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 site 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 <**`span>`**, 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).

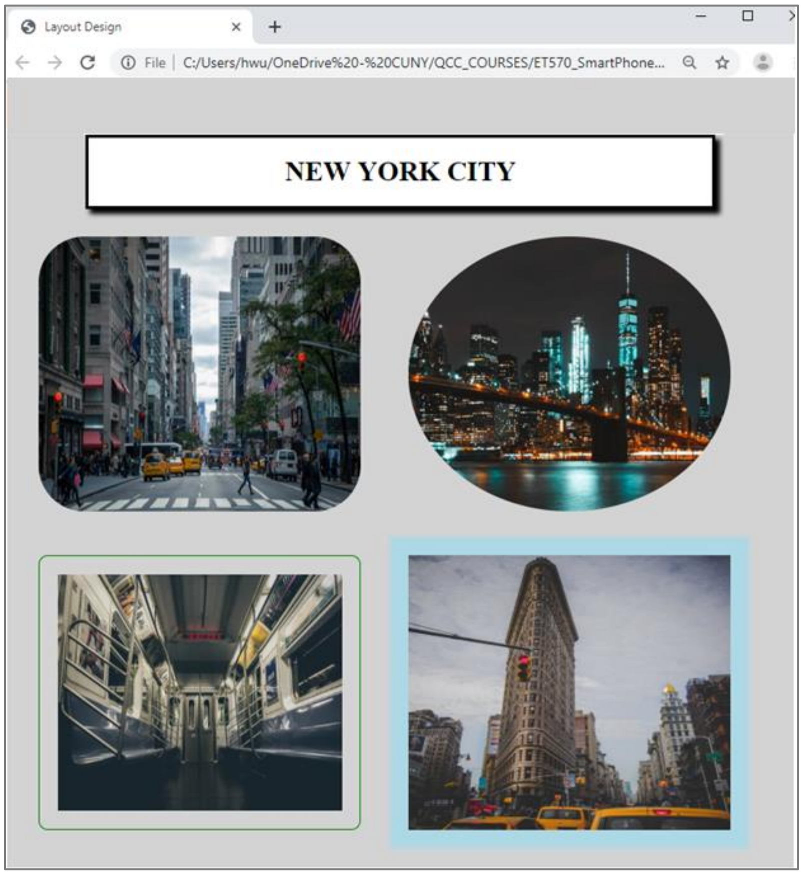
The CSS `float` property specifies how an element should float. The `float` property can have one of the following values:

- **`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 paren**t**

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.


Layout Design
File | C:/Users/hwu/OneDrive%20-%20CUNY/QCC_COURSES/ET570_SmartPhone...
NEW YORK CITY

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>
```

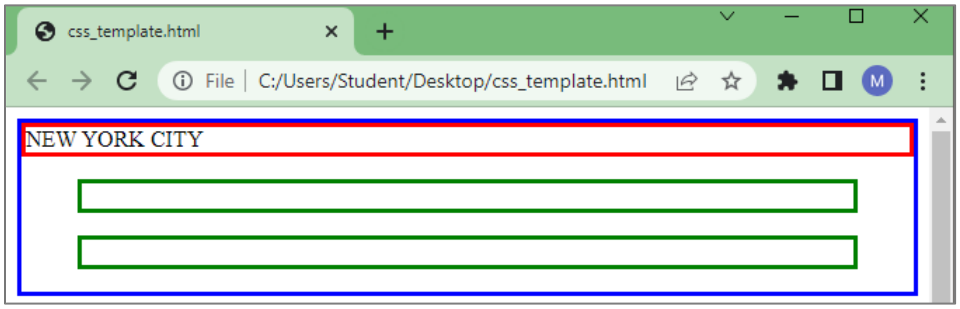
A complete HTML file looks as the following:

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:

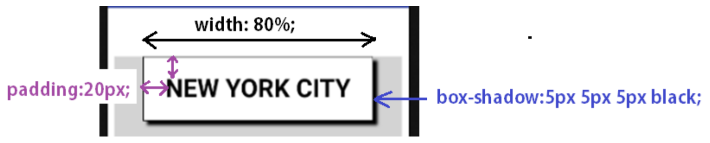
```
.container{border: solid blue;}
.header{border: solid red;}
.images{border:solid green;}
```



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:

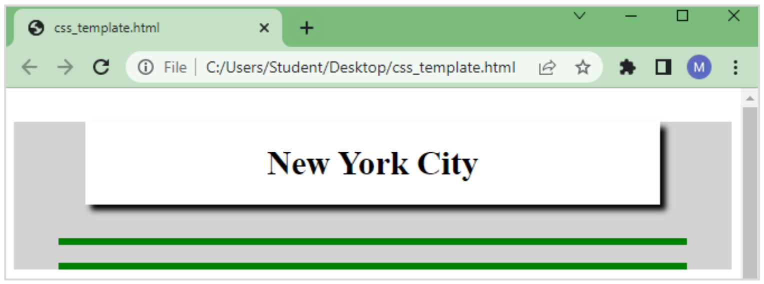


```
.header{
  width: 80%;
  margin: auto;
  box-shadow: 5px 5px 5px black;
  padding: 20px;
  font-size: 30px;
  font-weight: 600;
  text-align: center;
  background-color: white;
}
```

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:

index.css

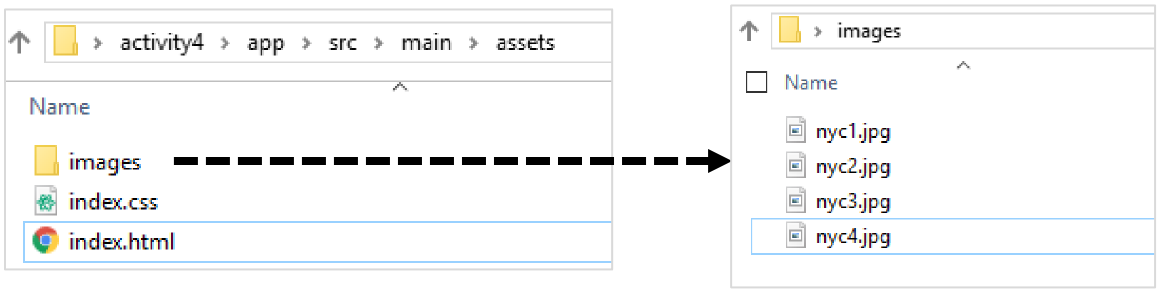
```
*{ box-sizing: border-box; }
.container{
  background-color: lightgray;
  width: 100%;
  height: auto;
  padding-bottom: 5%;
}
.header{
  width: 80%;
  margin: 1em auto;
  box-shadow: 5px 5px 5px black;
  padding: 20px;
  font-size: 30px;
  font-weight: 600;
  text-align: center;
  background-color: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

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.

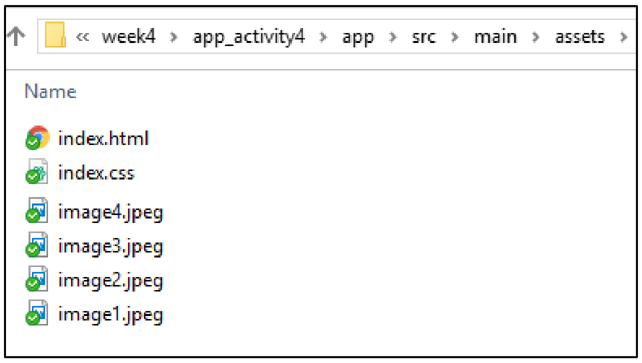
We can download four images and save them in assets folder. If we want to organize our assets folder, we can create a new folder, which we can name it *images,* inside our project folder place all the images:



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">
                <img src="images/nyc1.jpg" />
                <img src="images/nyc2.jpg" />
        </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">
                <img src="nyc1.jpg" />
                <img src="nyc2.jpg" />
        </figure>
```

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

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">
        <img class="img1" src="images/nyc1.jpg">
        <img class="img2" src="images/nyc2.jpg">
      </figure>
      <figure class="images">
        <img class="img3" src="images/nyc3.jpg">
        <img class="img4" src="images/nyc4.jpg">
      </figure>
    </div>
  </body>
</html>
```

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:

index.css

```
*{ 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%;
}
```

The images should look as the following:

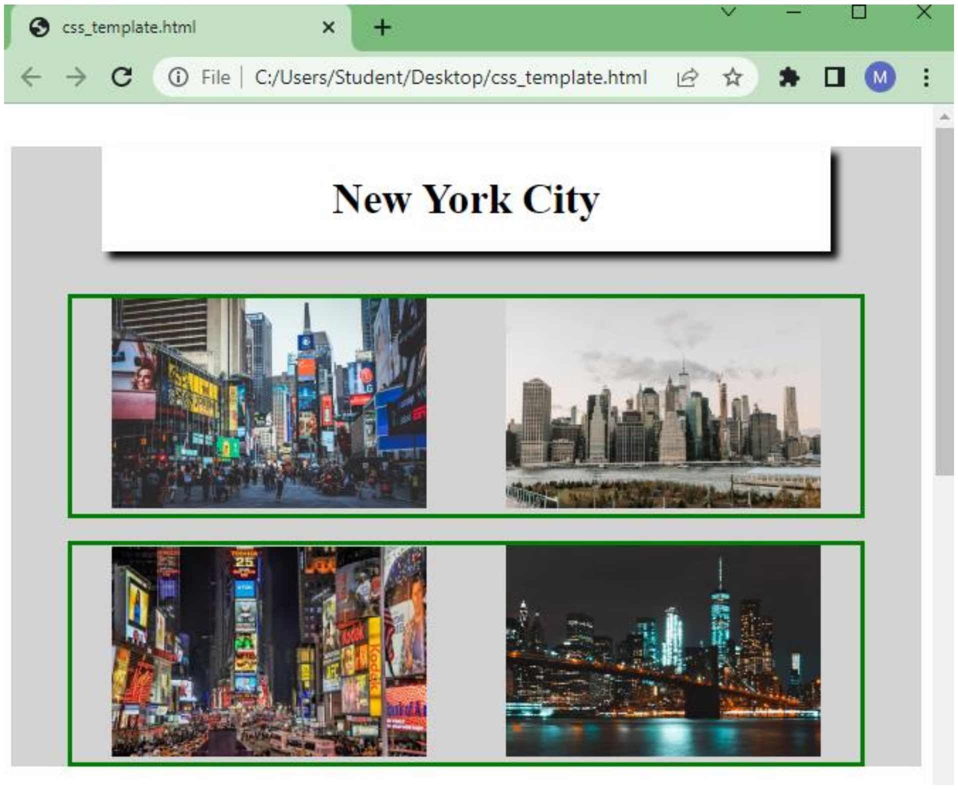


Image format

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.

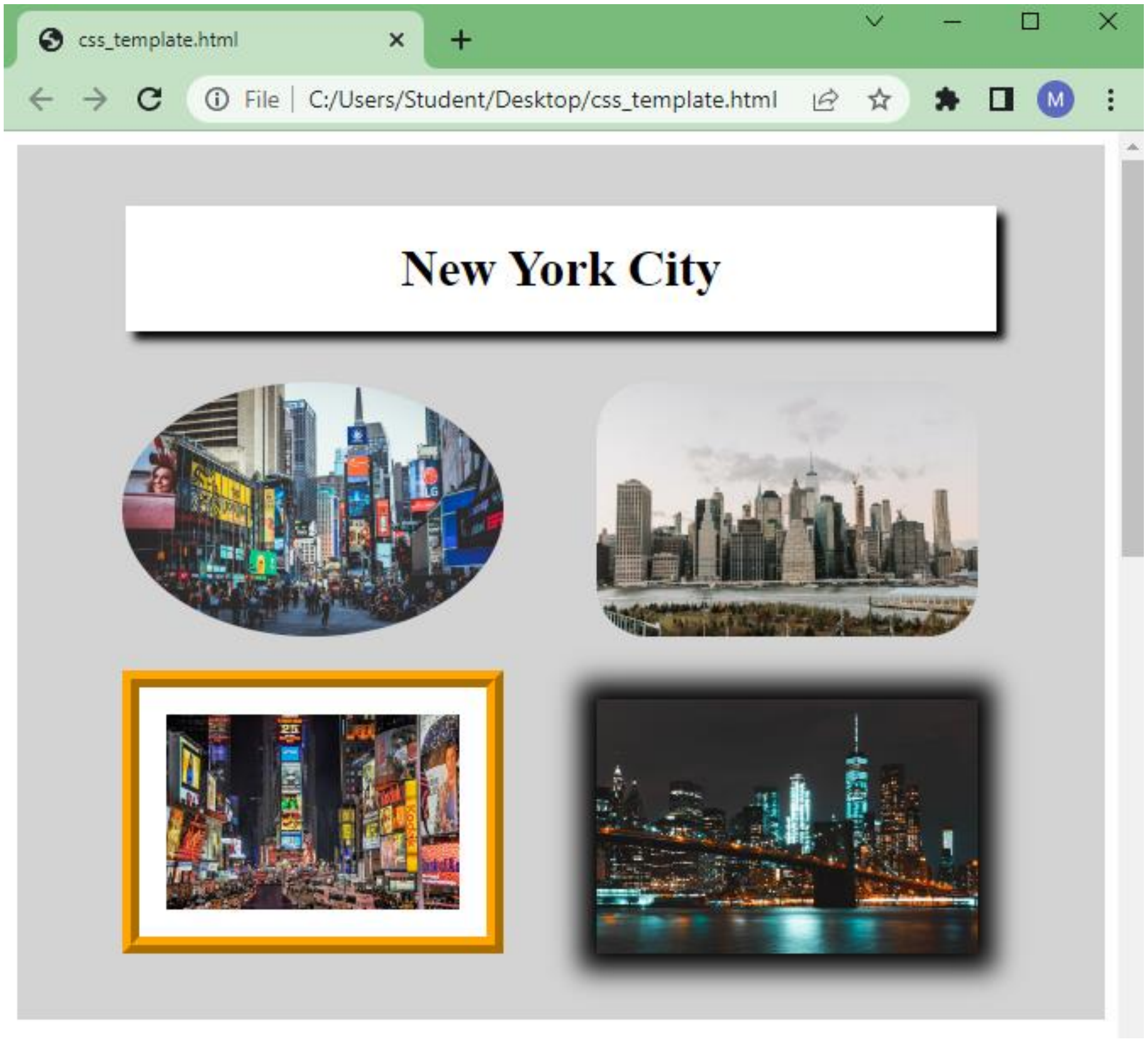
To complete this activity, we can add the css code within the HTML code using **`style`** as the following:

```
<figure class="images">
  <img class="img1" src="images/nyc1.jpg" style="border-radius: 50%;">
  <img class="img2" src="images/nyc2.jpg" style="border-radius: 30px;">
</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:


css_template.html
File
C:/Users/Student/Desktop/css_template.html
New York City