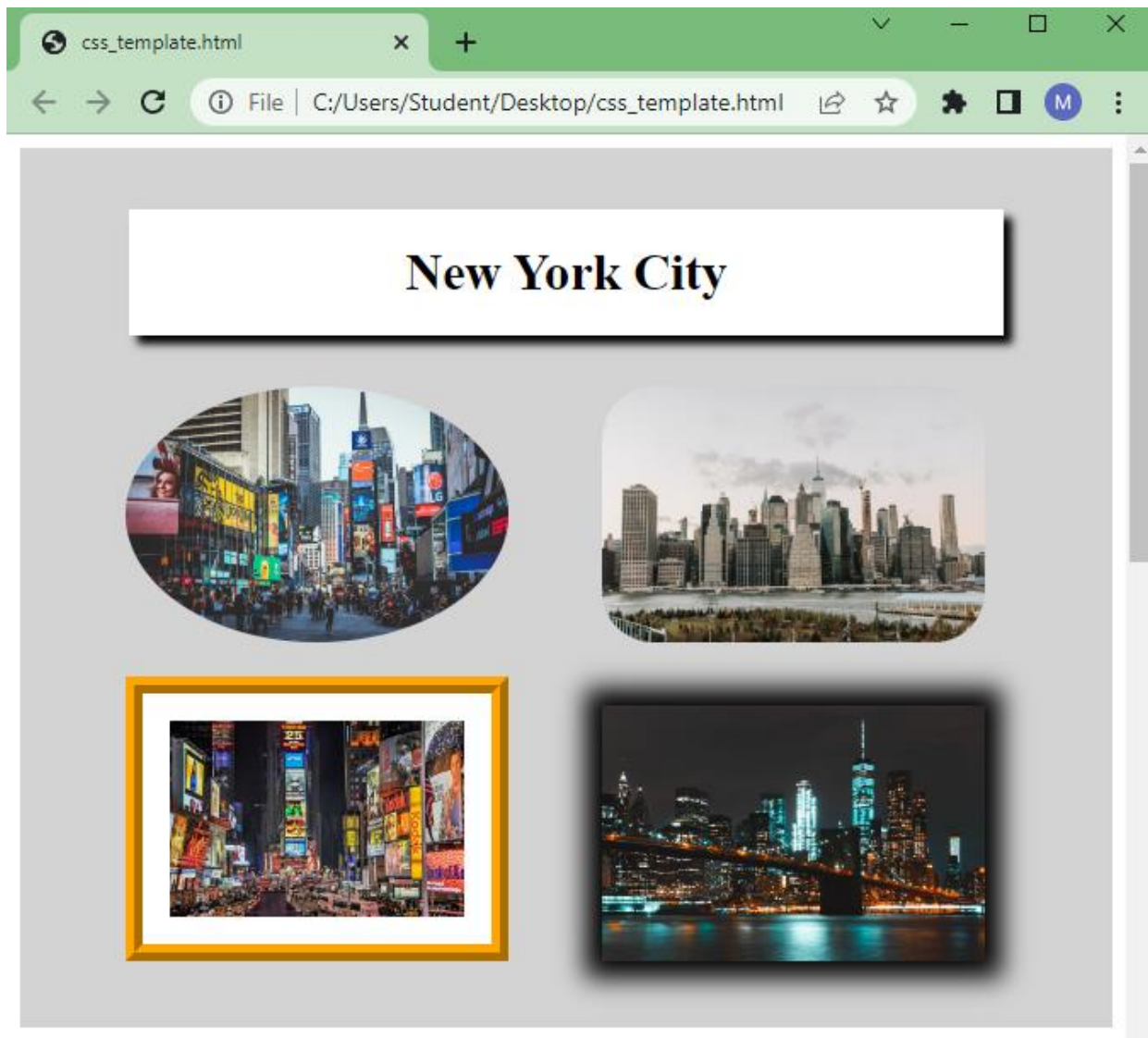
To complete this activity, we can add the css code within the HTML code using `style` as the following:

```
<figure class="images">  
    
    
</figure>
```

If we want to complete this activity using external css code, we can add a class name to each image and then add the css code in the css file. For example, if you want to complete the CSS for the third and fourth images:

```
.img3{  
  border: ridge 10px orange;  
  padding: 1em;  
  background-color: white;  
}  
.img4{box-shadow: 0 0 15px 10px black;}
```

The layout will look as:



Example) Create a layout to display six social media logo

```
img{width: 30%; margin:calc(10%/6);
```

3.4. Navigation Bars

A navigation bar needs standard HTML as a base, which can be built using the `` and `` elements or `<nav>` element. To create a navigation bar using `` and ``

3.4.1. Vertical Navigation Bars

To create a basic vertical navigation bar using ``, we can list each hyperlink using `<a>` tag as the following:

```
<ul class="navbar">
  <li><a href="#home">Home</a></li>
  <li><a href="#nav">Tabs</a></li>
  <li><a href="#flex">Flex-Wrap</a></li>
  <li><a href="#social">Social Media</a></li>
</ul>
```

To take off the bullet of each list, we can set the **list-style-type** to **none** of the ``:

```
.navbar{
  list-style-type: none;
}
```

To make the vertical navigation bar with a purple background-color, we can write CSS codes to make changes to the hyperlinks in the list item. Also, **display: block;** property sets the `<a>` elements in a block, column, format because `<a>` element by default is an inline element

```
.navbar li a {
  display: block;
  padding: 8px 16px;
  color: white;
  background-color: purple;
  width: 50%;
  text-decoration: none;
}
```

After it, we can change the background color of the links when the user clicks on them:

```
.navbar li a:hover {
  background-color: gray;
  color: white;
}
```

The HTML and CSS code looks as the following:

```
<ul class="navbar">
  <li><a href="#home">Home</a></li>
  <li><a href="#nav">Tabs</a></li>
  <li><a href="#flex">Flex-Wrap</a></li>
  <li><a href="#social">Social Media</a></li>
</ul>
```

html file

```
/* -- navigation bars --*/
h2{
  width: 90%;
  text-align: center;
  margin-top: 70px;
}
.navbar{
  list-style-type: none;
}
.navbar li a {
  display: block;
  padding: 8px 16px;
  color: white;
  background-color: purple;
  width: 50%;
  text-decoration: none;
}
/* Change the link color when is active*/
.navbar li a:hover {
  background-color: gray;
  color: white;
}
```

css file

The output will look as the following:

Vertical Navigation Tabs using ul



3.4.2. Horizontal Navigation Bars

To create a horizontal navigation bars, we can use inline element such as **<nav>** element and have <a> elements within it.

```
<nav class="nav_div">
  <a href="#home">Fantasy Shoes</a>
  <a href="#products">Season's Shoes</a>
  <a href="#sale">Winter Sale</a>
  <a href="#social">Follow Us</a>
</nav>
```

At the browser, the navigation will look as:



Once we have the HTML code set, in CSS, we can set the hyperlink container with background-color, the width, the height, and the font-size:

```
.nav_div{
  background-color: darkgray;
  width: 100%;
  font-size: 1.5em;
}
```

After it, we can set hyperlink with left border, height, width, font color, center the text, and add the padding. We also set the hyperlink to float from the left to right without underline:

```
.nav_div a {
  display:inline-block;
  color: white;
  background-color: darkgray;
  text-align: center;
  padding: 10px 20px;
  text-decoration: none
}
```

After it, we can change the background color of the links when the user hover on them:

```
.nav_div a:hover {
  background-color: orange;
}
```

The HTML and CSS code looks as the following:

```
<nav class="nav_div">
  <a href="#home">Fantasy Shoes</a>
  <a href="#products">Season's Shoes</a>
  <a href="#sale">Winter Sale</a>
  <a href="#social">Follow Us</a>
</nav>
```

html code

```
.nav_div{
  background-color: darkgray;
  width: 100%; /* width can be replaced with display: block; */
  font-size: 1.5em;
}
.nav_div a {
  display:inline-block;
  color: white;
  background-color: darkgray;
  text-align: center;
  padding: 10px 20px;
  text-decoration: none
}
.nav_div a:hover {
  background-color: orange;
}
```

css code

The output will look as the following:



Fantasy Shoes Season's Shoes Winter Sale Follow Us

3.5. Color and images manipulation in CSS

3.5.1. Colors

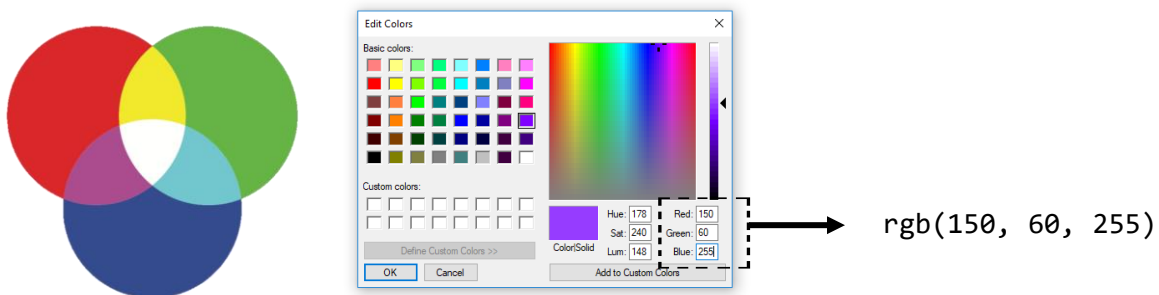
The color property allows us to specify the color of text inside an element. We can specify any color in CSS in one of three ways:

Color Names

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

RGB Values

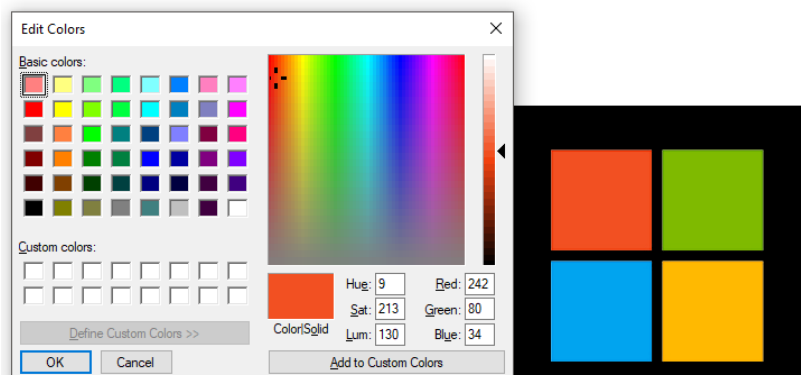
These express colors in terms of how much red, green and blue are used to make it up. Every color on a computer screen is created by mixing amounts of red, green, and blue. To find the color you want, we can use a color picker.



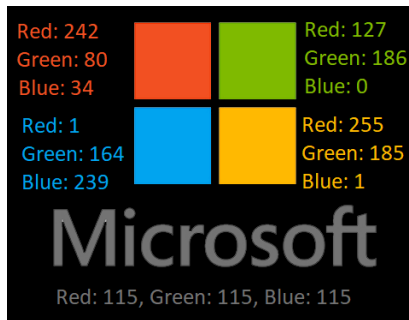
Example) find each RGB color the Microsoft logo:



To find a specific color, we can use a color picker tool from any graphic app. One simple graphic app that comes with Windows is *Paint*.



Using Paint, we can the four colors of the Microsoft logo as:



Using the RGB in CSS will look as following:

```
<section class="rgb">
  <div class="red"></div>
  <div class="green"></div>
  <div class="blue"></div>
  <div class="orange"></div>
  <div class="logo">Microsoft</div>
</section>
```

HTML code

```
.rgb{margin: 2em; display: inline-block;}
.red,.green,.blue,.orange{
  width:100px;
  height:100px;
  margin: 20px;
  float: left;
}
.red{background-color:rgb(242,80,34)}
.green{background-color:rgb(127,186,0)}
.blue{background-color:rgb(1,164,239)}
.orange{background-color:rgb(255,185,1)}
.logo{
  color:rgb(115,115,115);
  font-size:5em;
  font-family:"Arial Black"
}
```

CSS code

The display in the browser will be as:



Microsoft

rgba() color

The `rgba()` function defines 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 transparent (0% of opacity) and 1 means fully opaque (100% of opacity). The syntax of the code is:

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

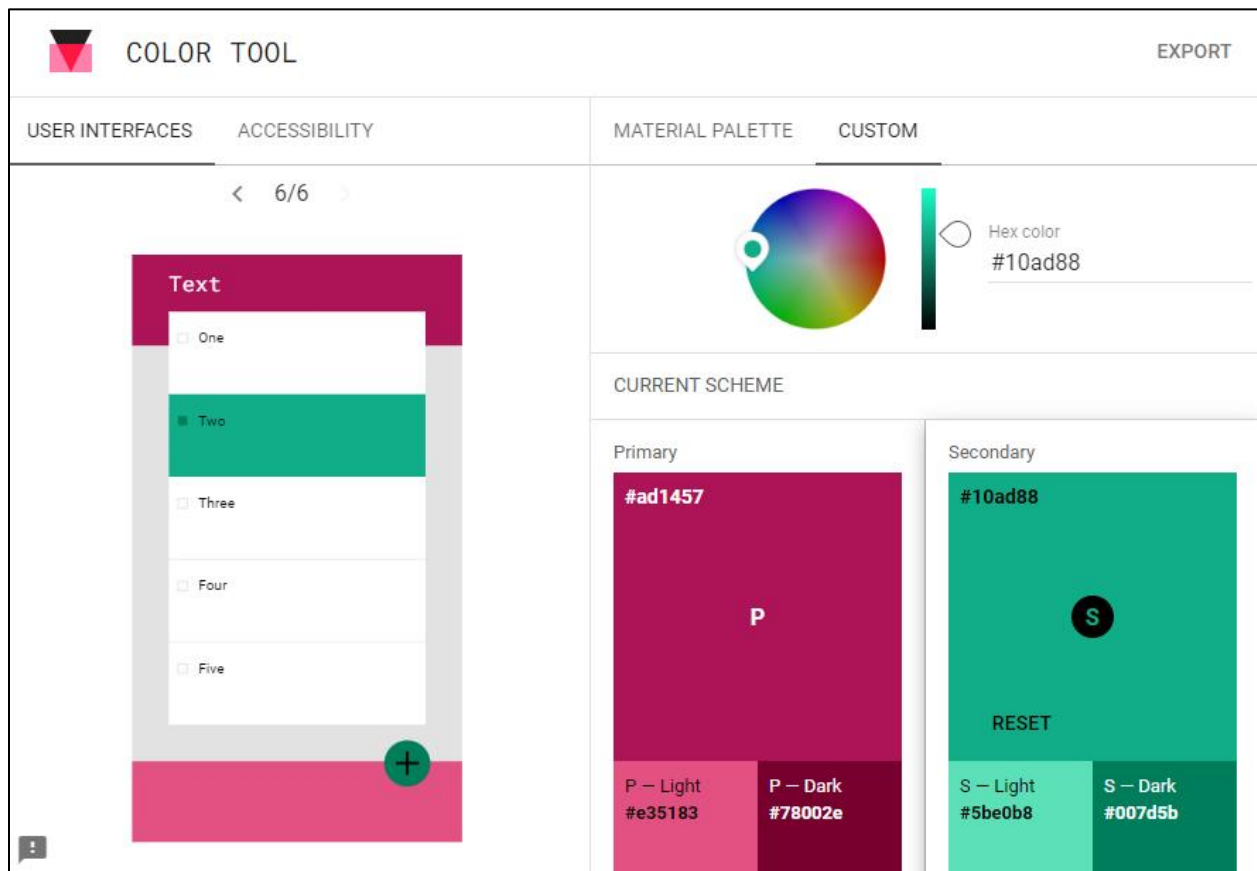
For example, for color `rgb(150, 60, 255)` with 35% of opacity, we can write the code as:

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

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`

We can find different HEX color picker tools online. One of the websites to explore the HEX color picker as a primary or secondary color is *material.io*. *material.io* is a website dedicated to the UI design of smartphone. Some visitors of *material.io* like to use its *Color Tools* to explore the combination of a primary and secondary color. <https://m2.material.io/resources/color/#!/?view.left=0&view.right=0>



Position Property

The position CSS property sets how an element is positioned in a document. For position: relative, absolute, or sticky, the top, right, bottom, and left properties determines the final location of positioned elements.

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`.



Exercise) Create container with a back image with `position:relative;` and a message with `position:absolute;`

We can create a container for the back image using `<div>` 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="image">
      
    </div>
  </body>
</html>
```

CSS

```
img{ width: 100%; height: 100%;}
.image{
  width: 80%;
  height: 450px;
  margin: auto;
  position: relative;
}
```

Position relative and absolute



After it, we can add the text message using another `<div>` or `<p>` and use `position: absolute;`

```
<div class="message"> Creating Smartphone Apps</div>
```

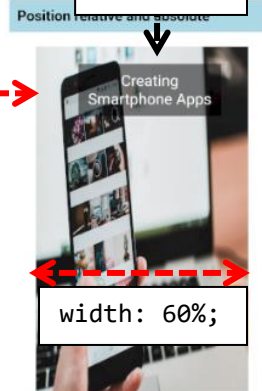
HTML

```
.message{  
→ position: absolute;  
  top: 10%;  
  left: 20%;  
  width: 60%;  
  background-color: rgba(0, 0, 0, 0.3);  
  color: white;  
  text-align: center;  
  padding: 10px;  
  font-size: 20px;  
}
```

CSS

left:20%;

top:10%;



position: sticky;

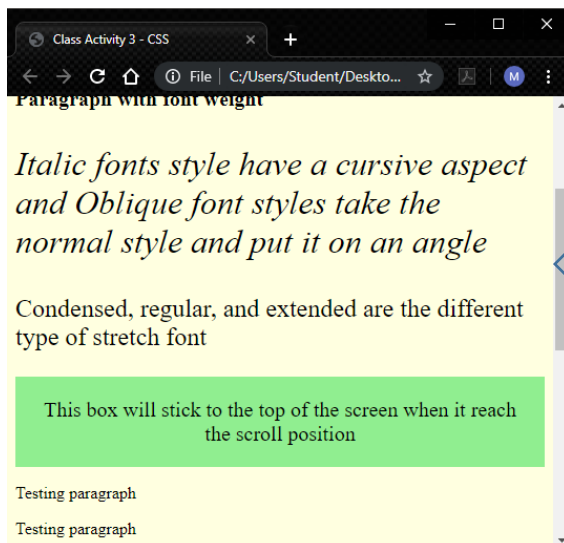
An element with `position: sticky;` is positioned based on the user's scroll position.

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`).

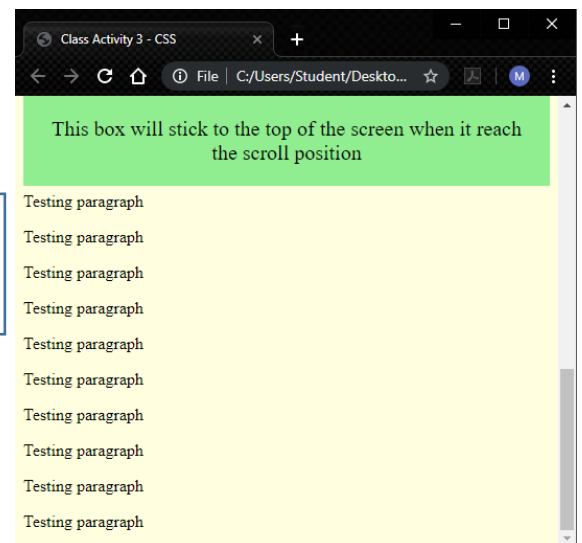
Exercise) Create a message box using `position: sticky;`

We can create a container for the sticky message using `<div>` 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;`

We can also add 16 testing paragraphs to see the sticky effect.



Scroll the
app page
to the top



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>
```

CSS

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

Fixed navigation bar

Fixed navigation bar happens when the navigation bar remain at the top or bottom of the app, even when the user moves the page. For this, you can use the CSS property `position: sticky;` to the element that contains the hyperlinks:

```
position: fixed;
top: 0;
```

`position:fixed;` and `top: 0;` will make the element to have a position fixed to the top of the app.

Exercise) Create a navigation bar that will stick on the top of the app.

We can create the hyperlink using `<div>`

```
<nav class="nav_top">
  <a href="#home">Home</a>
  <a href="#nav">Tabs</a>
  <a href="#flex">Flex-Wrap</a>
  <a href="#social">Social Media</a>
</nav>
```

Once we have the HTML code set, in CSS, we can set the hyperlink container with background-color, the width, the height, the font-size, and the position:

```
.nav_top{
  background-color: orange;
  font-size: 1.2em;
  height: 3em;
  position: sticky;
  top: 0;
}
```

After it, we can set hyperlink with left border, height, font color, center the text, and add the padding. We also set the hyperlink to float from the left to right without underline:

```
.nav_top a {
  float: left;
  color: white;
  text-align: center;
  text-decoration: none;
  padding: 1em 2em;
}
```

After it, we can change the background color of the links when the user clicks on them:

```
.nav_top a:hover{
  background-color: olive;
  font-weight: bold;
}
```

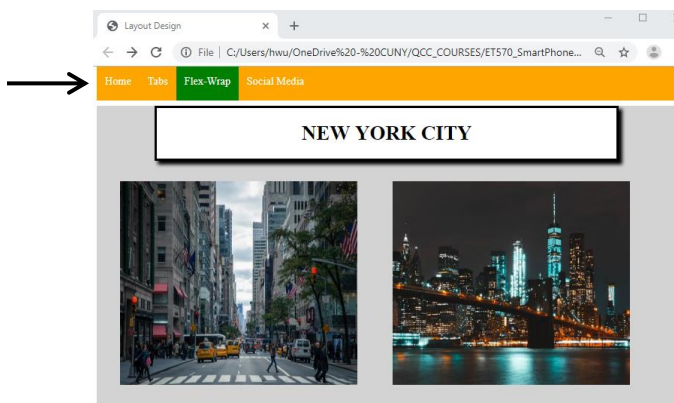
The HTML and CSS code looks as the following:

```
<nav class="nav_top">
  <a href="#home">Home</a>
  <a href="#nav">Tabs</a>
  <a href="#flex">Flex-Wrap</a>
  <a href="#social">Social Media</a>
</nav>
```

html file

```
.nav_top{
  background-color: orange;
  font-size: 1.2em;
  height: 3em;
  position: sticky;
  top: 0;
}
.nav_top a {
  float: left;
  color: white;
  text-align: center;
  text-decoration: none;
  padding: 1em 2em;
}
.nav_top a:hover{
  background-color: olive;
  font-weight: bold;
}
```

css file



If you need the navigation bar stick in the bottom, you can change **top: 0;** to **bottom: 0;**

Display properties

The **display** property specifies the display behavior (the type of rendering box) of an element.

In HTML, the default display property value is taken from the HTML specifications or from the browser/user default style sheet. The default value in XML is inline, including SVG elements.

Note: The values "flex" and "inline-flex" requires the -webkit- prefix to work in Safari.

Note: "display: contents" does not work in Edge prior version 79.

Some of the display properties are:

inline	Displays an element as an inline element (like). Any height and width properties will have no effect
block	Displays an element as a block element (like <p>). It starts on a new line, and takes up the whole width
contents	Makes the container disappear, making the child elements children of the element the next level up in the DOM
flex	Displays an element as a block-level flex container
grid	Displays an element as a block-level grid container
inline-block	Displays an element as an inline-level block container. The element itself is formatted as an inline element, but you can apply height and width values
inline-flex	Displays an element as an inline-level flex container
inline-grid	Displays an element as an inline-level grid container
inline-table	The element is displayed as an inline-level table
list-item	Let the element behave like a element

Background properties

Background images and 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 grading space. For example, we can create a **linear-gradient** of the red and blue color from top to bottom as the following:

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

HTML

```
.gradient1 {  
  height: 200px;  
  background-image: linear-gradient(red, blue);  
}
```

CSS



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, blue);
```

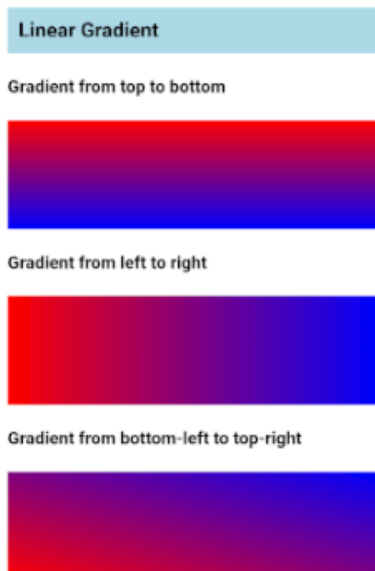


We can also set the linear-gradient from bottom-left to top-right:

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



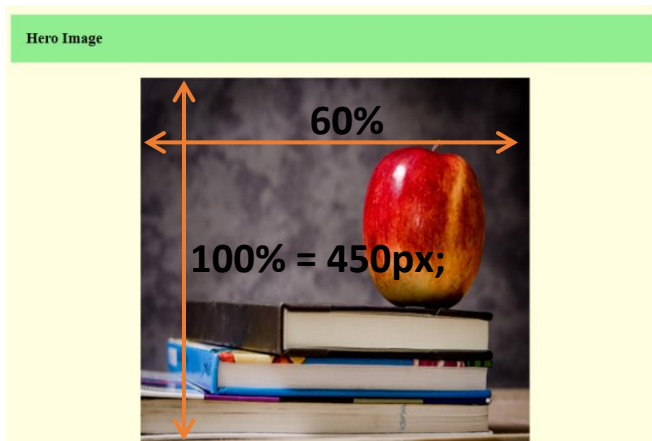
The output will look as the following:



Hero image

A hero image is a large image with text places on the top of it. For this, we can use the position absolute and relative.

Exercise) Create an ad of an image with a text places on top of it as the following:



First, we create a container with a background image:

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

</div>
```

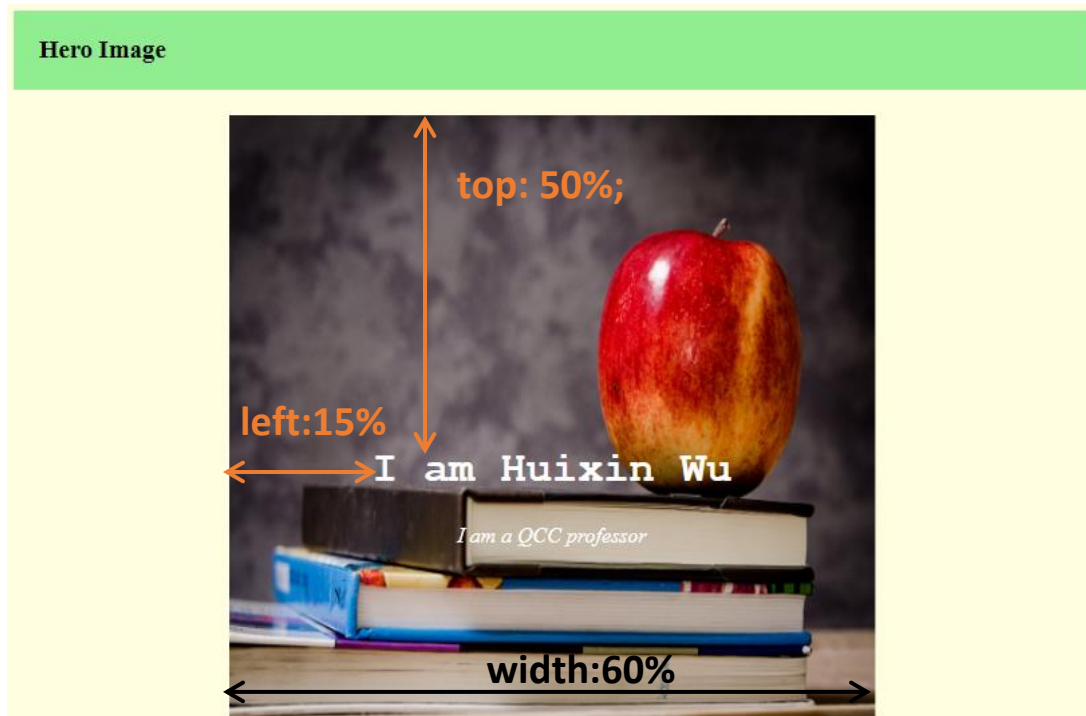
HTML

We can insert a background image from a CSS using **background-image** property. Also, we can align the background image using **background-position: center;** and adjust the width and height using **background-size: 60% 100%;** which 60% is the width and 100% is the height.

```
.heroImage{
  background-image: url("apple.jpg");
  background-repeat: no-repeat;
  background-position: center;
  background-size: 60% 100%;
  height: 450px;
  position: relative;
}
```

CSS

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**



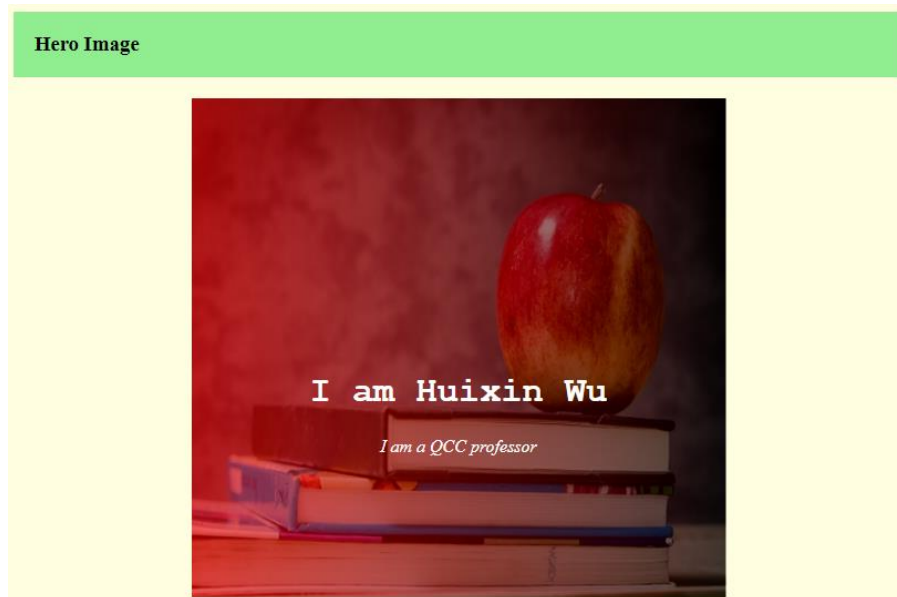
```
<h3>Hero Image </h3>
<div class="heroImage">
  <div class="heroText">
    <h1>I am Huixin Wu</h1>
    <i>I am a QCC professor</i>
  </div>
</div>
```

HTML

```
.heroText{
  position: absolute;
  top: 50%;
  left: 15%;
  width: 70%;
  color: white;
  text-align: center;
}
```

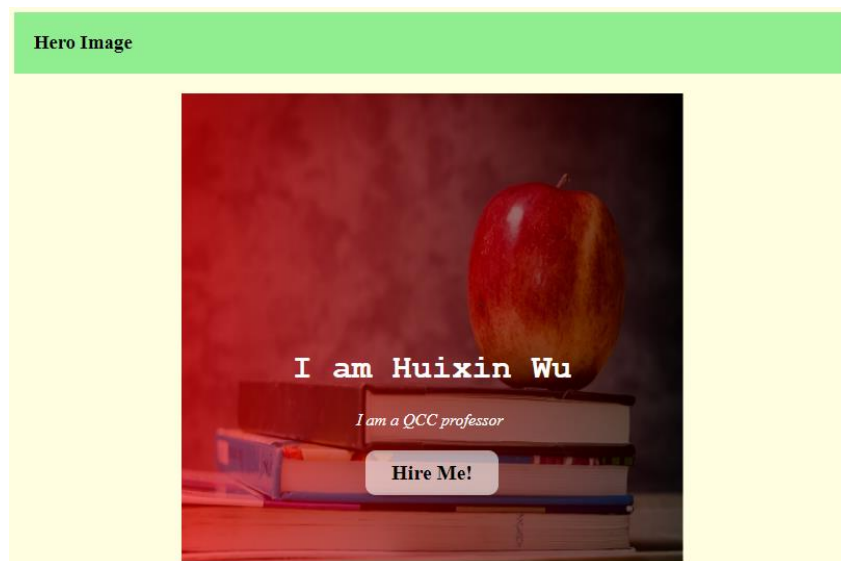
CSS

A text on top of an image makes the text information very hard to write, for this, we can add linear-gradient to the background image. We can set a red to black gradient with 60% of opacity from left to right:



CSS

```
.heroImage{  
  background-image: linear-gradient(to right, rgba(255,0,0,0.6),rgba(0,0,0,0.6)),  
  url("apple.jpg");  
  background-position: center;  
  background-repeat: no-repeat;  
  background-size: 100% 100%;  
  height: 450px;  
  position: relative;
```



HTML

```
<h3>Hero Image </h3>
<div class="heroImage">
  <div class="heroText">
    <h1>I am Huixin Wu</h1>
    <i>I am a QCC professor</i>
    <a href="#" class="submit">Hire Me!</a>
  </div>
</div>
```

CSS

```
.submit{
  display:block;
  padding: 10px 25px;
  color: black;
  background-color: rgba(255,255,255,0.6);
  font-weight: 600;
  font-size: 20px;
  margin-top: 20px;
  text-align: center;
  border-radius: 10px;
  text-decoration: none;}
}
```

z-index property

The z-index property specifies the stack order of an element.

An element with greater stack order is always in front of an element with a lower stack order.

Note: z-index only works on positioned elements (position: absolute, position: relative, position: fixed, or position: sticky) and flex items (elements that are direct children of display:flex elements).

Note: If two positioned elements overlap without a z-index specified, the element positioned last in the HTML code will be shown on top.

Example) Use z-index on two overlap images and one text

To do so, we first use a <figure> element to hold two images and one paragraph

HTML

```
<figure>
  
  
  <p class="nyc_txt">New York City</p>
</figure>
```

Once we have the HTML layout, in the CSS file, we can use z-index to set the order of how the images and paragraph display on the <figure> element. For example, we can set the first image, with class="nyc1", to the back of the other image in the paragraph using z-index: -1. After it, the second image can have z-index:0 and the paragraph with z-index: 1.

CSS

```
figure{width: 500px; height: 400px; position: relative;}
.nyc1{ width:100%;position: absolute; z-index: 0;}
.nyc2{ width:50%;position: absolute; border: ridge white 1em; z-index:1; }
.nyc_txt{font-size: 4em; position: absolute; z-index:2; }
```

The output in the browser looks as:



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.

Example) Create four different blur effects to an image with 1px, 4px, 8px, and 12px.

HTML

```
<div class="filterContainer">
  <div class="filter1">
    <h4>blurred with 1px</h4>
    </div>
  <div class="filter2">
    <h4>blurred with 4px</h4>
    </div>
  <div class="filter3">
    <h4>blurred with 8px</h4>
    </div>
  <div class="filter4">
    <h4>blurred with 12px</h4>
    </div>
</div>
```

CSS

```
/* BLURRED IMAGES */
.filterContainer{height:700px;}
.filter1, .filter2, .filter3, .filter4{
  width: 40%; height: 40%;
  padding:2%; float:left;
}
```

Blurred Images

blurred with 1px



blurred with 4px



blurred with 8px



blurred with 12px



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

```
<h3>Images with Contrast</h3>
<div class="filterContainer">
  <div class="filter1">
    <h4>Contrast 0%</h4>
    </div>
  <div class="filter2">
    <h4>Contrast 20%</h4>
    </div>
  <div class="filter3">
    <h4>Contrast 50%</h4>
    </div>
  <div class="filter4">
    <h4>Contrast 90%</h4>
    </div>
</div>
```

Images with Contrast

Contrast 0%



Contrast 20%



Contrast 50%



Contrast 90%



sepia(%)

sepia converts the element to sepia color. **sepia(0%)** is default means that there is no sepia applies to the element, and **sepia(100%)** will make the image completely sepia. For this attributes, negative values are not allowed.

HTML

```
<h3>Sepia Images</h3>
<div class="filterContainer">
  <div class="filter1">
    <h4>Sepia 0%</h4>
    </div>
  <div class="filter2">
    <h4>Sepia 20%</h4>
    </div>
  <div class="filter3">
    <h4>Sepia 50%</h4>
    </div>
  <div class="filter4">
    <h4>Sepia 90%</h4>
    </div>
</div>
```

Sepia Images

Sepia 0%



Sepia 20%



Sepia 50%



Sepia 90%



grayscale(%)

grayscale converts the element to loose its black and white color. **grayscale(0%)** is default and it means that there isn't any changes in its grayscale. **grayscale(100%)** will make the image completely gray, black and white. This property does not allowed negative values.

HTML

```
<h3>Grayscale Images</h3>
<div class="filterContainer">
  <div class="filter1">
    <h4>Grayscale 0%</h4>
    </div>
  <div class="filter2">
    <h4>Grayscale 20%</h4>
    </div>
  <div class="filter3">
    <h4>Grayscale 50%</h4>
    </div>
  <div class="filter4">
    <h4>Grayscale 90%</h4>
    </div>
</div>
```

Grayscale Images

Grayscale 0%



Grayscale 20%



Grayscale 50%



Grayscale 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

```
<h3>Invert Images</h3>
<div class="filterContainer">
  <div class="filter1">
    <h4>Invert 0%</h4>
    </div>
  <div class="filter2">
    <h4>Invert 20%</h4>
    </div>
  <div class="filter3">
    <h4>Invert 60%</h4>
    </div>
  <div class="filter4">
    <h4>Invert 90%</h4>
    </div>
</div>
```

Invert Images

Invert 0%



Invert 20%



Invert 60%



Invert 90%



3.6. Understand CSS animation

CSS transitions

CSS transitions are one of the ways we can create animation with CSS, even though they are not called animations. By definition, animation means causing change over time and transitions absolutely do that. They just do it in a little bit of a different way than CSS keyframe animations do.

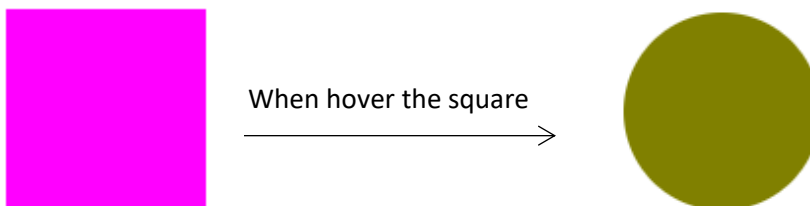
transition as property

To create a transition effect using the **transition** property, we must specify two things:

- The CSS property you want to add an effect to
- The duration of the effect

Note: If the duration part is not specified, the transition will have no effect, because the default value is 0.

Example) add transition property of 3 second to a square figure. The square figure changes to a circle when it is hovered



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

HTML

```
.square{height:100px; width:100px; background-color:magenta; transition: 3s;}  
.square:hover{background-color: olive; border-radius: 50%; }
```

CSS

We can also transition a specific property of an element just by adding the property name to the transition. To do so, we just need to make sure that the property is already part of the element.

Example) from the previous example, transition the background-color of the element

```
.square{height:100px; width:100px; background-color:magenta; transition:  
background-color 3s;}  
.square:hover{background-color: olive; border-radius: 50%; }
```

CSS

We can also use transition using different properties. We separate the properties using a comma.

Example) from the previous example, add a transition of 1 second to the **border-radius**

CSS

```
.square{height:100px; width:100px; background-color:magenta; transition:  
background-color 3s, border-radius 1s;}  
.square:hover{background-color: olive; border-radius: 50%; }
```

CSS transform property

The **transform** property applies a 2D or 3D transformation to an element. This property allows you to rotate, scale, move, skew, etc., elements.

Transform property has values as:

translate(x,y)	Defines a 2D translation
translateX(x)	Defines a translation, using only the value for the X-axis
translateY(y)	Defines a translation, using only the value for the Y-axis
scale(x,y)	Defines a 2D scale transformation
rotate(angle)	Defines a 2D rotation, the angle is specified in the parameter

For any CSS keyframe animation we create, we have to first define the animation. Basically, tell CSS what it is that should happen and we also need to assign that animation to a specific element or elements in your HTML. We can do these two steps in any order.

@keyframes

We will start creating our CSS animation by defining it's keyframes using the **@keyframes** rule. **@keyframes** are essentially a list describing everything that should happen over the course of the animation. We define the values for the animating properties at various points during the animation and any property that we will see change over the course of one's cycle of an animation.

- The **@keyframes** rule works like most other CSS rules, anything contained within its curly braces are considered part of that block.
- The second step to our **@keyframes** rule is giving our animation a name. It's really important to name the animation because without a name, it would not be able to assign the animation to an element.

There are a few options available to us for how we can define each **@keyframes** within our **@keyframes** rule:

- **from, to:** You define where to animate from and where to animate to.

Activity) create an animation that will move an image from one place to another within the x-axis.

For this exercise, we will download two images, one for the moving object and the other one as a back image. Once we have the images, we will create two containers for each of the images using `<div>` with position relative and absolute respectively.

```
<h3>CSS animation using @keyframes with attributes from - to </h3>
<section class="mountainImage">
  <div class="busImage"></div>
</section>
```

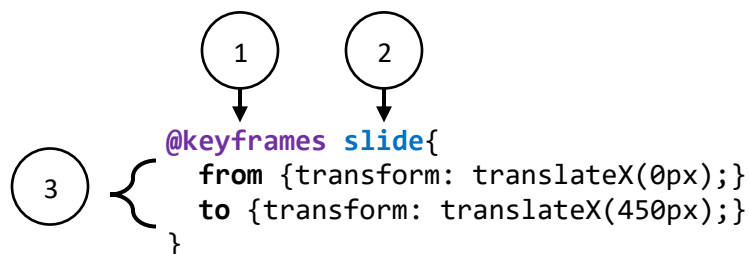
HTML

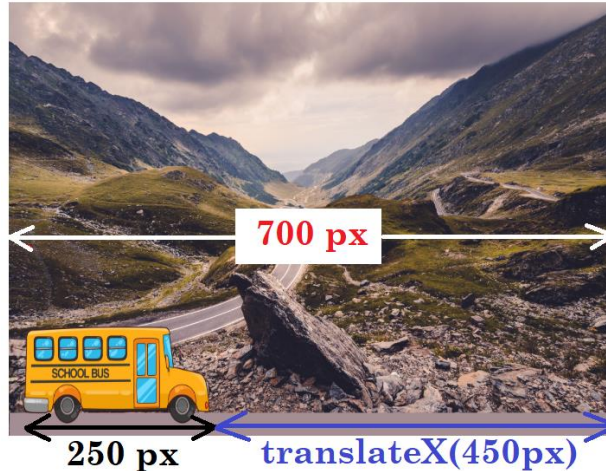
```
.mountainImage{
  width: 700px;
  height: 500px;
  position: relative;
  margin: auto;}
.busImage{
  width: 250px;
  height: 150px;
  position: absolute;
  bottom: 0;}
```

CSS

Once we have the images set up, we can start writing the css animation using **@keyframes**. We are going to name this animation *slide*. The animation will make the element to move from left to right. The basic rules to use **@keyframes** are:

1. Use **@keyframes** to create the animation frame
2. Assign a name to the keyframe
3. Specify what do you want the animation to do. For this project, we want the image to move *from* left *to* right. For this, we need to transform the element within the X axis from position 0 to 220px.





Now that we have the animation created, we need to add it into the image that we want to be animated, in our case, we will add the animation to the element that has the bus image, which has the class name *busImage*:

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

Now in css, we call for the class name and add some animation properties to it:

1. **animation-name: slide;** → This tells our image with class name **busImage** to apply the **@keyframes** animation named **slide**
2. **animation-duration: 2s;** → The second property we need to define is the animation duration. Our keyframes define what should change over the course of an animation, but they don't give any indication as to how long this should take, so that's what we do in animation duration. For this case, our animation **slide** is set to 2 seconds.

```
.busImage{
  1 → animation-name: slide;
    animation-duration: 3s; ← 2
```

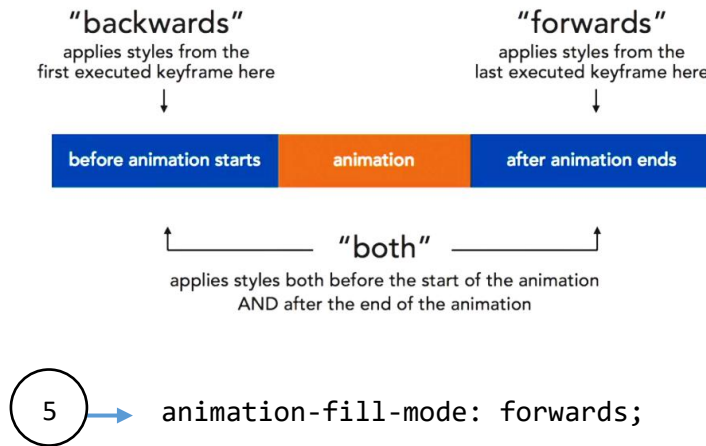
3. The **animation-iteration-count** property is also a good one to set for yourself even though it has a default value. This one determines how many times the animation will play. By default, the animation runs once, but we can set it runs **infinite** times.

```
3 → animation-iteration-count: 2;
```

4. **animation-delay** property sets the waiting before the animation actually executes.

```
4 → animation-delay: 2s;
```


5. **animation-fill-mode** is a way of telling the animating element what to do outside of the actual duration of its animation and it can take four values: *none*, *backwards*, *forwards*, and *both*. The default value is *none*. If we use the value *forwards*, the image will stay at the position at the end of the animation.



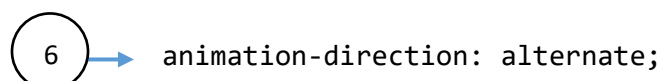
6. **animation-direction** let us to manipulate what order our **keyframes** are executed in. Animation-direction can be set to *normal*, *reverse*, *alternate*, and *alternate-reverse*.

An animation-direction of **normal** means that all iterations of the animation will be played as specified. So your keyframes will play from start to end. That means your keyframes will play from your from keyframe to your to keyframe, or your 0% percent keyframe to your 100% keyframe exactly in the order that you wrote them.

The animation direction of **reverse** means that all iterations of your animation will be played in the reverse direction. Your last keyframe will be played first, and your first keyframe will be played last. So this would play your animation from your to keyframe to your from keyframe, or your 100% keyframe to your 0% keyframe.

An animation-direction of **alternate** means that the direction of your animation will alternate each iteration of the animation. It will play in the normal direction the first time. So that would be 0% keyframe to 100% keyframe, and in the reverse direction the second time. So that would be the 100% keyframe back to 0%. **alternate** can only be used if your animation plays more than once because otherwise you won't have any space to see the alternating. So we'll change our animation iteration count to two. With those two small changes made, we'll go back to our robot and see how it's changed.

An **alternate-reverse** works in very much the same way as *alternate*, but it starts playing your animation in the reverse direction first, and then normal. **alternate-reverse** works in much the same way as *alternate*, it plays in reverse the first time, and then forwards, but it still plays twice.



animation-timing-function is the way speed is distributed across the duration of our animation. The animation-timing-function keywords include *ease*, *linear*, *ease-in*, *ease-out*, and *ease-in-out*.

With the **linear** easing our robot moves across the screen at the same constant speed for the entire animation. Linear easing creates a constant speed of motion that never changes at all. This is often perceived as mechanical motion because nothing in real life actually moves like that.

ease allows the animation to start and slowly and then speed up to a uniform speed.

ease-in has a distinct starting slow and then speeding up as the image reaches it's destination.

ease-out is the opposite of ease-in. The image starts at a higher speed and then slows down as it gets to it's destination.

ease-in-out combines the effects of ease-in and ease-out. It starts out a little bit slow, hits a top speed in the middle of the animation, and then slows down as it reaches the end.

7 → animation-timing-function: ease;

Infinitely looping animation

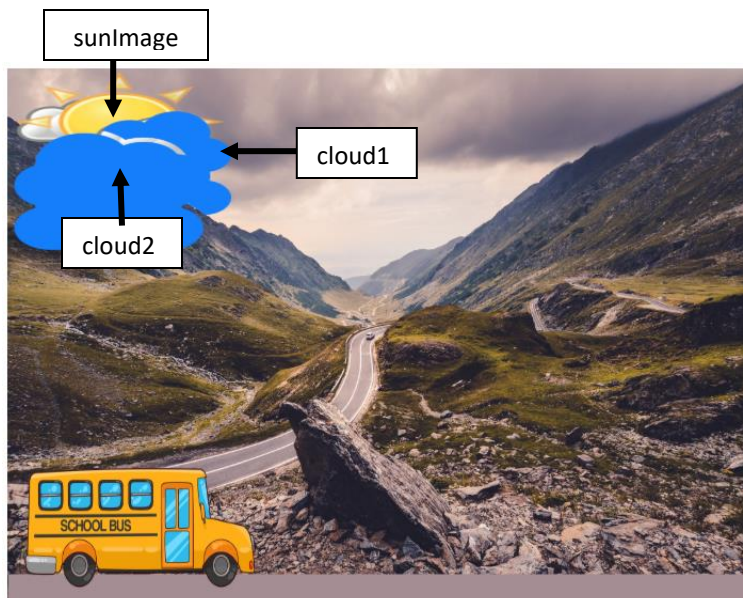
CSS is great for looping animation because it makes it possible to set an infinite loop with much less code than other methods.

Example) Using the previous exercises, creating a couple of animated clouds that continually drift across the sky with different distance and speed. For this, we can download a cloud image and create three containers of different sizes for each cloud.

```
<div class="mountainImage">
  <div class="busImage"></div>
  <div class="sunImage"></div>
  <div class="cloud1"></div>
  <div class="cloud2"></div>
</div>
```

HTML

Since the clouds are displayed on top of the back image, the containers' position must be set to absolute. We can also set different width and height to the clouds container and also the top position to each cloud.



CSS

```
/* animation clouds */
.sunImage{
  width: 30%;
  height: 20%;
  position: absolute;
  top: 0;
}
.cloud1{
  width: 30%;
  height: 26%;
  position: absolute;
  top: 20px;
}
.cloud2{
  width: 30%;
  height: 26%;
  position: absolute;
  top: 60px;
}
```

Once we have the HTML and CSS set, now we can set the animation using the @keyframes. Since we want the clouds to flow from left to right, we can use the property **from** to **to**

CSS

```
@keyframes cloudsFlowing {
  from {transform: translateX(0);}
  to {transform: translateX(500px);}
}
```

We are going to use the same animation for all three clouds, but we are going to adjust the properties to make each cloud to behave in a slightly different way. For example, we can set different **animation-iteration-count** to each cloud.

CSS

```
.sunImage{
  width: 30%;
  height: 20%;
  position: absolute;
  top: 0;
  animation-name: cloudsFlowing;
  animation-duration: 30s;
  animation-timing-function: linear;
  animation-iteration-count: infinite;
}
```

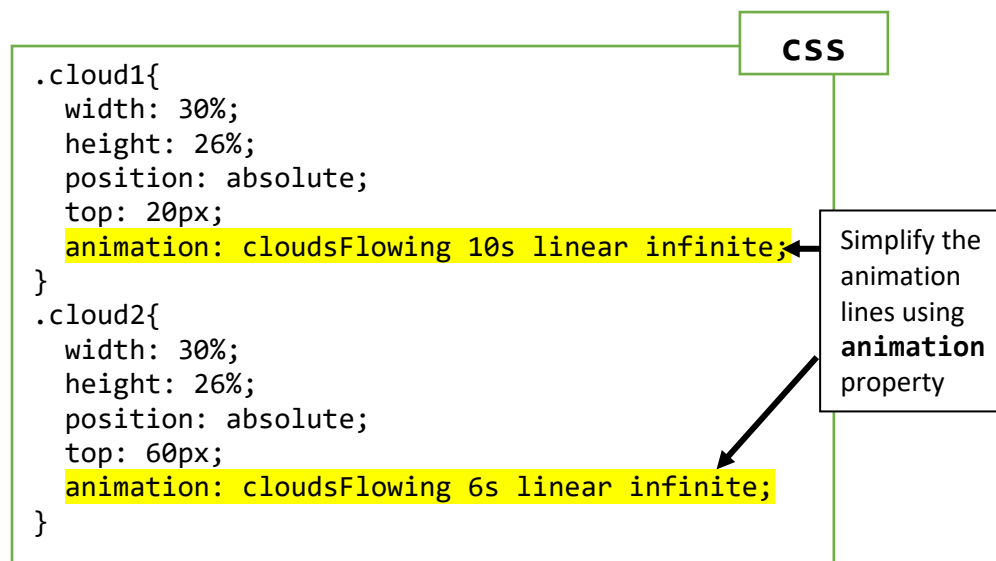
The animation properties can also be written in one line using the property **animation**. For example, we can simplify the following properties:

```
animation-name: cloudsFlowing;  
animation-duration: 30s;  
animation-timing-function: linear;  
animation-iteration-count: infinite;
```

using **animation** property, we can write each property values in one line and separate the values with a space:

```
animation: cloudsFlowing 30s linear infinite;
```

Now, we can apply the **cloudsFlowing** animation to the other two clouds with 8 and 5 seconds respectively.



opacity

One of the transition properties is **opacity**. **Opacity** property happens when a new value is assigned to it. It will cause a smooth change between the old value and the new value over a period of time. The value of opacity is ranged in between 0 and 1 where 0 is clear and 1 is the full color of the element.

Example) Using the previous animation, make the clouds to be almost half visible when they reach the edge of the right side.

CSS

```
@keyframes cloudsFlowing {  
  from{transform: translateX(0); opacity: 0.9;}  
  to{transform: translateX(500px); opacity: 0.3;}  
}
```



CSS animation using "%" time frame

CSS animation in an element can be slip using % time frame. For example, from a `@keyframes` animation using **from** to **to** attributes, we can replace **from** to **0%** and **to** to **100%**:

```
@keyframes cloudsFlowing{  
  from{ transform: translateX(0px); opacity:1;}  
  to{transform: translateX(250px);opacity:0.5;}  
}  
  
@keyframes cloudsFlowing{  
  0%{ transform: translateX(0px); opacity:1;}  
  100%{transform: translateX(250px);opacity:0.5;}  
}
```

Example) Create an animation that will move a square from one position to another and change its background color at the different position. The total time of the animation is 10s and the animation will repeat 20 times.

To understand how the position works on an element, first we need to create a container to place the square:

HTML

```
<h3> CSS animation using "%" time frame </h3>  
<div class=" frameContainer">  
</div>
```

```
.frameContainer{
  width: 1500px;
  height: 600px;
  background-color: lightblue;
  position: relative;
}
```

CSS

Once we have the container for the square, we create the square that sits inside the container:

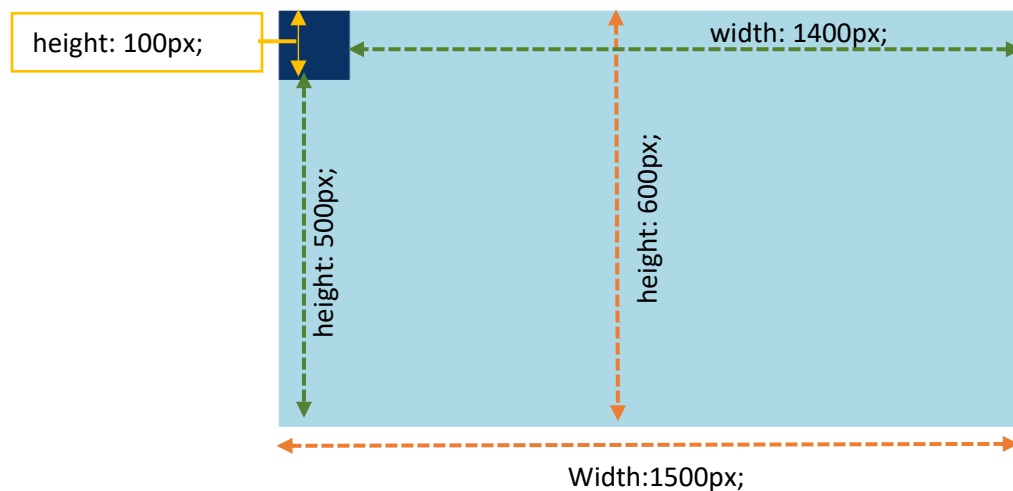
Add this line to create a square using <div> element

```
<div class="frameContainer">
  <div class="animationSquare"></div>
</div>
```

HTML

```
.animationSquare{
  width: 100px;
  height: 100px;
  background-color: rgb(10,50,100);
  position: absolute;
}
```

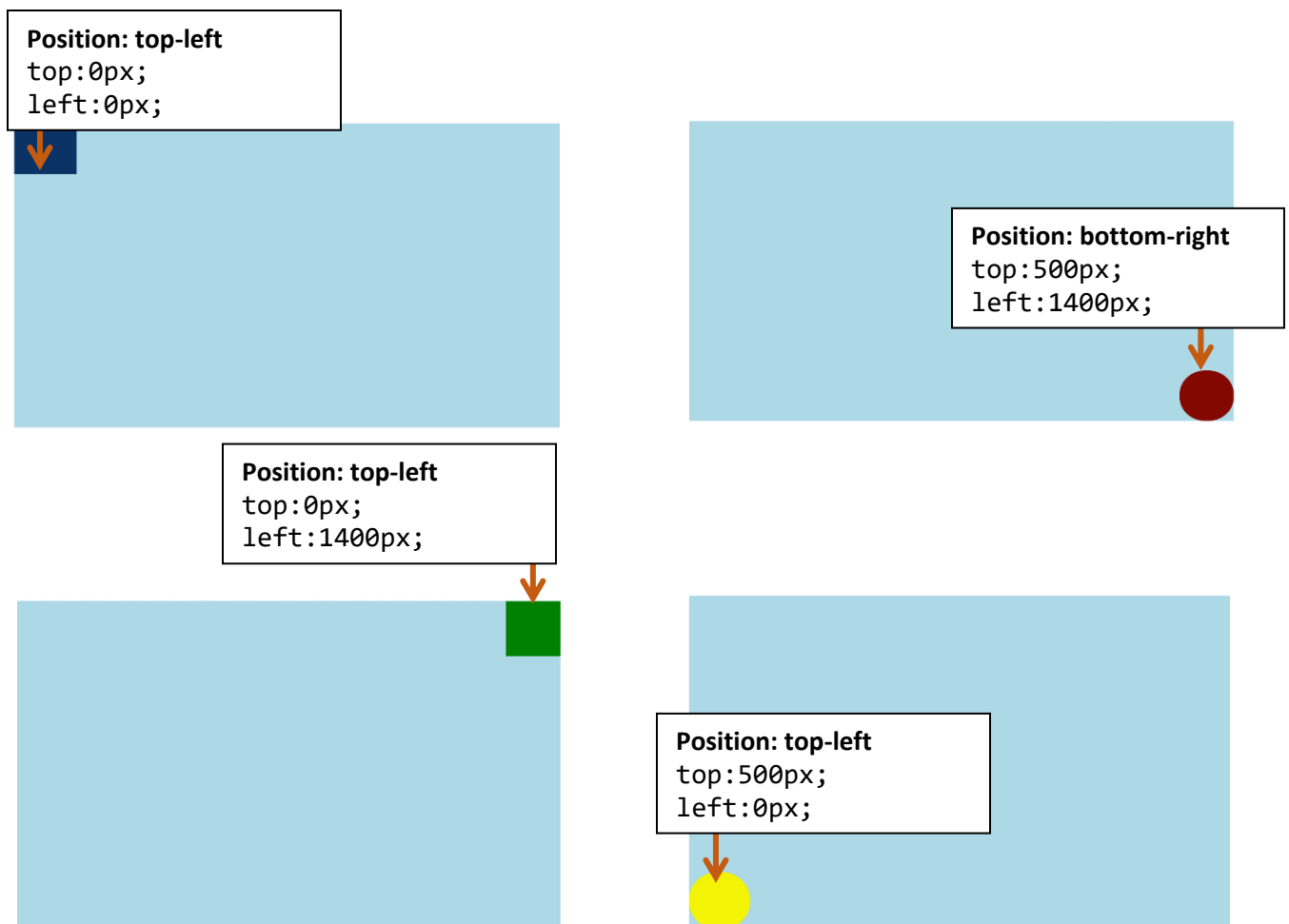
CSS



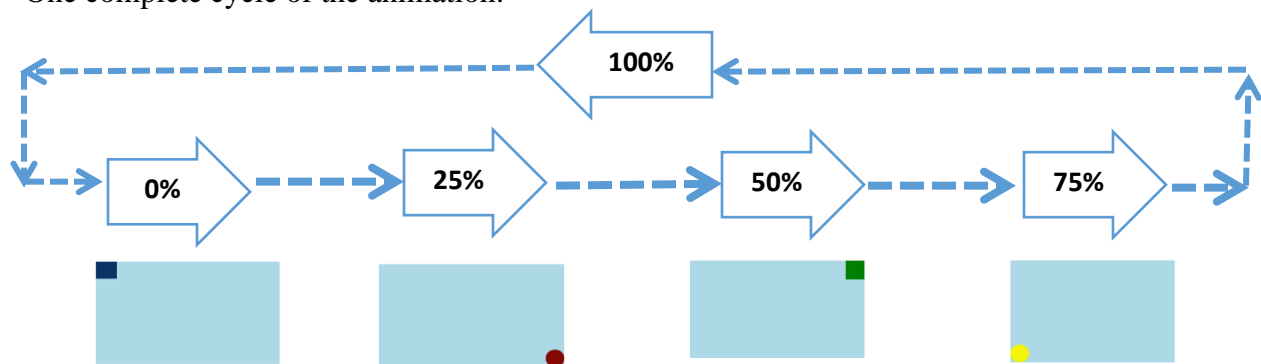
After the square is set, we need to create the animation using % time frame as the following:

Total time	10 seconds				
Time frame	0%	25%	50%	75%	100%
	0 to 2 secs	2.1 to 4 secs	4.1 to 6 secs	6.1 to 8 secs	8.1 to 10 secs
background color	Darkblue	Darkred	Green	Yellow	darkblue
border-radius	0% (squared shape)	50% (circled shape)	0% (squared shape)	50% (circled shape)	0% (squared shape)
Position	Top-left	Bottom-right	Top-right	Bottom-left	Top-left

To create the animation, we need to understand the position first. The position of an element is measured from the top-left corner of the element:



One complete cycle of the animation:



CSS

```
@keyframes animation_square{
0% {background-color: darkblue; left: 0px; top: 0px; border-radius: 0%;}
25%{background-color: darkred; left:1400px; top: 500px; border-radius: 50%;}
50%{background-color: green; left: 1400px; top: 0px; border-radius: 0%;}
75%{background-color: yellow; left: 0px; top: 500px; border-radius: 50%;}
100%{background-color: darkblue; left: 0px; top: 0px;border-radius: 0%;}
}
```

Rotating animation

Rotation is a CSS property that defines a 2D rotation and the rotation angle is specified within the parenthesis. For example, if we want to rotate an element 200 degree clockwise, we can write: `rotate(200deg)`. If we want to rotate an element 200 degree counter-clockwise, we can write: `rotate(-200deg)`

Activity) Create an animation that will rotate an image clockwise 180 degree and then 360 degree counter-clockwise. The animation duration will be 6s and it will repeat 10 times.

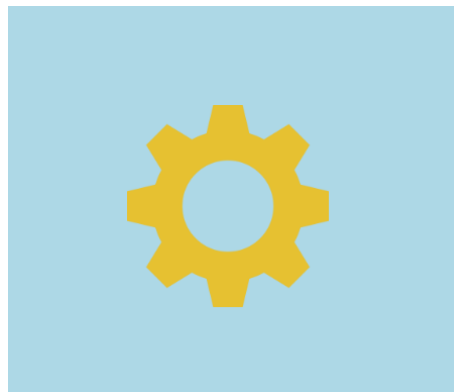
For this animation, first, we create the animation structure in HTML and CSS:

HTML

```
<h3>animation using transform property "rotate"</h3>
<div class="frameRotation">
  <div class="rotationImage">
    
  </div>
</div>
```

CSS

```
.frameRotation{
width: 1500px;
height: 500px;
background-color: lightblue;
position: relative;}
.rotationImage{
width: 200px;
height: 200px;
position: absolute;
top: 20%;
left: 40%;
animation: spin 10s 20;
}
```



Once the structure is set, we can start the animation according to the following time frame:

Total time	6 seconds			
Time frame	0%	30%	70%	100%
rotation	Rotate 0 deg	Rotate 180 deg	Rotate -360 deg	Rotate 0 deg

```
@keyframes spin {  
  0%{transform: rotate(0deg);}  
  30%{transform: rotate(180deg);}  
  70%{transform: rotate(-360deg);}  
  100%{transform: rotate(0deg);}  
}
```

CSS

Zooming an image

CSS animation transform property has value `scale()` to create zoom-in and zoom-out effect on an element. There are different parameters that we can use within the scale's parenthesis, for now, we are going to use one value inside the parenthesis. The highest value is 1 and the lowest value is 0. 1 means 100% of the actual element's size, and 0 means 0% of the actual element's size.

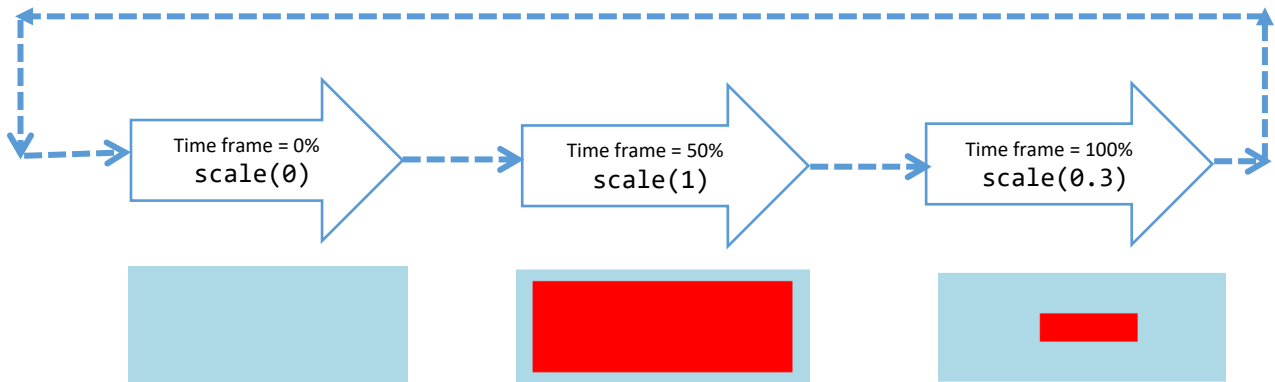
100% of the element size
`transform: scale(1);`



30% of the element size
`transform: scale(0.3);`



Example) Create an animation that the element will zoom-in to 100% of the image and zoom-out to 30% of the image. The animation duration is 10s and it will play 5 times.



HTML

```
<h3>animation using transform property "scale"</h3>
<div class="scaleAnimation">
  <div class="zoomImage"></div>
</div>
```

CSS

```
.zoomImage{
  width: 80%;
  height: 80%;
  top: 10%;
  left: 10%;
  background-color: red;
  position: absolute;
  animation: zoomAnimation 10s 5;}
@keyframes zoomAnimation {
  0%{transform: scale(0);}
  50%{transform: scale(1);}
  100%{transform: scale(0.3);}
}
```

3.7. Responsive Webpages

Making a webpage responsive to the screen size is an important features in web technology. Nowadays, web pages open on devices with different screen sizes, for example a **desktop computer** screen has a width of 1200px, a Samsung Galaxy Tab 10.1 tablet has the width of 900px, and the width of a Samsung Galaxy J7 smartphone is 720px . Therefore, it is important that when you design a webpage, the elements on the webpage can adjust to the screen of the device. You can visit: <http://screensiz.es/> to view more devices width.

For app view development, it is an important to add the following line:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

The **viewport** is the user's visible area of a web page. It varies with the device, and will be smaller on a mobile phone than on a computer screen. **viewport** element gives the browser instructions on how to control the page's dimensions and scaling.

The **width=device-width** part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The **initial-scale=1.0** part sets the initial zoom level when the page is first loaded by the browser.



Without the viewport meta tag



With the viewport meta tag

Flexbox container

The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning. Flexbox is a one-dimension layout method for layout g out items in rows or columns. It allows us to distribute space dynamically across elements of an unknown size, hence the term *flex*.

Example) Create a flex container for three square division

```
<section class="flex_container">
  <div class="sqr"></div>
  <div class="sqr"></div>
  <div class="sqr"></div>
</section>
```

HTML

```
.flex_container{
  border: solid gray;
  width: 80%;
  margin: 10%;
}
.sqr{width: 100px; height: 100px;}
.sqr:nth-child(1){background-color: magenta;}
.sqr:nth-child(2){background-color: olive;}
.sqr:nth-child(3){background-color: orange;}
```

CSS

without applying a *display: flex* to the flex container, the output will look as:



Now, if we add the *display: flex* property to the flex container, the three division will wrap, from left to right, around the flex container

```
.flex_container{
  border: solid gray;
  width: 80%;
  margin: 10%;
  display: flex;
}
```



We can also change the direction from right to left by adding the property *flex-direction: row-reverse*;

```
.flex_container{  
  border: solid gray;  
  width: 80%;  
  margin: 10%;  
  display: flex;  
  flex-direction: row-reverse;  
}
```



justify-content property

The CSS justify-content property defines how the browser distributes space between and around content items along the main-axis of a flex container, and the inline axis of a grid container.

justify-content uses different values such as: **start**, **center**, **space-between**, **space-around**, and **space-evenly**

```
.flex_container{  
  border: solid gray;  
  width: 80%;  
  margin: 10%;  
  display: flex;  
  flex-direction: row;  
  justify-content: center;  
}
```



```
.flex_container{
  border: solid gray;
  width: 80%;
  margin: 10%;
  display: flex;
  flex-direction: row;
  justify-content: space-between;
}
```



```
.flex_container{
  border: solid gray;
  width: 80%;
  margin: 10%;
  display: flex;
  flex-direction: row;
  justify-content: space-around;
}
```



```
.flex_container{
  border: solid gray;
  width: 80%;
  margin: 10%;
  display: flex;
  flex-direction: row;
  justify-content: space-evenly;
}
```



Flex-wrap property

The **flex-wrap** property specifies whether the flexible items should wrap or not.

Example) from the previous example, let change the **width** of the divisions to **500px**.



For this case, if we change the browser window to a smaller view, all three divisions it will squeeze to fit the width of the *flex_container*



If we do not want the divisions to squeeze, but instead, we want to keep the division's width and have the divisions to wrap around the *flex_container*, then we can add the **flex-wrap:wrap;** property to the *flex_container*



Example) from the previous example, if we set the **height** of the *flex_container* to **200px** and the **flex-direction** to **column**,

```
.flex_container{
  border: solid gray;
  width: 80%;
  margin: 10%;
  height: 200px;
  display: flex;
  flex-direction: column;
  justify-content: space-evenly;
}
```

all three division at the container will squeeze to fit the height of the container. The output will look as:



Now, if we want to keep the height of each divisions, we can use **flex-wrap: wrap** to allow the flexible divisions to wrap within the *flex_container*

```
.flex_container{
  border: solid gray;
  width: 80%;
  margin: 10%;
  height: 200px;
  display: flex;
  flex-direction: column;
  justify-content: space-evenly;
  flex-wrap: wrap;
}
```



Align-item property

Align-items property sets the align-self value on all direct children as a group. In flexbox, it controls the alignment of items inside the flex container on the cross axis.

Example) using the three divisions, set the divisions to align to the end of the flex container.

Without using the *align-item* property,

```
.flex_container{
  border: solid gray;
  width: 80%;
  height: 200px;
  margin: 10%;
  display: flex;
  flex-direction: row;
  justify-content: center;
}
```



By adding the ***align-items:flex-end;*** to *flex_container*

```
.flex_container{  
  border: solid gray;  
  width: 80%;  
  height: 200px;  
  margin: 10%;  
  display: flex;  
  flex-direction: row;  
  justify-content: center;  
  align-items:flex-end;  
}
```



We can also set the divisions to the center of the cross axis of the flex container by using ***align-items:center;***

```
.flex_container{  
  border: solid gray;  
  width: 80%;  
  height: 200px;  
  margin: 10%;  
  display: flex;  
  flex-direction: row;  
  justify-content: center;  
  align-items:center;  
}
```



align-content property

align-content property sets the distribution of space between and around content items along a flexbox's cross-axis or a grid's block axis. The ***align-content*** property modifies the behavior of the ***flex-wrap*** property. It is similar to ***align-items***, but instead of aligning flex items, it aligns flex lines. It sets the distribution of space between and around content items along a flexbox's cross-axis or a grid's block axis.

Note: There must be multiple lines of items for this property to have any effect!

Example) use the previous example and change the **width** of the division to **300px**

```
.flex_container{  
    border: solid gray;  
    width: 80%;  
    height: 300px;  
    margin: 0 auto;  
    display: flex;  
    flex-direction: row;  
    justify-content: center;  
    align-items: flex-end;  
    flex-wrap: wrap;  
}
```

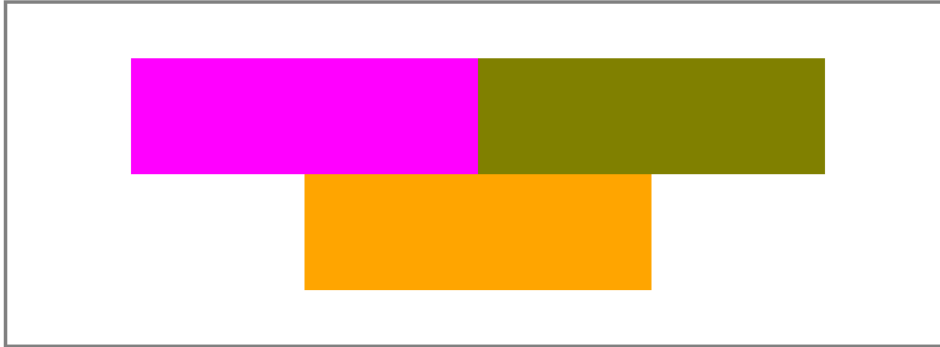
```
.sqr{width: 300px; height: 100px;}
```



Now, by adding the property ***align-content: space-between;***



Now, if we change ***align-content: center;***



align-self property

The ***align-self*** CSS property overrides a grid or flex item's align-items value. In Grid, it aligns the item inside the grid area. In Flexbox, it aligns the item on the cross axis. The ***align-self*** property specifies the alignment for the selected item inside the flexible container.

Example) using the previous example, align the second division to the bottom of the flex container

```
.flex_container{
  border: solid gray;
  width: 80%;
  height: 300px;
  margin: 0 auto;
  display: flex;
  flex-direction: row;
  justify-content: center;
  align-items:center;
  flex-wrap: wrap;
}
.sqr:nth-child(2){background-color: olive; align-self: flex-end;}
```



@media query

Media query is a CSS technique introduced in CSS3 and it is used to make responsive pages.

It uses the `@media` rule to include a block of CSS properties only if a certain condition is true.

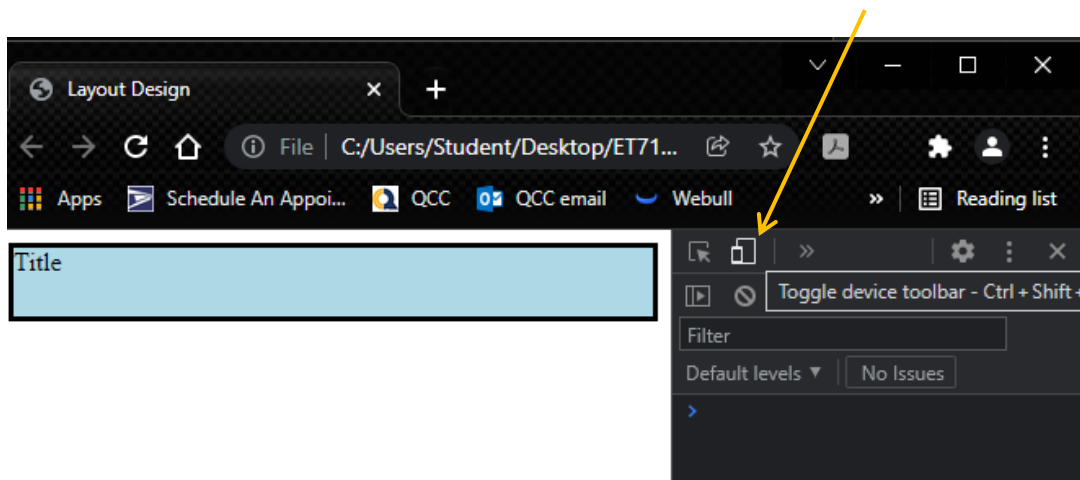
When using `@media`, instead of changing styles when the width gets *smaller* than 800px, we should change the design when the width gets *larger* than 800px. This will make our design Mobile First. The syntax code will look as:

```
@media only screen and (min-width: 800px){  
  
}
```

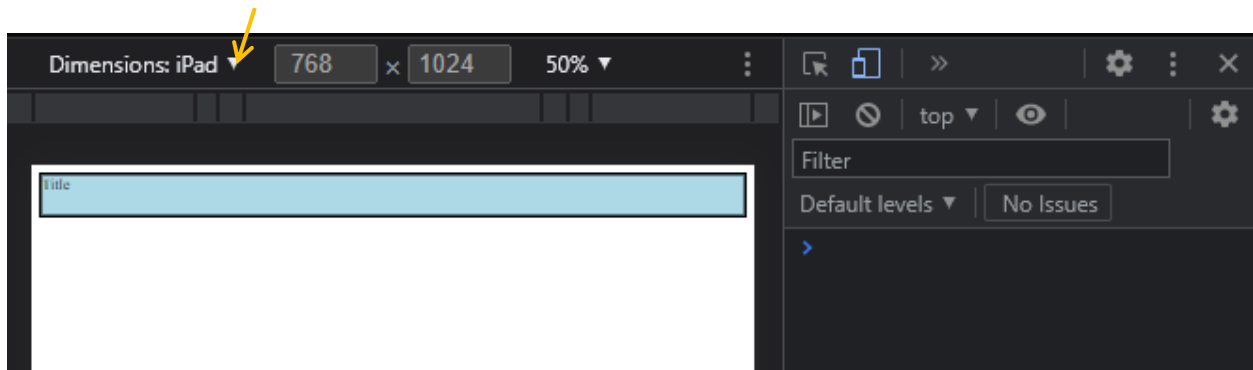
Between the curly brackets should go the CSS attributes of the elements that will be changed when the screen has the width of 800px or greater.

Mobile First means designing for mobile before designing for desktop or any other device (This will make the page display faster on smaller devices). Some web developer prepares to design a mobile view first as it moves toward the tablet's, laptop's, and desktop's screen size. Therefore, when we apply `@media` query, the screen size has property `min-width: 800px`. On the other hand, since the material in this lab manual was designed from a desktop computer screen view, then we can design from the desktop computer screen toward the tablet's and smartphone's screen size. For this, instead of using `min-width: 800px` we use `max-width: 800px`.

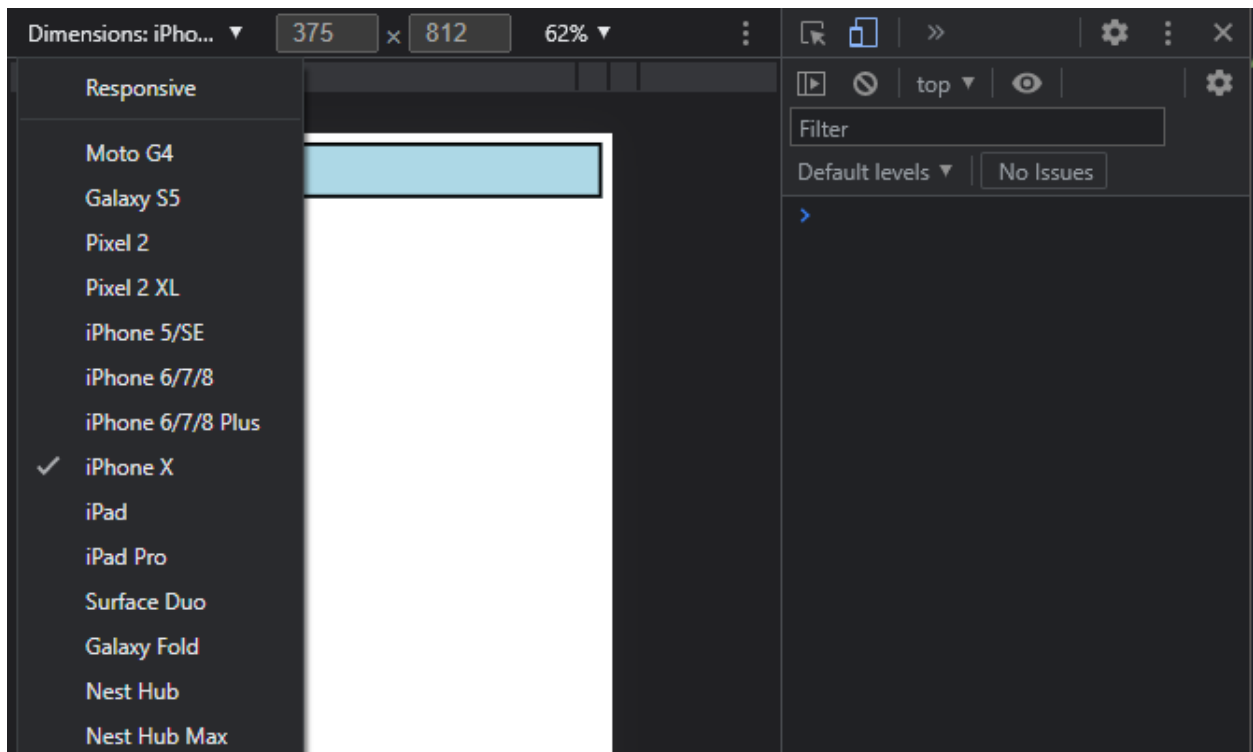
Also, if we are designing from mobile view first, we can change the browser view to mobile view by using the function key **F12** and then click on the **Toggle device toolbar**



Once clicked on the Toggle device toolbar, we can select the size of the mobile view:



Let us to pick the iPhone X screen size. In this case, since iPhone X has a width of 375px, then we can design a view up to 375px or 450px so the design can be used to other smartphone screen.



Example) Create the following three different layout, for smartphone view, tablet, and laptop or desktop view, using media query. The sizes for smartphone is up to 450px, for tablet is from 450px up to 800px, and for laptop or desktop view from 800px and up. Starts designing from smartphone view.

New York City

[External link 1](#)

[External link 2](#)

[External link 3](#)



New York City comprises 5 boroughs sitting where the Hudson River meets the Atlantic Ocean. At its core is Manhattan, a densely populated borough that's among the world's major commercial, financial and cultural centers. Its iconic sites include skyscrapers such as the Empire State Building and sprawling Central Park. Broadway theater is staged in neon-lit Times Square.

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>Layout Design</title>
  </head>
  <body>
    <header>New York City </header>
    <div class="container">
      <div class="info_body">
        <nav>
          <a href="#">External link 1</a>
          <a href="#">External link 2</a>
          <a href="#">External link 3</a>
        </nav>
        <div class="figure">
          
        </div>
      </div>
      <section>New York City comprises 5 boroughs sitting where the Hudson River
meets the Atlantic Ocean. At its core is Manhattan, a densely populated
borough that's among the world's major commercial, financial and cultural
centers. Its iconic sites include skyscrapers such as the Empire State
Building and sprawling Central Park. Broadway theater is staged in neon-lit
Times Square. </section>
    </div>
    <footer> </footer>
  </body>
</html>
```

```

*{box-sizing: border-box;}
img{width: 100%; height: 100%;}
/* --- smartphone view - small view --- */
@media only screen and (max-width: 450px){
  header{
    background-color: purple;
    height: 3em;
    font-size: 2em;
    text-align: center;
    padding-top: 1em;
    color: white;
  }
  .container{
    margin-top: 2em;
    height: 35em;
  }
  nav a{
    text-decoration: none;
    display: block;
    padding: 0.3em 0.6em;
    background-color: lightblue;
    font-size: 1.1em;
    margin: 0.3em 0em;
    text-align: center;
  }
  section{
    height:auto;
    margin-top: 1em;
    background-color: lightblue;
    padding: 1em;
    font-size: 1.1em;
  }
  footer{
    height: 5em;
    background-color: blue;
    margin-top: 1em;
  }
}
}
```

Once the CSS file is complete with the smartphone or small screen view, we can separate the styling that are used for all screen size outside the @media query

```

*{box-sizing: border-box;}
img{width: 100%; height: 100%;}
header{
    background-color: purple;
    text-align: center;
    color: white;
    height: 3em;
    font-size: 3em;
    padding-top: 1em;
}
.container{
    height: 35em;
}
nav a{
    text-decoration: none;
    background-color: lightblue;
    text-align: center;
    display: block;
}
section{
    height:auto;
    background-color: lightblue;
}
footer{
    background-color: blue;
}
/* --- smartphone view - small view --- */
@media only screen and (max-width: 450px){
    .container{
        margin-top: 1em;
    }
    header{
        font-size: 2em;
    }
    nav a{
        padding: 0.3em 0.6em;
        font-size: 1.1em;
        margin: 0.3em 0em;
    }
    .figure{margin-top: 1em;}
    section{
        margin-top: 1em;
        padding: 1em;
        font-size: 1.1em;
    }
    footer{
        height: 5em;
        margin-top: 1em;
    }
}

```

Once the smartphone view is set, we can create the following layout for a tablet screen. For this, we set the screen size in between 450px up to 800px:

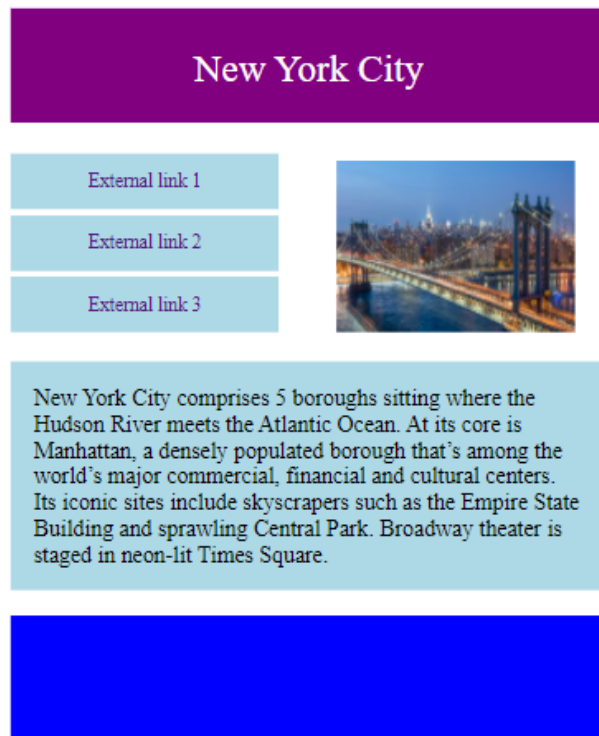
```

@media only screen and (max-width: 800px) and (min-width:450px){

}

```

Inside the @media query, we can work on styling the elements in the layout until it looks like the view below:



Now we can create a breakpoint for a desktop view. For this, we can set the @media query to 800px and up.

```
@media only screen and (min-width:800px){  
  
}
```

Once again, we work on styling the elements in the layout within the @media query until it looks like the view below:



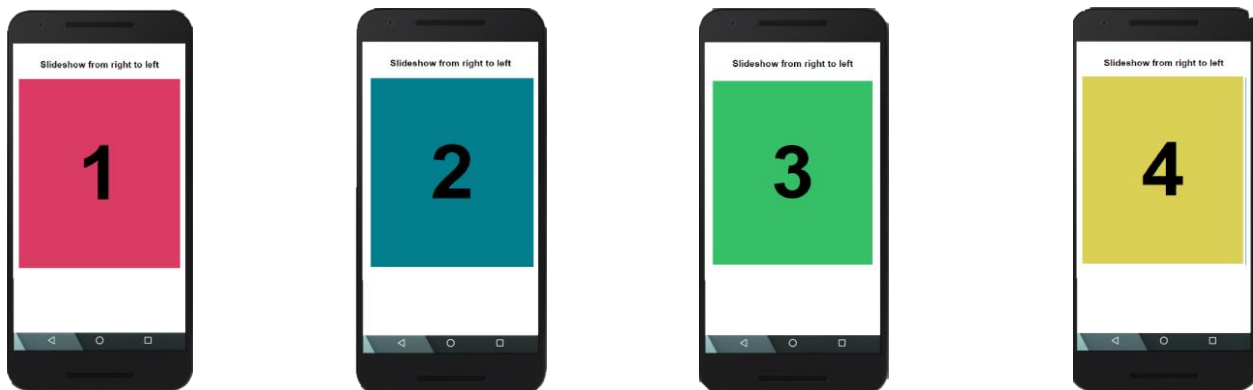
For this example, we created a breakpoints media to three screen sides, but always we can add as many breakpoints as we like. Also, the layout design of each of the screen side is up to the designer decision but it is always recommended to have the layout design before writing the HTML and CSS script.

3.8. Creating Slideshow using CSS animation

Slideshow is one of the fun features in website. It presents to the viewer a series of information, images, or text slides looping in certain time. The purpose of a slideshow is to promote visual interest or artistic value, sometimes accompanied with description or text, or to clarify or reinforce information, ideas, comments, solutions, or suggestions.

There are different way to create slideshow, one of the way to create slideshow in CSS is to use the *animation* feature.

Example) Create an infinite slideshow to display a number from 1 to 4, increasing order, with different background color.



Let's set up the html structure with a link to index.css

```
<!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>Slideshow by Huixin Wu</title>
  </head>
  <body>

  </body>
</html>
```

HTML

In index.css, we are going to set the **body** padding to 2%. This will prevent that elements within the body sit in the border of the smartphone's screen.

```
body {
  font-family: Helvetica, sans-serif;
  padding: 2%;
  text-align: center;
}
*{
  box-sizing: border-box;
}
```

CSS

Now that we have the **body** set, we can create the slide container with the id name as **slideshow1**

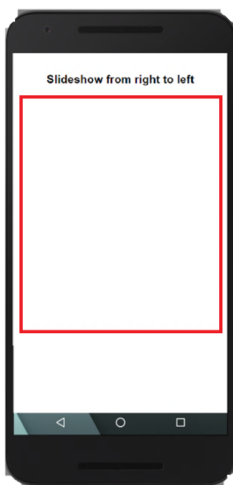
HTML

```
<!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>Slideshow by Huixin Wu</title>
  </head>
  <body>
    <div id="slideshow1">
      </div> <!--end of slideshow -->
  </body>
</html>
```

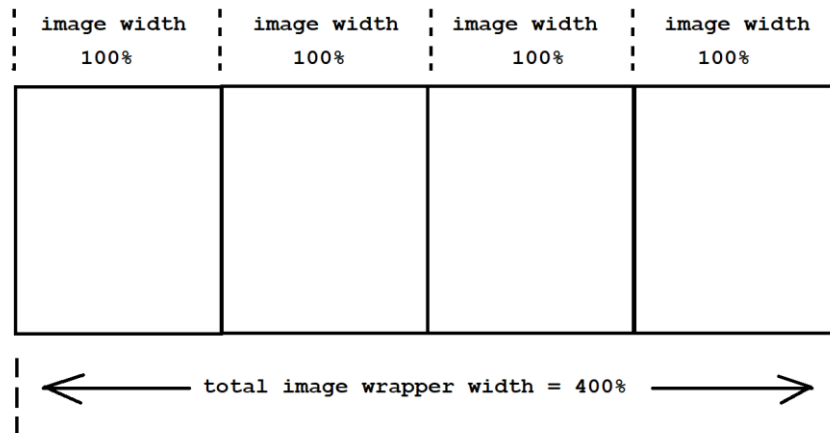
Once we have the html structure for slideshow1, in css, we are going to set the height to 400px, the width to fit 100% of the app screen, and all the overflow information in slideshow1 will be hidden. Also, we are going to add border to slideshow1 for reference, but remember that the border will be deleted once the slideshow frame is set.

CSS

```
#slideshow1{
  overflow: hidden;
  height: 400px;
  width: 100%;
  border: solid red;
}
```



After it, we need to create a container to wrap all four numbers. We are going to name the wrapper **slide-wrapper**. The total width of the **slide-wrapper** is the sum of the width of each image in the slideshow. For example, if we are planning to display four images in a slideshow and each image will display 100% to the width of the app screen, then the width of the **slide-wrapper** will be 400%.



```
<!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>Slideshow by Huixin Wu</title>
  </head>
  <body>
    <div id="slideshow1">
      <div class="slide-wrapper">

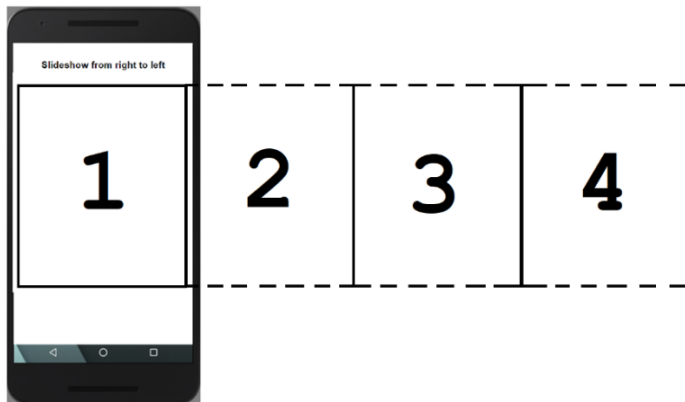
        </div> <!--end of slide-wrapper -->
      </div> <!--end of slideshow1 -->
    </body>
  </html>
```

HTML

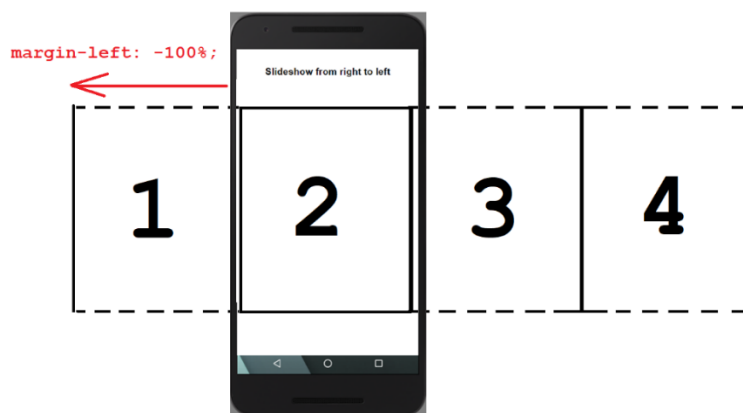
```
.slide-wrapper {
  width: 400%;
  height: 100%;
}
```

CSS

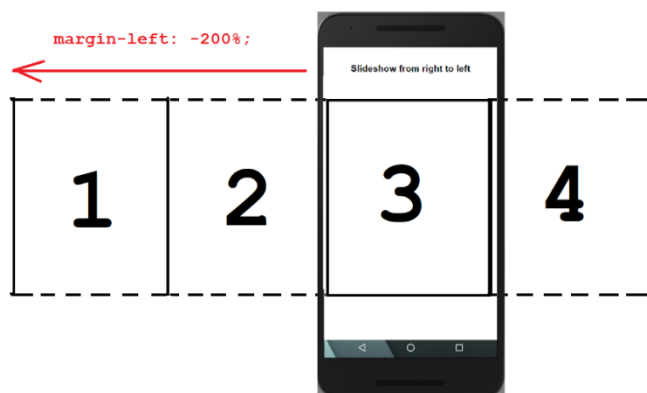
The other feature that we need to add to the **slide-wrapper** is the animation. The animation will slide each image in the slideshow from right to left one at the time. To do so, at each time frame the **slide-wrapper** will add margin to the left by 100% of the image width. Since the **slide-wrapper** is shifted to the left, the **margin-left** should be set to **-100%**. For example, at the first time frame, the slideshow will show the first image:



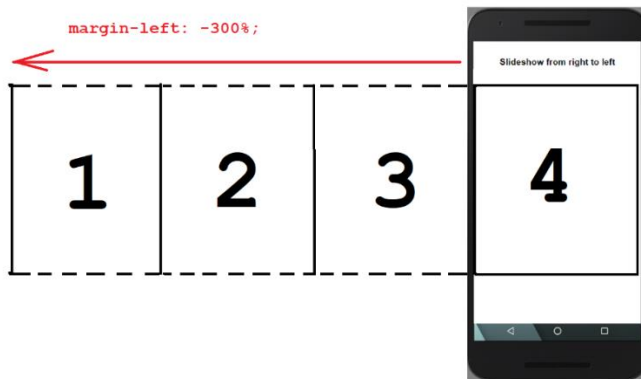
At the second time frame, the **margin-left** set to **-100%** to show the second image:



At the third time frame, the **margin-left** set to **-200%** to show the third image:



At the fourth time frame, the **margin-left** set to **-300%** to show the fourth image:



Add animation to **.slide-wrapper**

We are naming the animation as **slideshow**, the total animation duration is 12 seconds, and it is an infinite slideshow

```
.slide-wrapper {  
  width: 400%;  
  height: 100%;  
  animation: slideshow 12s infinite;  
}  
@keyframes slideshow {  
  0% {margin-left: 0px;}  
  35% {margin-left: -100%;}  
  70% {margin-left: -200%;}  
  100% {margin-left: -300%;}  
}
```

CSS

If we run the slideshow, nothing is showing because we have not set each image into the **slide-wrapper** yet. Since we are planning to show four numbers, then we need to create four slide divisions:

```
<!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>Slideshow by Huixin Wu</title>  
  </head>  
  <body>  
    <div id="slideshow1">  
      <div class="slide-wrapper">  
        <div class="slide"><h1>1</h1></div>  
        <div class="slide"><h1>2</h1></div>  
        <div class="slide"><h1>3</h1></div>  
        <div class="slide"><h1>4</h1></div>  
      </div> <!--end of slide-wrapper -->  
    </div> <!--end of slideshow1 -->  
  </body>  
</html>
```

HTML

Since all four **slide** divisions is set inside the **slide-wrapper**, then each **slide** division width must be set to 25%, $\frac{100\%}{4} = 25\%$

Also, all four divisions will be in-line from left to right, for this, we can set all four divisions to float left

```
.slide {  
  float: left;  
  height: 100%;  
  width: 25%;}
```

CSS

We can also add some CSS to all **h1** within **slide** class:

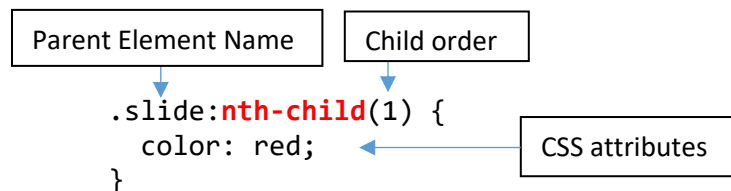
```
.slide h1{  
  text-align: center;  
  font-size: 10em;}
```

CSS

Now, we can run the slideshow and see the number displaying one at the time.

The next features to add to each slide is the background color. There is different way we can set different background color to each slide, for example, we can use different class name to each slide division, or add the CSS code in the HTML code using **style="background-color: yellow;"**

For this activity, we are going to use the **nth-child** selector in CSS to set different background color to each **slide** division. The **nth-child** selector is a structural pseudo-class, meaning it is used to style content based on its relationship with the parent element. For our program, the parent element will be **slide** because **slide** has four divisions of the same class name. Then, each **slide** division, according to line order, will be the **nth-child** of the **slide** element. The syntax of the **nth-child** is:



For example, if we are setting the background color for the first slide to lightblue, then we can write the code as:

```
.slide:nth-child(1) {  
  background-color: lightblue;  
}
```

Then, to set different background color to each four slide elements:

```
.slide:nth-child(1) {background-color:lightblue; }
.slide:nth-child(2) {background-color: pink;}
.slide:nth-child(3) {background-color: lightgreen;}
.slide:nth-child(4) {background-color: orange;}
```

Now, we can run the slideshow and see the number of each slide changes with different background color.

If we want each number of the slideshow to rest before it shifts to the next number, we can adjust the previous animation's time frame, **@keyframes** slideshow, to rest by 10% of the total time frame at each slide, and 20% of the total time frame as the transition time between slide:

```
@keyframes slideshow {
  0% {margin-left: 0px;}
  10% {margin-left: 0px;}
  30% {margin-left: -100%;}
  40% {margin-left: -100%;}
  60% {margin-left: -200%;}
  70% {margin-left: -200%;}
  90% {margin-left: -300%;}
  100% {margin-left: -300%;}
}
```

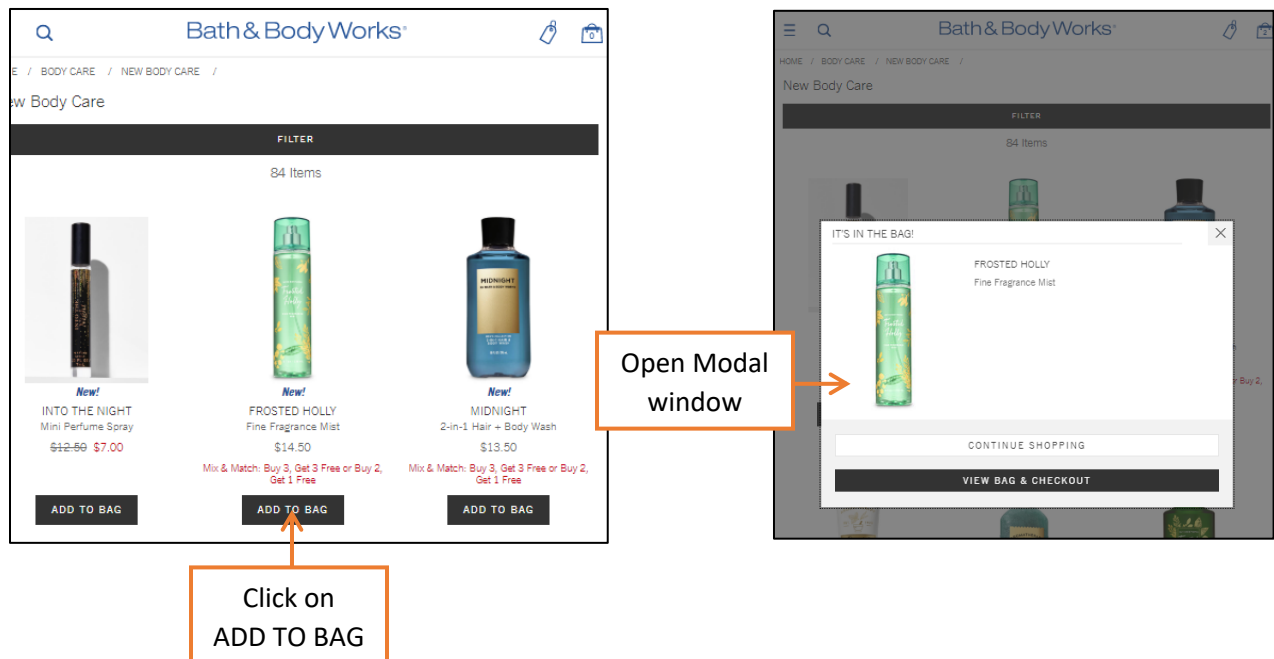
Run the slideshow and see how the number rest for 1.2 seconds, 10% of the total time frame 12 seconds, before it moves to the next number.

The other features that we can add to the animation is to see slideshow images to backward to the beginning to the slideshow when it reaches the last slide. For this, we can adjust the time frame to up 85% for the last slide. This will allow the slideshow to have the rest 86% to 100% to backward to the beginning to the slideshow. Therefore, the rest time at each slide will be 10% of the total time frame, the transition time between slides will be 15% of the total time frame, and the backward time will be 15% of the total time frame, which is from 86% to 100%:

```
@keyframes slideshow {
  0% {margin-left: 0px;}
  10% {margin-left: 0px;}
  25% {margin-left: -100%;}
  35% {margin-left: -100%;}
  50% {margin-left: -200%;}
  60% {margin-left: -200%;}
  75% {margin-left: -300%;}
  85% {margin-left: -300%;}
}
```

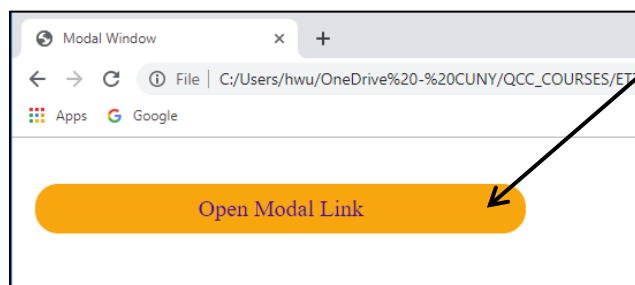
3.9. Creating a modal window using HTML and CSS

Modal windows are boxes that opens in front on the computer screen. They are widely used for advertisements, promotion code, item description, login/register forms, etc.



There are different techniques to create modal windows, one way to create a modal window is by using CSS's attributes as transition, opacity, pointer-event, and background gradient properties.

To create a modal window we need to create open modal link:




```

<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Modal Window</title>
    <link rel="stylesheet" href="modal.css" type="text/css">
  </head>
  <body>
    <a href="#openModal" class="linkModel">Open Modal Link</a>
  </body>
</html>

```

HTML

```

.linkModel{
  text-align:center;
  background-color: orange;
  font-size: 20px;
  padding: 10px;
  border-radius: 20px;
  text-decoration:none;
}

```

modal.css

Run the program to see if the modal link looks like this:

Open Modal Link

From the HTML code, the code line:

```
<a href="#openModal" class="linkModel">Open Modal Link</a>
```

we can see that the modal is linked to an element **id** as "#openModal". Therefore, we have to create the modal element using **<div>** element and we will **id** it as: **id="openModal"**. This division we will also has a class name as: **class="modalWindow"** for css attributes.

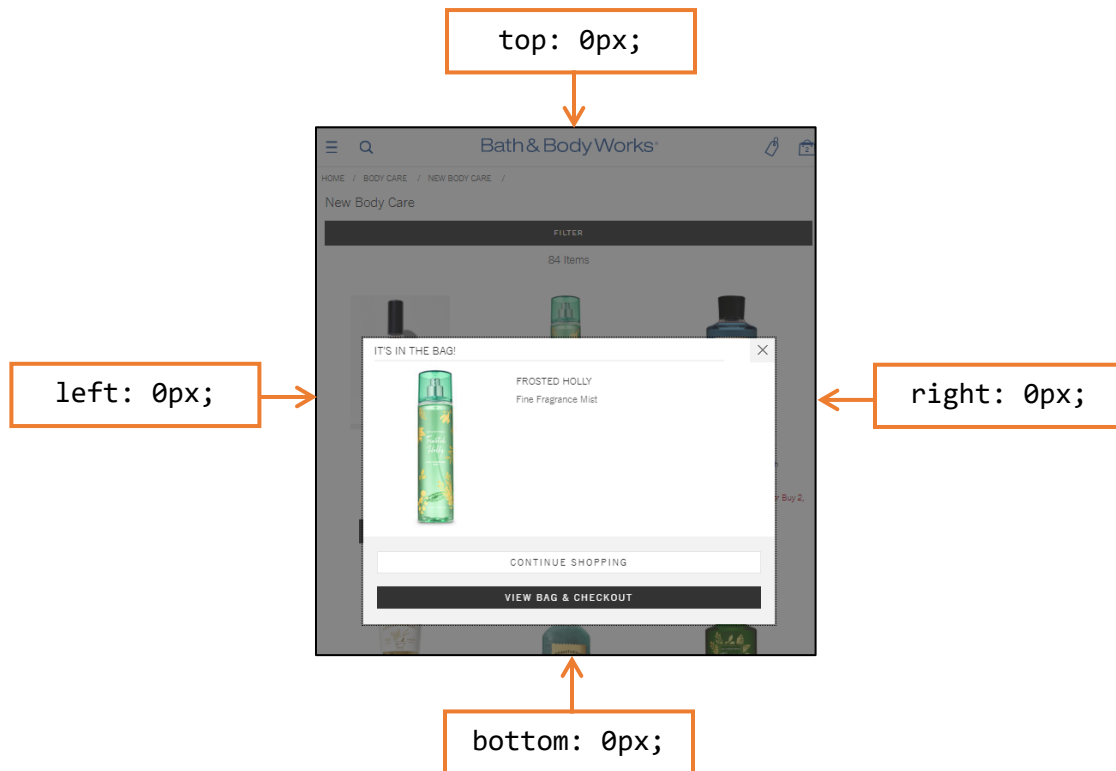
```

<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Modal Window</title>
    <link rel="stylesheet" href="modal.css" type="text/css">
  </head>
  <body>
    <a href="#openModal" class="linkModel">Open Modal Link</a>
    <!-- Modal Window container -->
    <div id="openModal" class="modalWindow">
      </div>
  </body>
</html>

```

HTML

Since the modal window will cover the entire screen, we can set the top, right, left, and bottom to 0px. This setting meaning that there will be 0px of distance between the top, right, left, and bottom to the modal window.



Once the modal window is created, we have to apply transparency to the modal window. This property can be set by setting the window background to 80% of transparency using the **rgba** color property:

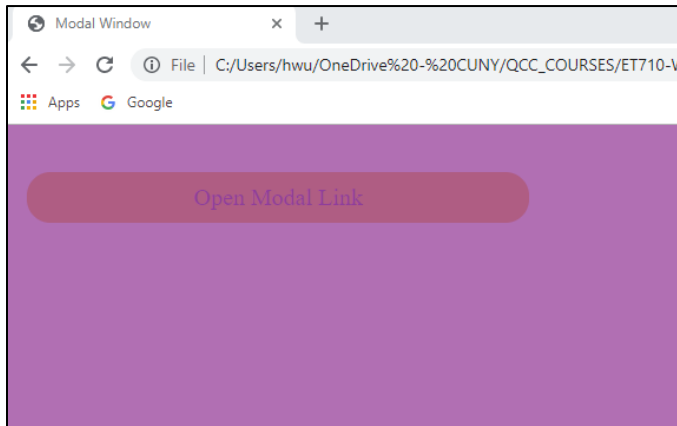
Background-color: rgba(160,80,160,0.8);

The CSS code should look as the following:

```
.linkModel{
width: 60%;
text-align:center; background-color: orange; font-size: 20px; padding: 10px;
border-radius: 20px;
}
.linkModel a{
text-decoration:none;
}
.modalWindow {
position: fixed; top: 0;
right: 0;
bottom: 0;
left: 0;
background-color: rgba(160,80,160,0.8);
}
```

modal.css

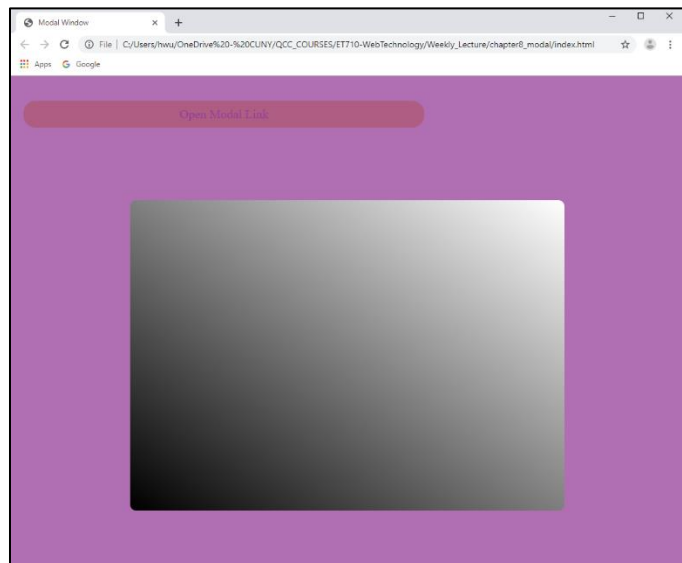
The internet browser should look as:



Having the background of the modal window set, we can add a message window on top of the modal window using a `<div>` with class name **msgText**. This message window is used to display the advertisement of the modal. Since the message window will display in-front of the modal window, therefore this division will have **position: relative**. To make the message window more colorful and organized, we can add linear-gradient background to it, set the width to 80%, the height to border- radius, and 40% of top margin:

```
.msgText{
  position: relative;
  width: 700px;
  height: 500px;
  border-radius: 10px;
  margin: auto;
  margin-top: 200px;
  background: linear-gradient(to top right, black, white);
}
```

The internet browser should look as:

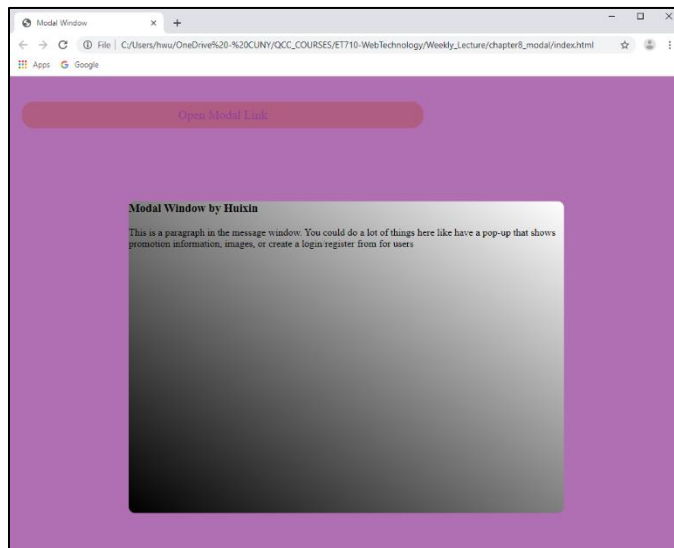


msgText is used as a message window. Therefore, we can add a title to this message window using **<h3>** and a text container using **<p>**.

HTML

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Modal Window</title>
    <link rel="stylesheet" href="modal.css" type="text/css">
  </head>
  <body>
    <a href="#openModal" class="linkModel">Open Modal Link</a>
    <!-- Modal Window container -->
    <div id="openModal" class="modalWindow">
      . <div class="msgText">
        <h3>Modal Window by Huixin</h3>
        <p>This is a paragraph in the message window. You could do a lot of
          things here like have a pop-up that shows promotion information, images,
          or create a login/register form for users</p>
      </div>
    </div>
  </body>
  .. .
```

Refreshing the internet browser, the modal window should look as:



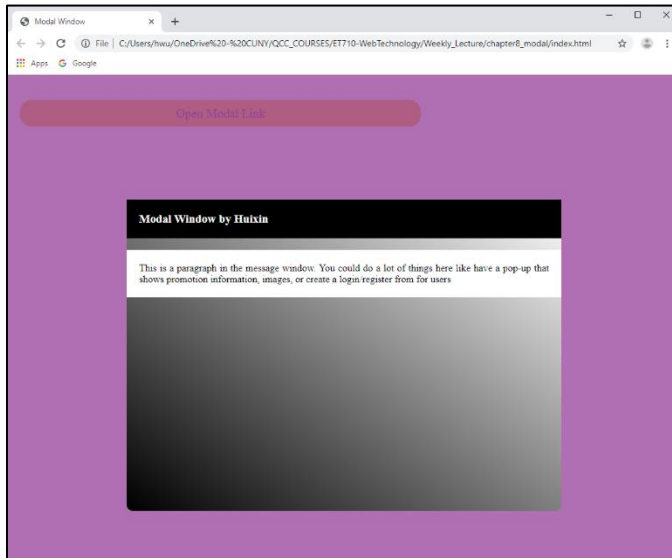
To make my message window more styling, in CSS, we add padding to **<h3>** and **<p>**, and align the text to justify in **<p>**. Remember that to call the **<h3>** and **<p>** without using a class name, we will need to call the division that contains them first and then the element, for example:

.msgText h3

```
.msgText h3{
padding: 20px; background-color: black; color: white; }

.msgText p{
padding: 20px; text-align: justify; background-color: white;}
```

Refreshing the internet browser, the modal window should look as:



We can also add an image to the message window:

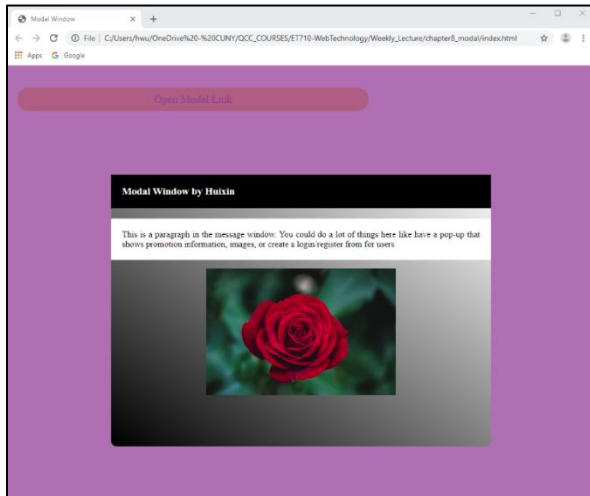
```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Modal Window</title>
    <link rel="stylesheet" href="modal.css" type="text/css">
  </head>
  <body>
    <a href="#openModal" class="linkModel">Open Modal Link</a>
    <!-- Modal Window container -->
    <div id="openModal" class="modalWindow">
      <div class="msgText">
        <h3>Modal Window by Huixin</h3>
        <p>This is a paragraph in the message window. You could do a lot of
        things here like have a pop-up that shows promotion information, images,
        or create a login/register form for users</p>
        
      </div>
    </div>
  </body>
</html>
```

HTML

To style the image in the message window, we can make the width of the image to 50% of the message window width, and center the image by setting the margin-left or margin-right to 25%:

```
.msgText img{
  width: 50%;
  height: auto;
  margin-left: 25%;
}
```

Refreshing the internet browser, the modal window looks as:



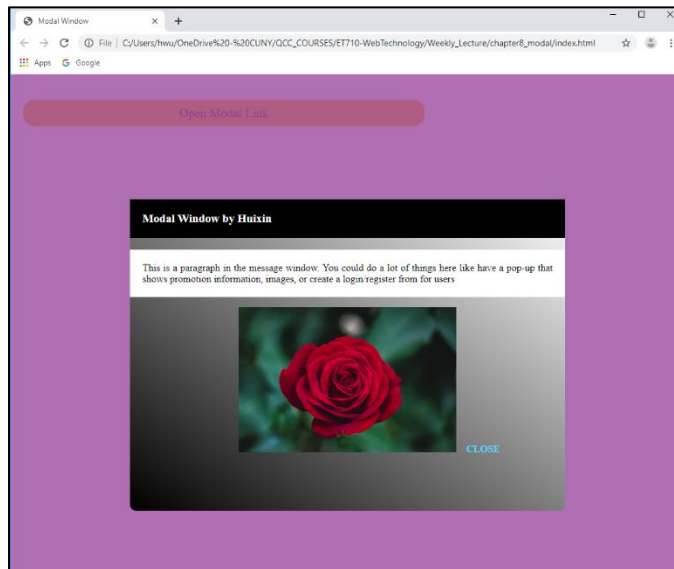
The next thing to add to the message window **msgText** is a link to close the modal window. For this, we can use an anchor **<a>** tag and link it to the top of the web app using: **href="#"**. We can name it as **class="modal-close"** for CSS attributes.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Modal Window</title>
    <link rel="stylesheet" href="modal.css" type="text/css">
  </head>
  <body>
    <a href="#openModal" class="linkModel">Open Modal Link</a>
    <!-- Modal Window container -->
    <div id="openModal" class="modalWindow">
      <div class="msgText">
        <h3>Modal Window by Huixin</h3>
        <p>This is a paragraph in the message window. You could do a lot of
          things here like have a pop-up that shows promotion information,
          images, or create a login/register form for users</p>
        
        <a href="#" class="modal-close">CLOSE</a>
      </div>
    </div>
  </body>
```

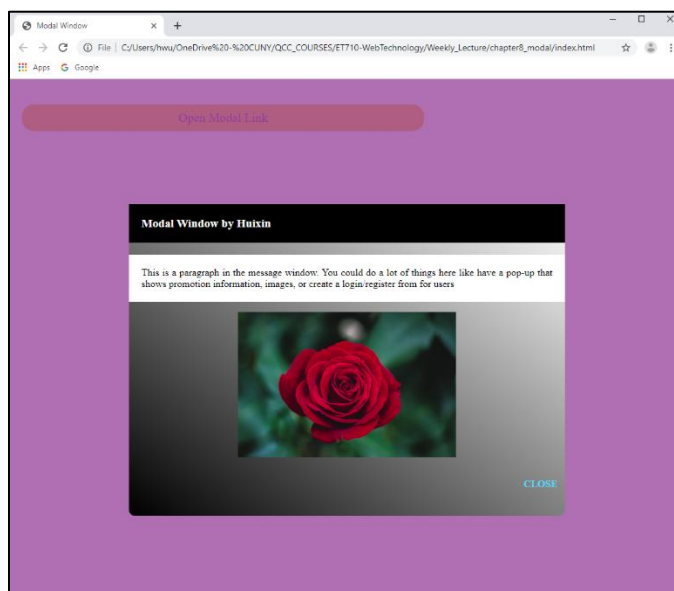
HTML

```
.modal-close {
  color: #00d9ff;
  padding: 12px;
  font-weight: bold;
  text-align: center;
  text-decoration: none; }
```

Refreshing the internet browser, the webpage looks as:



We can also add two break lines **
** in between the image and the CLOSE link to move the CLOSE link to the next line. If we want to have the CLOSE link in the right side, we can use **float:right;** attribute.



Once we have the modal window set, we can set the **opacity** attribute to it. The idea of using the **opacity** attribute is to make the modal window to invisible, **opacity:0;** , when the web app is loaded, then when the modal link is

clicked, the modal window will become visible, **opacity: 1;** . To set this attribute, we add **opacity:0;** in the CSS of the modal window **.modalWindow**

```
.modalWindow {  
  position: fixed; top: 0;  
  right: 0;  
  bottom: 0;  
  left: 0;  
  background-color: rgba(160,80,160,0.8);  
→ opacity: 0;  
}
```

The other setting is to create a hover effect to the modal window to open to modal window using **opacity: 1;**

```
.modalWindow:hover{  
  opacity: 1;  
}
```

If we run the app, the message window will not close when we click on the CLOSE link. This happens because we did not set the **pointer-events** to **target** an element that has the CLOSE link. For this, we can include **pointer-events:none;** to the element that has the anchor tag **<a>** beneath it. In this activity, we target the modal window **.modalWindow** and when click on the CLOSE link, the model window will close, that is why we need to change **.modalWindow:hover** to **.modalWindow:target** and add a **pointer-events: auto;** which means that the any anchor **<a>** tag under **.modalWindow** will react to the pointer events. In other words, when we click on the CLOSE link, the anchor **<a>** tag will take us to the beginning to the web app as states with **href="#"**

```
.modalWindow {  
  position: fixed; top: 0;  
  right: 0;  
  bottom: 0;  
  left: 0;  
  background-color: rgba(160,80,160,0.8);  
  opacity: 0;  
  pointer-events: none;  
}  
  .modalWindow:target{  
    opacity: 1;  
    pointer-events: auto;  
  }
```

Diagram annotations:

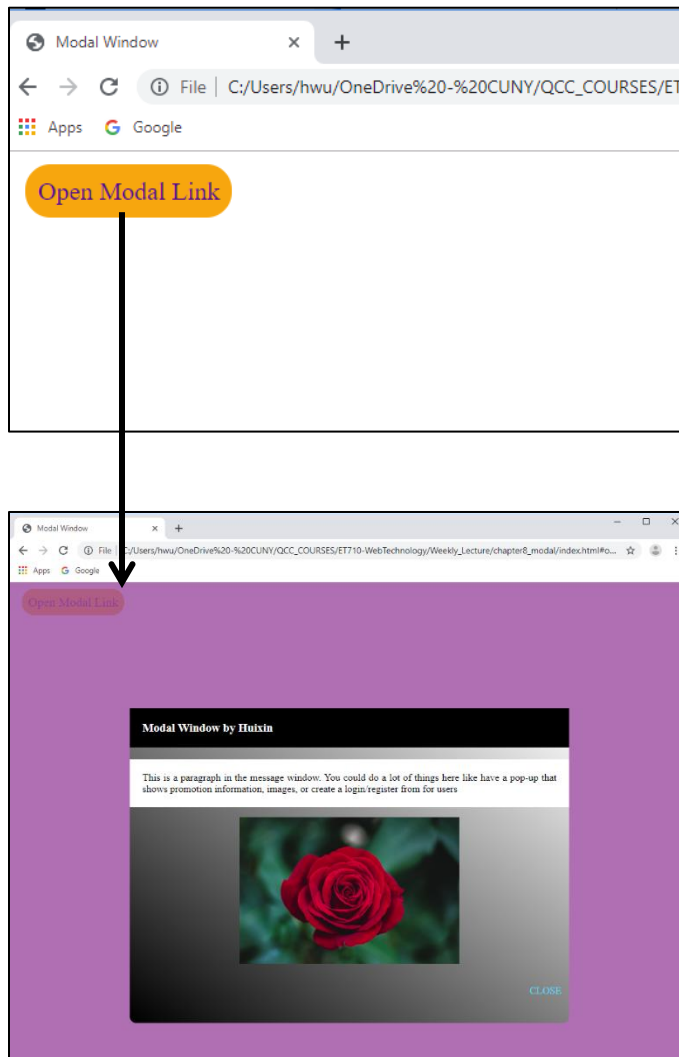
- Box: "Add this line" points to **pointer-events: none;**
- Box: "Change hover to target" points to **.modalWindow:target{**
- Box: "Add this line" points to **pointer-events: auto;**

We can also add some animation to our modal window, for example, we can make the modal window to **ease-in** when it opens.

Add this line →

```
.modalWindow:target{  
  opacity: 1;  
  pointer-events: auto;  
  transition: ease-in 0.5s;  
}
```

Refreshing the internet browser, the modal window looks as:



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

Some material compiled from www.lynda.com (May 2018), www.w3schools.com (2020),
www.coursera.org (2020)

IMPORTANT NOTE

The materials used in this manual have the author's rights and are for educational use only.