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

HTML

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

CSS

```
.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;}
```

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;
}
```

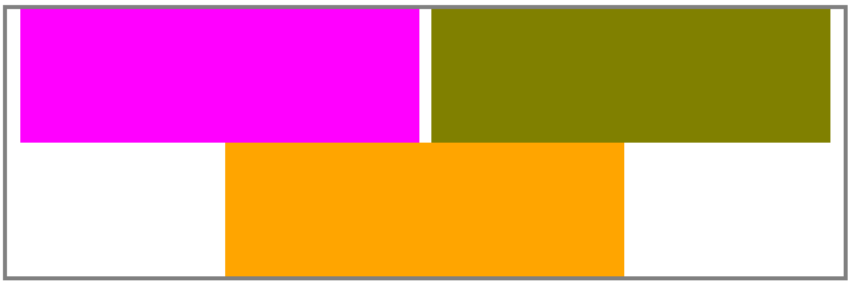
Flex-wrap property

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;
}
```

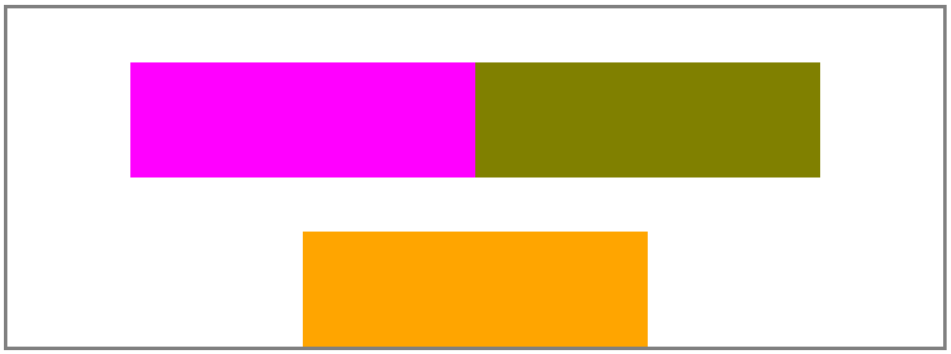
align-content property

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.

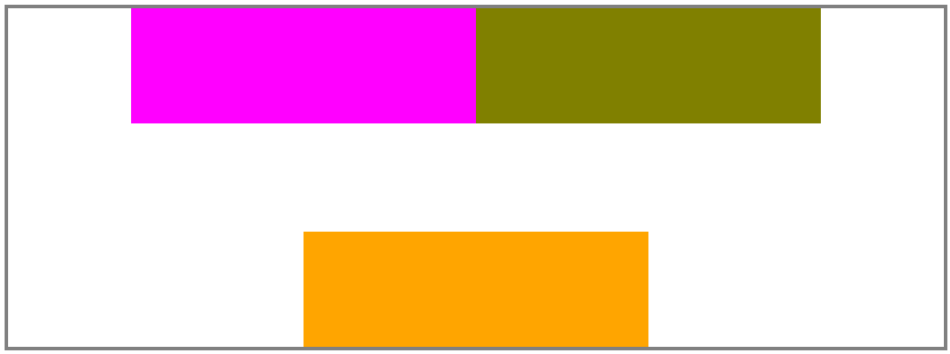
Note: There must be multiple lines of items for this property to have any effect!

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

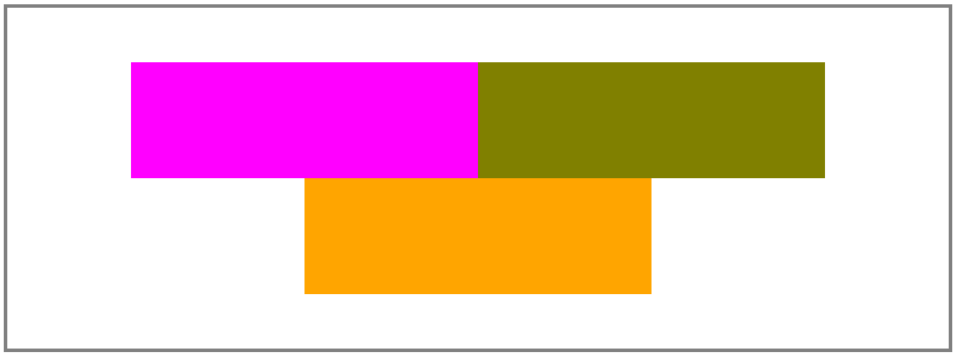
```
.flex_container{
      border: solid gray;
      width: 80%;
      height: 300px;
      margin: 0 auto;
      display: flex;
      flex-direction: row;
      justify-content: center;
      align-items:flex-end;
      flex-wrap: wrap;
    }

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



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



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

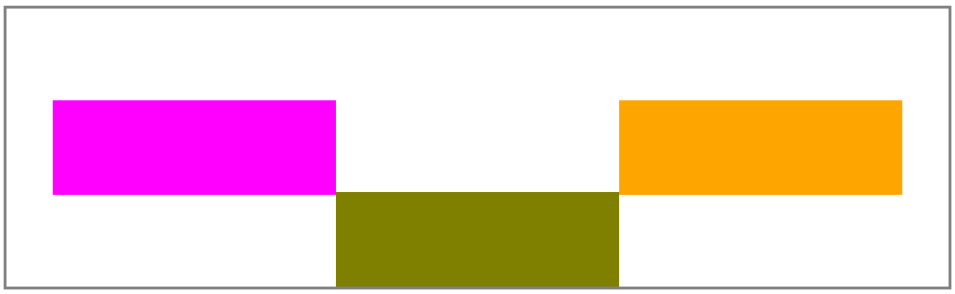


align-self property

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.

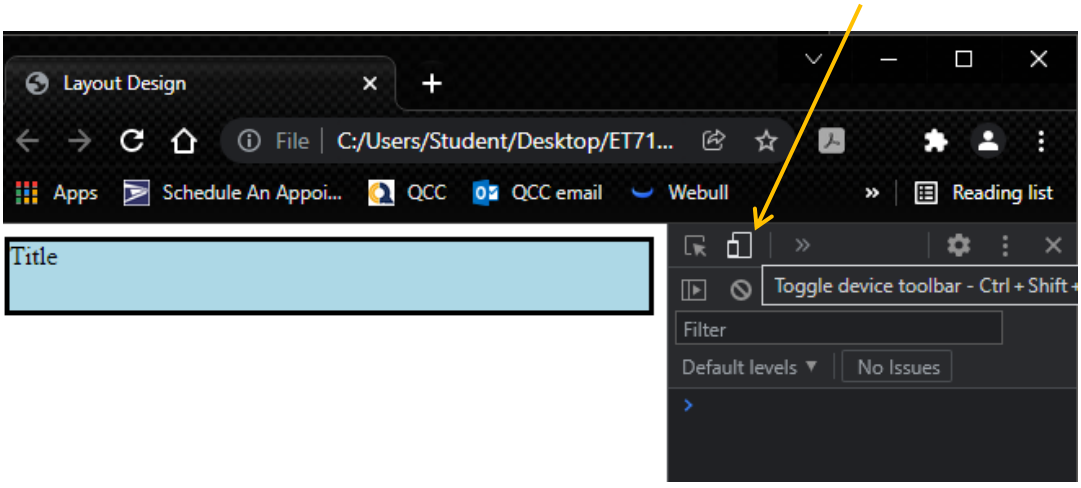
When using @media, instead of changing styles when the width gets *smaller* than 800px, we should change the design when the width gets *larger* than 800px. This will make our design Mobile First. The syntax code will look as:

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

}
```

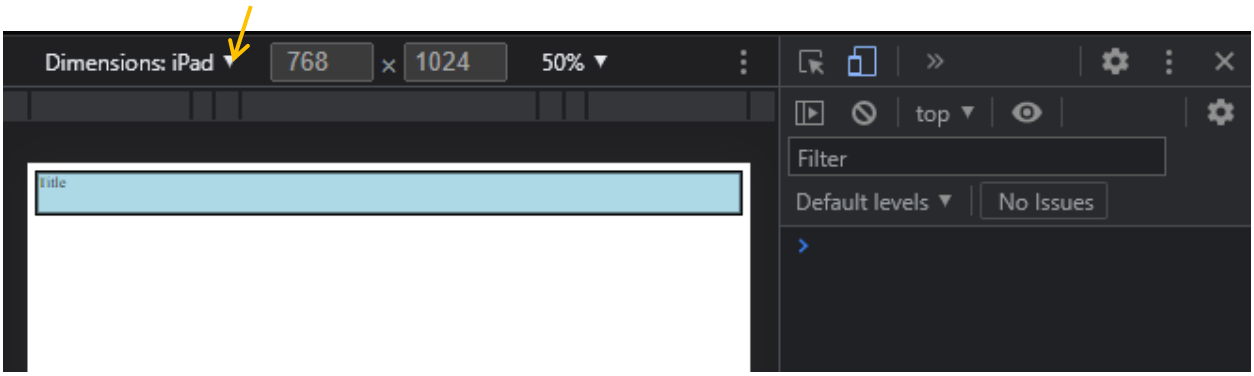
Between the curly brackets should go the CSS attributes of the elements that will changed when the screen has the width of 800px or greater.

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`**

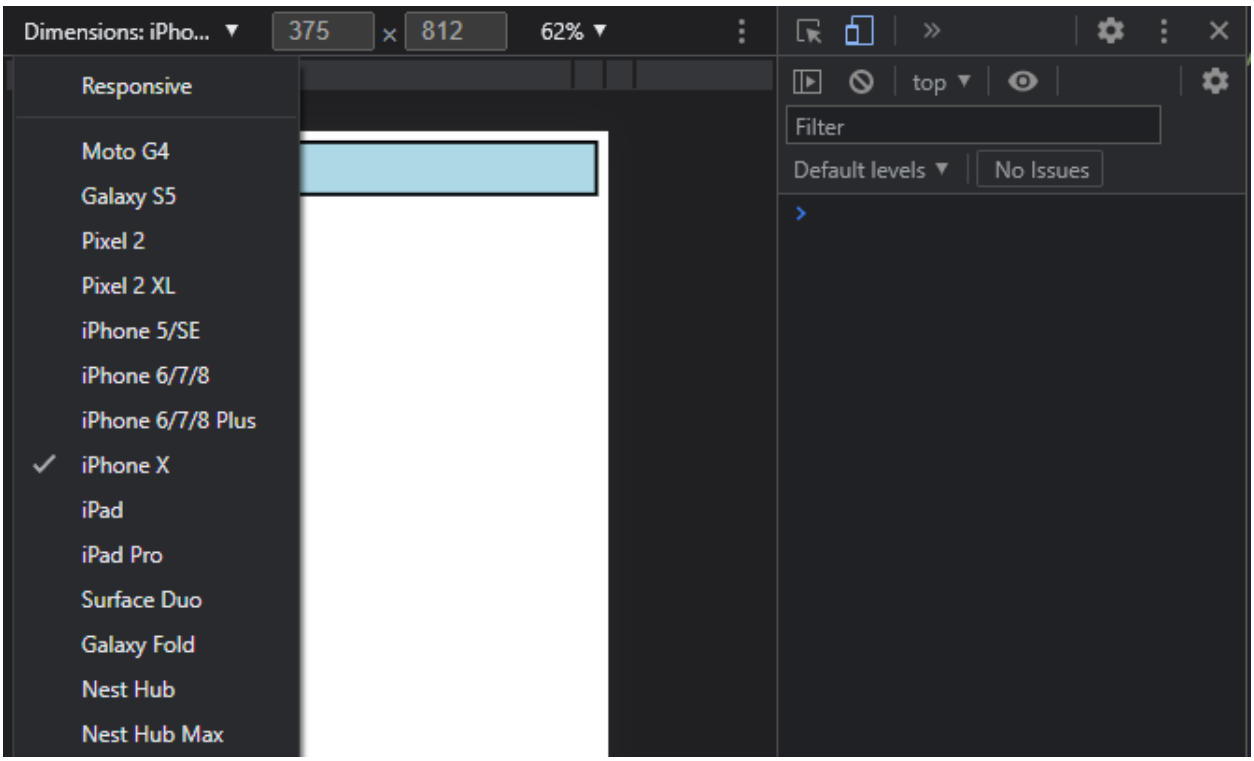
Also, if we are designing from mobile view first, we can change the browser view to mobile view by using the function key **F12** and then click on the **Toggle device toolbar**



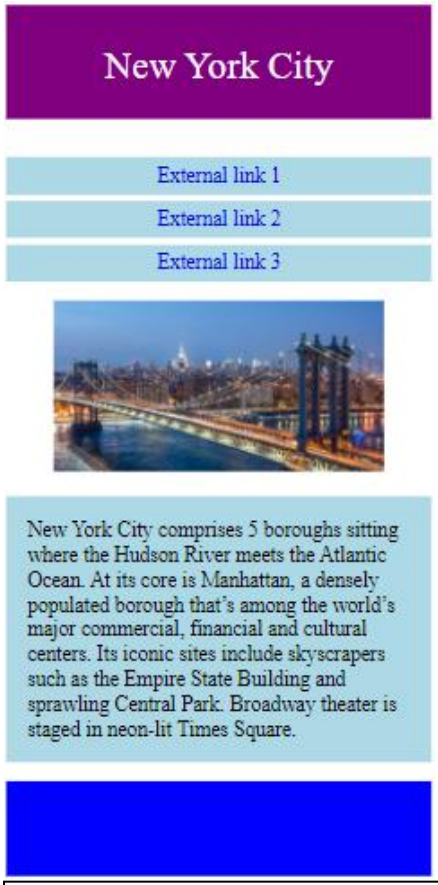
Once clicked on the Toggle device toolbar, we can select the size of the mobile view:



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 of desktop view from 800px and up. Starts designing from smartphone view.



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">
          <img src="images/image4.jpg" alt="NYC image">
        </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>
```

CSS

```
*{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

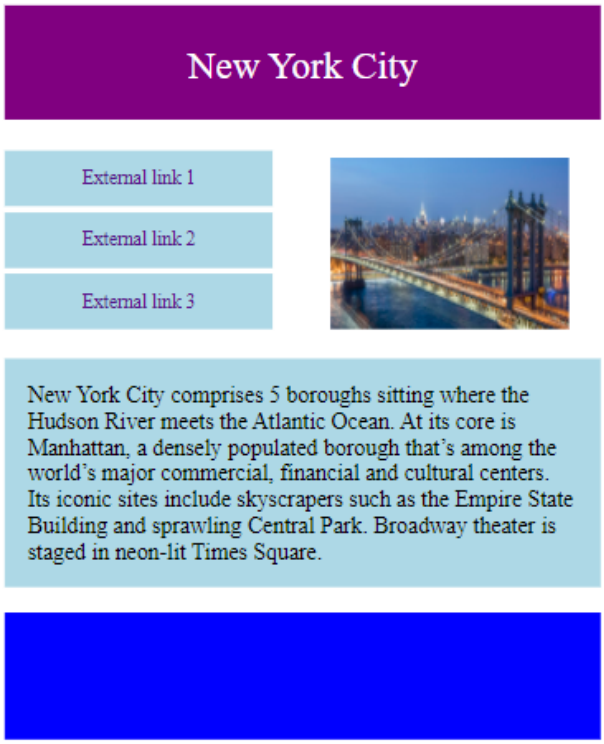
CSS

```
*{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.