Agenda:

- Understanding Box Model in css
- Display and positioning in css

In the last module of css, we have learnt about the basics of css and about some of its properties. So, in this module we are going to dive into th advance topics of css, starting with the typography in css:

Introduction to Box Model:

We have seen in the last module that we can think of every HTML element as a box and that we can style with css.

So, All elements on a web page are interpreted by the browser as “living” inside of a box. This is what is meant by the box model.

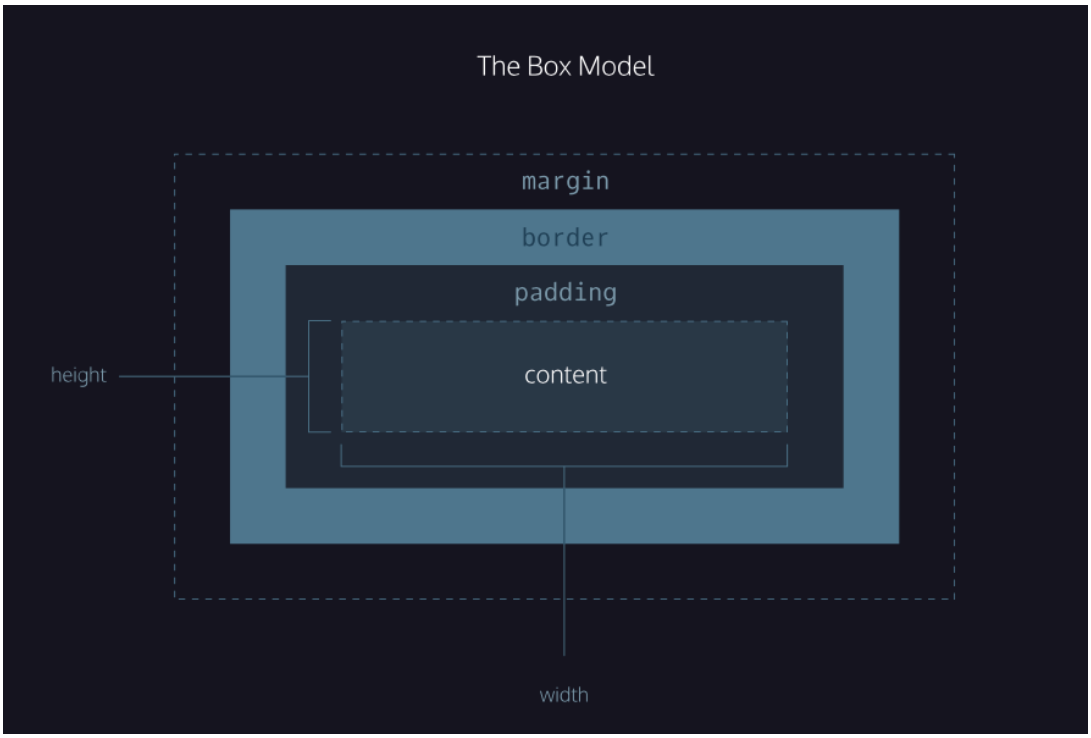
Browsers load HTML elements with default position values. This often leads to an unexpected and unwanted user experience while limiting the views yc can create.

So, In this module, you will learn about the *box model*, an important concept to understand how elements are positioned and displayed on a website.

The Box Model:

The box model comprises the set of properties that define parts of an element that take up space on a web page. The model includes the content area size (*width* and *height*) and the element’s *padding*, *border*, and *margin*. The properties include:

1. `width` and `height` : The width and height of the content area.
2. `padding` : The amount of space between the content area and the border.
3. `border` : The thickness and style of the border surrounding the content area and padding.
4. `margin` : The amount of space between the border and the outside edge of the element.



Now, let’s take a look at each property of the Box Model in detail:

Height and Width:

An element’s content has two dimensions: a height and a width. By default, the dimensions of an HTML box are set to hold the raw contents of the box.

The CSS `height` and `width` properties can be used to modify these default dimensions.

The most popular ways to specify the size of a box are to use pixels, percentages, or ems. Traditionally, pixels have been the most popular metho because they allow designers to accurately control their size.

When you use percentages, the size of the box is relative to the size of the browser window or, if the box is encased within another box, it is percentage of the size of the containing box.

When you use ems, the size of the box is based on the size of text within it. Designers have recently started to use percentages and ems more f measurements as they try to create designs that are flexible across devices which have different-sized screens.

Consider the example below:

HTML:

```
<div>
<p>The Moog company pioneered the commercial
manufacture of modular voltage-controlled analog synthesizer systems
 in the early 1950s.</p>
</div>
```

CSS:

```
div.box {
  height: 300px;
  width: 300px;
  background-color: #bbbbaa;}
p{
    height: 75%;
    width: 75%;
    background-color: #0088dd;}
```

In the example above, you can see that a containing
element is used which is 300 pixels wide by 300 pixels high. Inside of this is a paragraph that is 75% of the width and height of the containing element. This means that the size of the paragraph is 225 pixels wide by 225 pixels high.

Borders:

A *border* is a line that surrounds an element, like a frame around a painting. Borders can be set with a specific `width`, `style`, and `color`:

- `width` —The thickness of the border. A border's thickness can be set in pixels or with one of the following keywords: `thin`, `medium`, or `thick`.
- `style` —The design of the border. Some of these styles include: `none`, `dotted`, and `solid`.
- `color` —The color of the border. Web browsers can render colors using a few different formats.

Consider the example below:

```
p {
  border: 3px solid coral;
}
```

In the example above, the border has a width of 3 pixels, a style of `solid`, and a color of `coral`. All three properties are set in one line of code.

The default border is `medium none color`, where `color` is the current color of the element. If `width`, `style`, or `color` are not set in the CS file, the web browser assigns the default value for that property.

To understand this, consider the example below:

```
p.content-header {
  height: 80px;
  width: 240px;
  border: solid coral;
}
```

In this example, the border style is set to `solid` and the color is set to `coral`. The width is not set, so it defaults to `medium`.

Border Radius:

Ever since we revealed the borders of boxes, you may have noticed that the borders highlight the true shape of an element's box: square.

But with the help of css, we can modify the corners of an element's box.

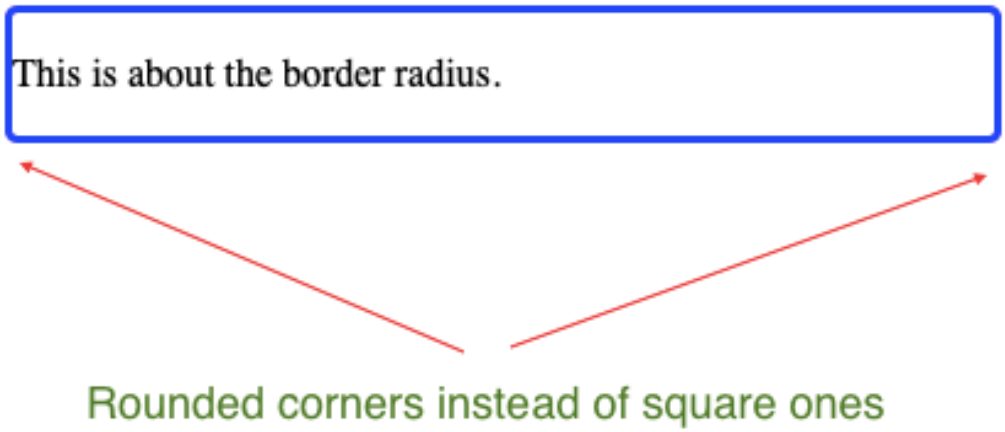
We can modify it with the help of `border-radius` property.

Consider the example below:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
          div.container {
                border: 3px solid blue;
                border-radius: 5px;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <p>This is about the border radius.</p>
    </div>
  </body>
</html>
```

In the above code we have used internal css to style the HTML elements.

Output:



Look at the corners of the box of div element, these are not square but are rounded by the amount specified in the border-radius property.

You can create a border that is a perfect circle by first creating an element with the same width and height, and then setting the radius equal to half th width of the box, which is 50%.

Consider the example below:

```
div.container {
  height: 60px;
  width: 60px;
  border: 3px solid blue;
  border-radius: 50%;
}
```

This will convert the shape of the div element's box into a circle.

Padding:

The space between the contents of a box and the borders of a box is known as *padding*. Padding is like the space between a picture and the fram surrounding it. In CSS, you can modify this space with the `padding` property.

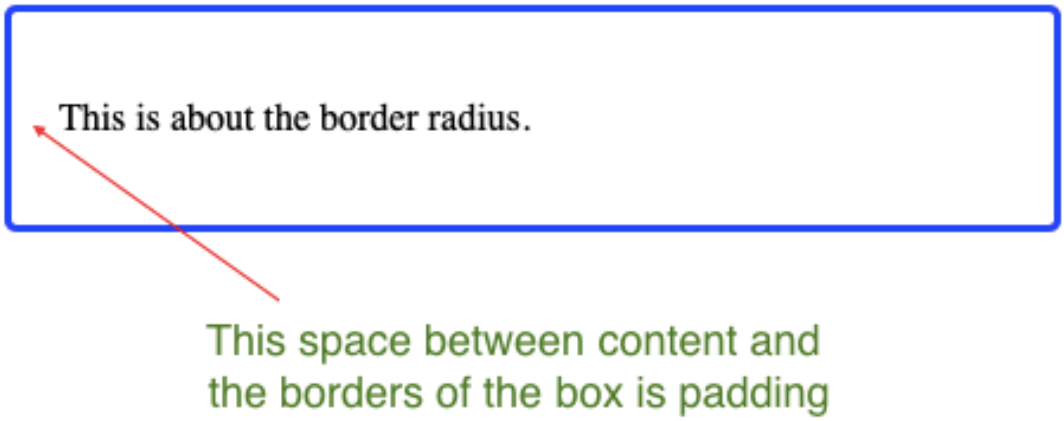
Consider the example below:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
          div.container {
```

```
            width: 400px;
                border: 3px solid blue;
                border-radius: 5px;
            padding: 20px;
        }
    </style>
  </head>
  <body>
    <div class="container">
      <p>This is about the border radius.</p>
    </div>
  </body>
</html>
```

The code in this example puts 20 pixels of space between the content of the paragraph (the text) and the borders, on all four sides.

Output:



The `padding` property is often used to expand the background color and make the content look less cramped.

If you want to be more specific about the amount of padding on each side of a box's content, you can use the following properties:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

Each property affects the padding on only one side of the box's content, giving you more flexibility in customization.

Consider the code below:

```
p.content-header {
  border: 3px solid fuchsia;
  padding-bottom: 10px;
}
```

In the example above, only the bottom side of the paragraph's content will have a `padding` of 10 pixels.

Margin:

The fourth and final component of the box model is *margin*.

Margin refers to the space directly outside of the box. The `margin` property is used to specify the size of this space.

Consider the example below:

```
p {
  border: 1px solid aquamarine;
  margin: 20px;
}
```

The code in the example above will place 20 pixels of space on the outside of the paragraph's box on all four sides. This means that other HTM elements on the page cannot come within 20 pixels of the paragraph's border.

If you want to be even more specific about the amount of margin on each side of a box, you can use the following properties:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

Each property affects the margin on only one side of the box, providing more flexibility in customization.

Consider the example below:

```
p {
  border: 3px solid DarkSlateGrey;
  margin-right: 15px;
}
```

In the example above, only the right side of the paragraph's box will have a margin of 15 pixels. It's common to see margin values used for a specific sic of an element.

Centering Contents with Margin property:

The `margin` property also lets you center content. However, you must follow a few syntax requirements. Take a look at the following example:

```
div.headline {
  width: 400px;
  margin: 0 auto;
}
```

In the example above, `margin: 0 auto;` will center the divs in their containing elements. The 0 sets the top and bottom margins to 0 pixel The `auto` value instructs the browser to adjust the left and right margins until the element is centered within its containing element.

In order to center an element, a width must be set for that element. Otherwise, the width of the div will be automatically set to the full width of i containing element, like the `<body>`, for example. It's not possible to center an element that takes up the full width of the page, since the width of th page can change due to display and/or browser window size.

Margin Collapse:

As you have seen, padding is space added inside an element's border, while margin is space added outside an element's border. One addition difference is that top and bottom margins, also called vertical margins, *collapse*, while top and bottom padding does not.

Horizontal margins (left and right), like padding, are always displayed and added together. For example, if two divs with ids `#div-one` and `#div two`, are next to each other, they will be as far apart as the sum of their adjacent margins.

```
#img-one {
  margin-right: 20px;
}

#img-two {
  margin-left: 20px;
}
```

In this example, the space between the `#img-one` and `#img-two` borders is 40 pixels. The right margin of `#img-one` (`20px`) and the left marg of `#img-two` (`20px`) add to make a total margin of 40 pixels.

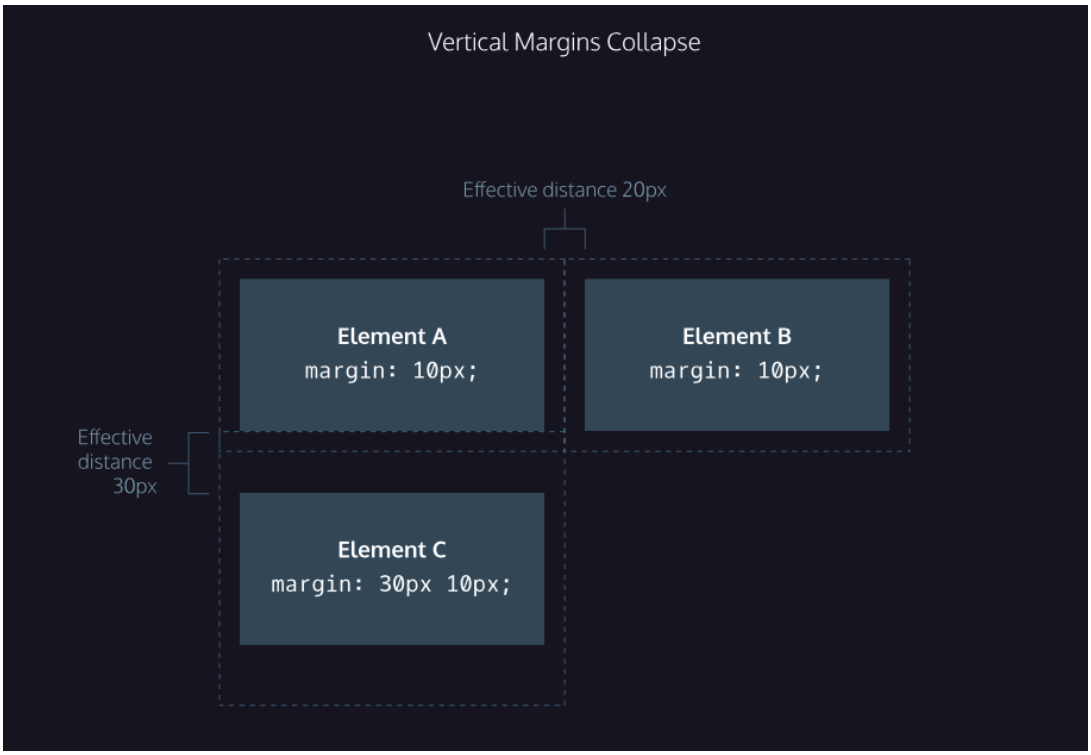
Unlike horizontal margins, vertical margins do not add. Instead, the larger of the two vertical margins sets the distance between adjacent elements.

```
#img-one {
  margin-bottom: 30px;
}

#img-two {
```

```
    margin-top: 20px;
}
```

In this example, the vertical margin between the `#img-one` and `#img-two` elements is 30 pixels. Although the sum of the margins is 50 pixels, th margin collapses so the spacing is only dependent on the `#img-one` bottom margin.



Till now as we have learned about the Box model in css , now let's start learning about the layouts, positioning and flow of elements.

Display and Positioning in css:

Understanding Flow:

A browser will render the elements of an HTML document that has no CSS from left to right, top to bottom, in the same order as they exist in th document. This is called the *flow* of elements in HTML.

In addition to the properties that it provides to style HTML elements, CSS includes properties that change how a browser *positions* elements. Thes properties specify where an element is located on a page, if the element can share lines with other elements, and other related attributes.

In this section, you will learn five properties for adjusting the position of HTML elements in the browser:

- `position`
- `display`
- `z-index`
- `float`
- `clear`

Each of these properties will allow us to position and view elements on a web page.

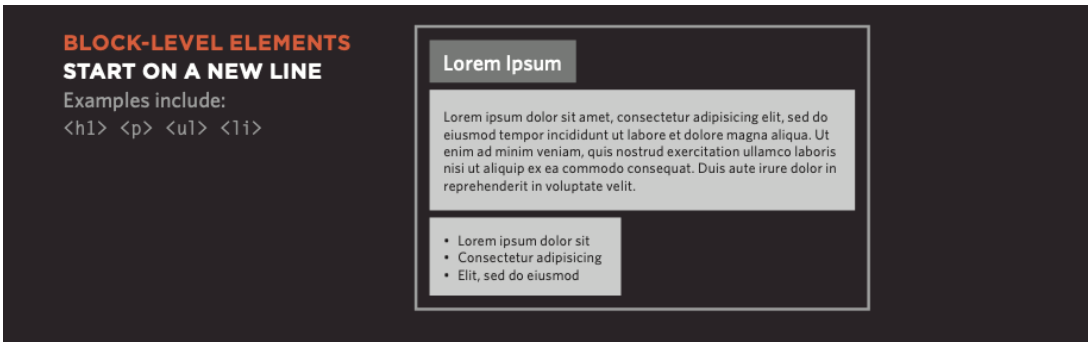
Understanding Positioning:

Let's first understand the blocks in css:

Block Level Elements:

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.

Block-level boxes start on a new line and act as the main building blocks of any layout, while inline boxes flow between surrounding text. Yo can control how much space each box takes up by setting the width of the boxes (and sometimes the height, too). To separate boxes, you ca useborders, margins, padding, and background colors.



Notice the block-level elements in the image above take up their own line of space and therefore don't overlap each other. In the image above, you ca see block-level elements also consistently appear on the left side of the browser. This is the default *position* for block-level elements.

The default position of an element can be changed by setting its `position` property. The `position` property can take one of five values:

- `static` - the default value (it does not need to be specified)
- `relative`
- `absolute`
- `fixed`
- `sticky`

Position: Relative

One way to modify the default position of an element is by setting its `position` property to `relative` .

This value allows you to position an element *relative* to its default static position on the web page.

```
.green-box {
  background-color: green;
  position: relative;
}
```

Although the code in the example above instructs the browser to expect a relative positioning of the `.green-box` element, it does not specify whe the `.green-box` element should be positioned on the page. This is done by accompanying the `position` declaration with one or more of th following *offset properties* that will move the element away from its default static position:

- `top` - moves the element down from the top.
- `bottom` - moves the element up from the bottom.
- `left` - moves the element away from the left side (to the right).
- `right` - moves the element away from the right side (to the left).

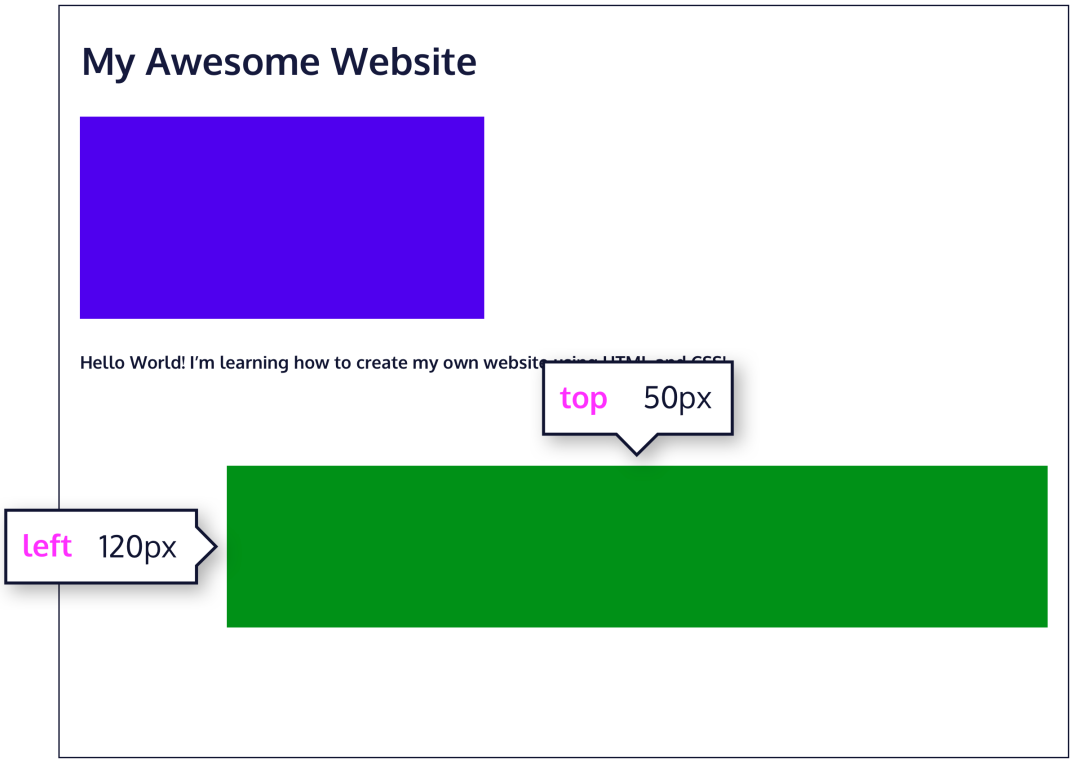
You can specify values in pixels, ems, or percentages, among others, to dial in exactly how far you need the element to move. It's also important to no that offset properties will not work if the element's `position` property is the default `static` .

```
.green-box {
  background-color: green;
  position: relative;
  top: 50px;
  left: 120px;
}
```

In the example above, the element of `green-box` class will be moved down 50 pixels, and to the right 120 pixels, from its default static position.

The green-box class is applied in HTML to the green box only.

Output:



Position: Absolute

Another way of modifying the position of an element is by setting its position to `absolute` .

When an element's position is set to `absolute` , all other elements on the page will ignore the element and act like it is not present on the page. Th element will be positioned relative to its closest positioned parent element, while offset properties can be used to determine the final position from there.

Consider the example below:

```
.paragraph {
    position: absolute;
    top: 300px;
    right: 0px;
}
```

Output:

In the above example, we have applied the paragraph class on the paragraph that is shown with blue background in the image above.

You can see that the paragraph is displaced from its static position at the top left corner of its parent container. It has offset property declaration of `top: 300px;` and `right: 0px;` , positioning it 300 pixels down, and 0 pixels from the right side of the page.

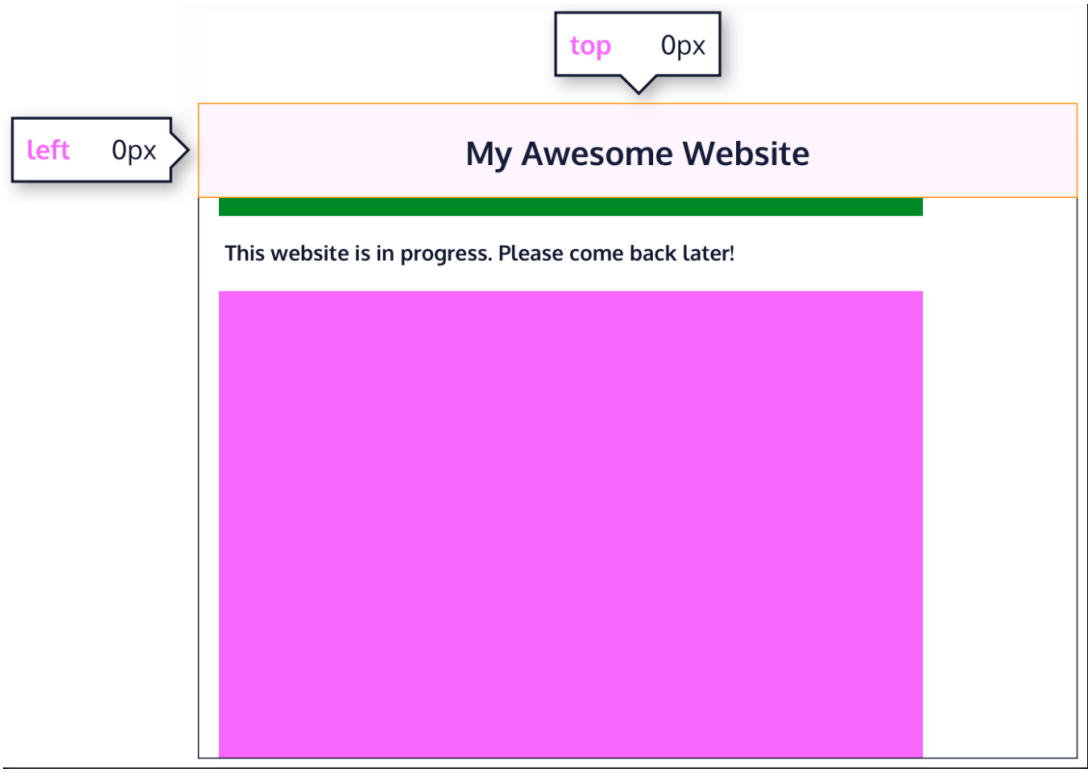
Position: Fixed

When an element's position is set to `absolute` , as we have seen just above, the element will scroll with the rest of the document when a user scrolls.

We can *fix* an element to a specific position on the page (regardless of user scrolling) by setting its position to `fixed` , and accompanying it with th familiar offset properties `top` , `bottom` , `left` , and `right` .

```
.title {
  position: fixed;
  top: 0px;
  left: 0px;
}
```

Output:



In the example above, the `.title` element will remain fixed to its position no matter where the user scrolls on the page

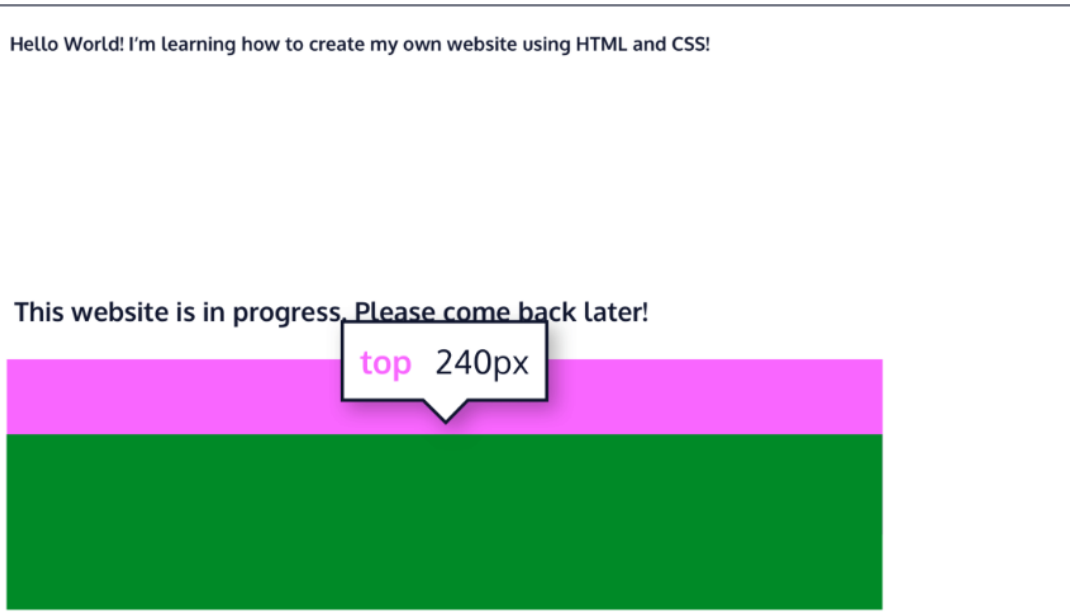
Position: Sticky

Since `static` and `relative` positioned elements stay in the normal flow of the document, when a user scrolls the page (or parent element) thes elements will scroll too. And since `fixed` and `absolute` positioned elements are removed from the document flow, when a user scrolls, thes elements will stay at their specified offset position.

The `sticky` value is another position value that keeps an element in the document flow as the user scrolls, but *sticks* to a specified position as th page is scrolled further. This is done by using the `sticky` value along with the familiar offset properties, as well as one new one.

```
.box-bottom {
  background-color: darkgreen;
  position: sticky;
  top: 240px;
}
```

Output:



In the example above, the `.box-bottom` `<div>` will remain in its relative position, and scroll as usual. When it reaches 240 pixels from the top, it w stick to that position until it reaches the bottom of its parent container where it will "unstick" and rejoin the flow of the document.

Z-Index Property:

The `z-index` property controls how far back or how far forward an element should appear on the web page when elements overlap. This can b thought of as the *depth* of elements, with deeper elements appearing behind shallower elements.

When you use relative, fixed, or absolute positioning, boxes can overlap. If boxes do overlap, the elements that appear later in the HTML code sit on tc of those that are earlier in the page.

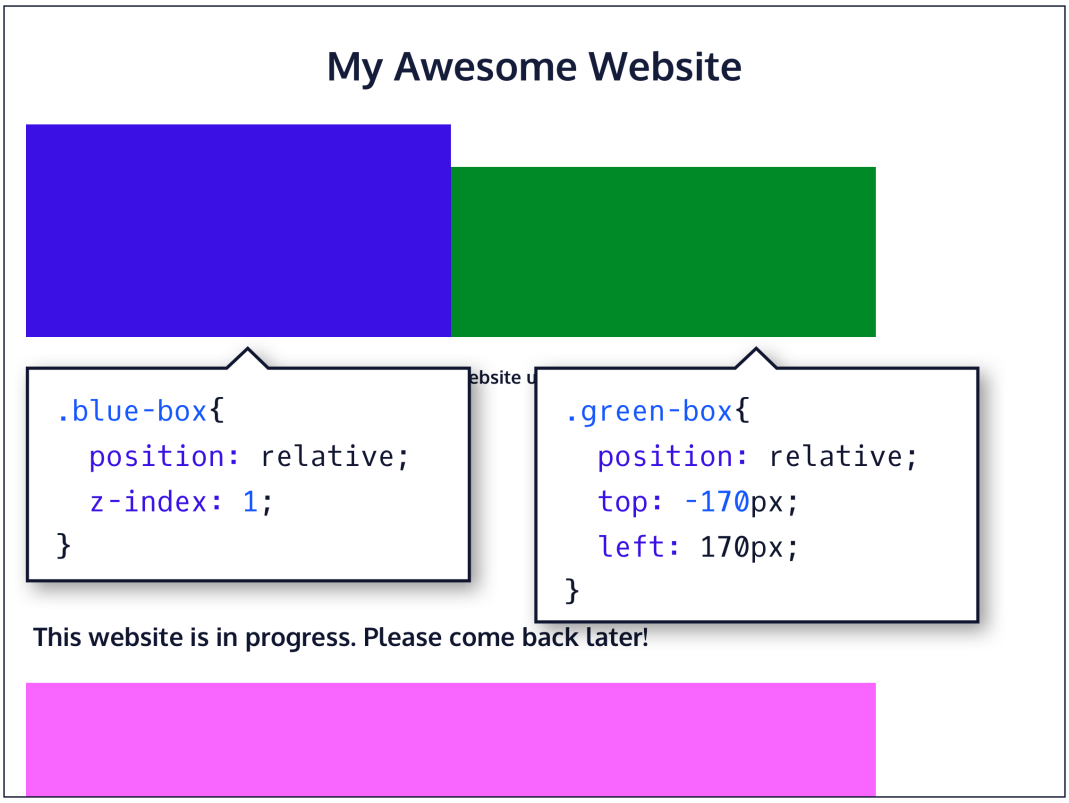
If you want to control which element sits on top, you can use the **z-index** property. Its value is a number, and the higher the number the closer th element is to the front. For example, an element with a **z-index** of **10** will appear over the top of one with a **z-index** of **5**.

Consider the example shown below:

```
.blue-box {
  background-color: blue;
  position: relative;
  z-index: 1;
}

.green-box {
  background-color: green;
  position: relative;
  top: -170px;
  left: 170px;
}
```

Output:



In the example above, we set the `.blue-box` position to `relative` and the z-index to 1. We changed position to `relative`, because the `z-index` property does *not* work on static elements. The z-index of `1` moves the `.blue-box` element forward, because the `z-index` value has n been explicitly specified for the `.green-box` element, which means it has a default `z-index` value of 0.

Visibility:

Elements can be hidden from view with the `visibility` property.

The `visibility` property can be set to one of the following values:

- `hidden` — hides an element.
- `visible` — displays an element.
- `collapse` — collapses an element.

Consider the example shown below:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .future {
      visibility: hidden;
    }
  </style>
</head>
<body>

 <ul>
  <li>Explore</li>
  <li>Connect</li>
  <li class="future">Donate</li>
</ul>

</body>
</html>
```

In the example above, the list item with a class of `future` will be hidden from view in the browser.

Understanding Display:

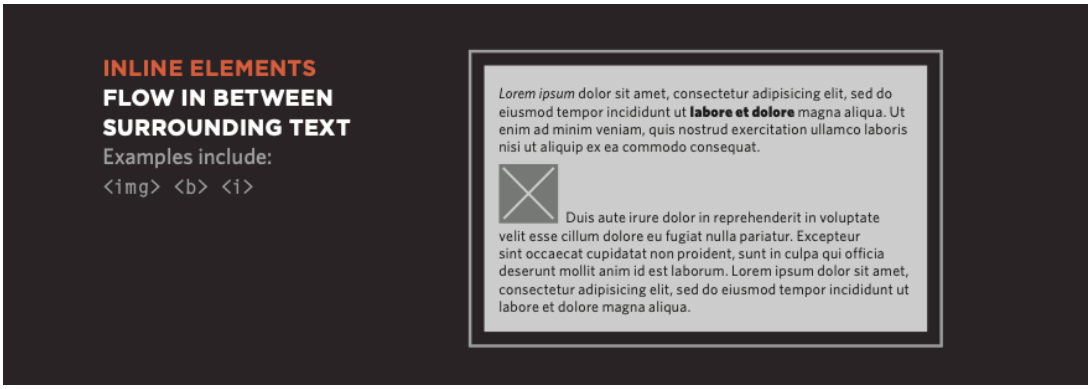
Every element on a web page is a rectangular box. The `display` property in CSS determines just how that rectangular box behaves. The default valu for all elements is `inline` .

As we have seen that some elements fill the entire browser from left to right regardless of the size of their content. Other elements only take up as muc horizontal space as their content requires and can be directly next to other elements. So, this behaviour is determined by the display property of eve element.

We have learned about the block elements above and now before diving into display property in css let's understand about inline elements:

Inline Elements:

Inline elements have a box that wraps tightly around their content, only taking up the amount of space necessary to display their content and n requiring a new line after each element. The height and width of these elements cannot be specified in the CSS document.



Now, after understanding the inline elements, let's start with the display property now,

we'll cover three values for the `display` property: `inline` , `block` , and `inline-block` .

Display: Inline

The default `display` for some elements, such as `<em>` , `<strong>` , and `<a>` , is called *inline*.

The height and width of these elements cannot be specified in the CSS document.

`inline` elements cannot be altered in size with the `height` or `width` CSS properties.

For example, the text of an anchor tag (`<a>`) will, by default, be displayed on the same line as the surrounding text, and it will only be as wide a necessary to contain its content.

```
To learn more about <em>inline</em> elements, read
<a href="#">MDN documentation</a>.
```

In the example above, the `<em>` element is `inline` , because it displays its content on the same line as the content surrounding it, including th anchor tag.

The CSS `display` property provides the ability to make any element an inline element. This includes elements that are not inline by default such a paragraphs, divs, and headings.

Consider the example shown below:

```
h1 {
  display: inline;
}
```

The CSS in the example above will change the display of all `<h1>` elements to `inline` . The browser will render `<h1>` elements on the same line a other inline elements immediately before or after them (if there are any).

Display: Block

As we have seen the block-level elements above, that these elements fill the entire width of the page by default, but their `width` property can also b set.

The display property of these elements is block.

Elements that are block-level by default include all levels of heading elements (`<h1>` through `<h6>`), `<p>` , `<div>` and `<footer>` .

We can make inline elements to behave as block level elements by changing their display property to block.

Consider the example shown below:

```
strong {
  display: block;
}
```

In the example above, all `<strong>` elements will be displayed on their own line, with no content directly on either side of them even though the contents may not fill the width of most computer screens.

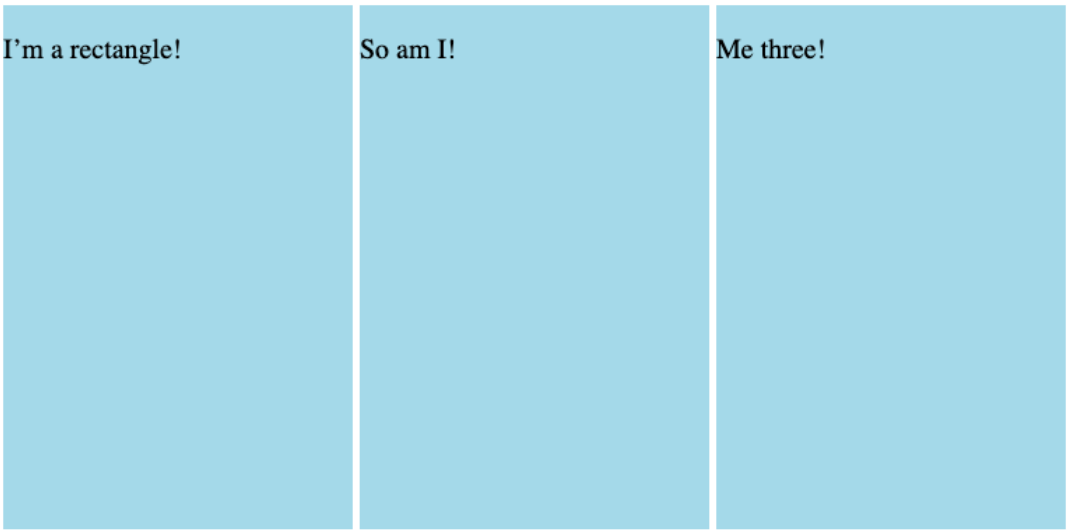
Display: Inline-Block

Inline-block display combines features of both inline and block elements. Inline-block elements can appear next to each other and we can specify the dimensions using the `width` and `height` properties.

Consider the example shown below:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
          .rectangle {
                      display: inline-block;
                      width: 200px;
                      height: 300px;
                     }
    </style>
  </head>
  <body>
    <div class="rectangle">
          <p>I’m a rectangle!</p>
        </div>
        <div class="rectangle">
          <p>So am I!</p>
        </div>
        <div class="rectangle">
          <p>Me three!</p>
        </div>
  </body>
</html>
```

Output:



There are three rectangular divs that each contain a paragraph of text. The `.rectangle` `<div>`s will all appear inline (provided there is enough space from left to right) with a width of 200 pixels and height of 300 pixels, even though the text inside of them may not require 200 pixels by 300 pixels of space.

Concept of Floating Elements:

float:

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 ca 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 `float` property is often set using one of the values below:

- `left` - moves, or floats, elements as far left as possible.
- `right` - moves elements as far right as possible.

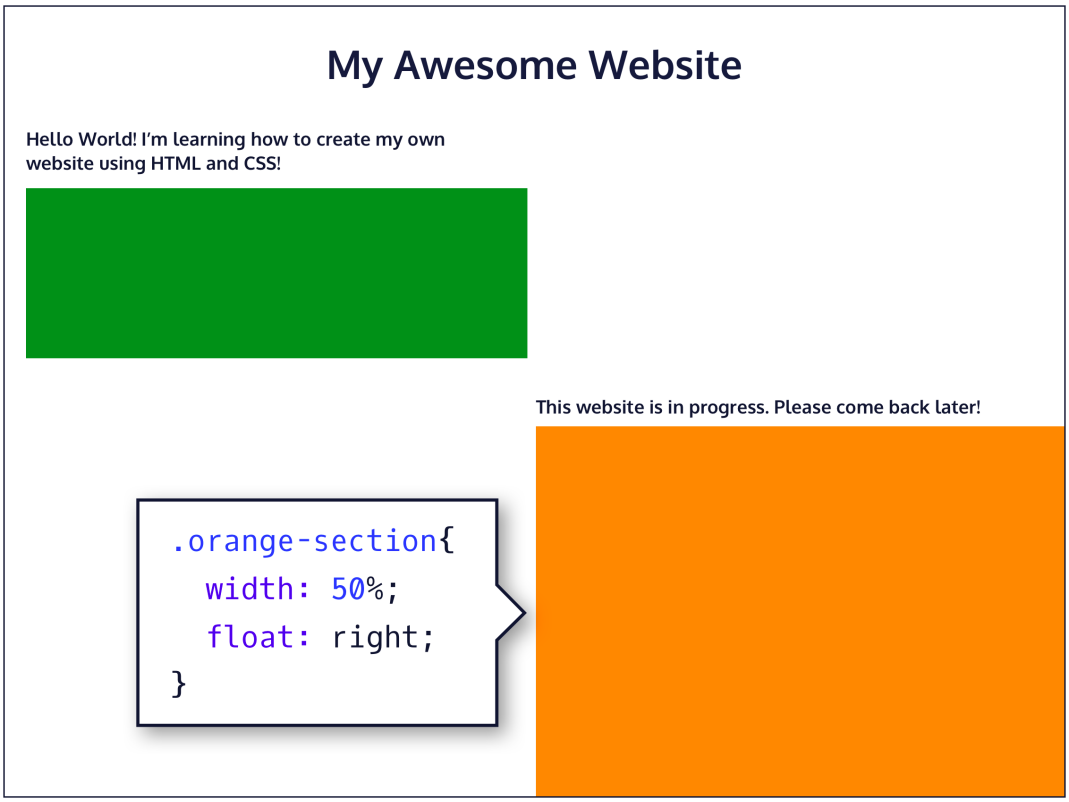
Consider the example shown below:

```
.green-section {
  width: 50%;
  height: 150px;
}

.orange-section {
  background-color: orange;
  width: 50%;
  float: right;
}
```

In the example above, we float the `.orange-section` element to the `right` . This works for static and relative positioned elements.

Output:



Concept of Clearing:

clear :

The `float` property can also be used to float multiple elements at once. However, when multiple floated elements have different heights, it can affe their layout on the page. Specifically, elements can “bump” into each other and not allow other elements to properly move to the left or right.

The `clear` property specifies how elements should behave when they bump into each other on the page. It can take on one of the following values:

- `left` —the left side of the element will not touch any other element within the same containing element.
- `right` —the right side of the element will not touch any other element within the same containing element.
- `both` —neither side of the element will touch any other element within the same containing element.
- `none` —the element can touch either side.

Consider the example:

```
div {
  width: 200px;
  float: left;
}

div.special {
  clear: left;
}
```

In the example above, all `<div>` s on the page are floated to the left side. The element with class `special` did not move all the way to the le because a taller `<div>` blocked its positioning. By setting its `clear` property to `left` , the `special<div>` will be moved all the way to the left sic of the page.

So, in this module we have learnt about some advance topics in css like typography, box model, display properties and positioning properties and we w continue further advance topics in css in the next module.

Interview Questions

> What is the Box model in CSS? Which CSS properties are a part of it?

A rectangle box is wrapped around every HTML element. The box model is used to determine the height and width of the rectangular box. The CSS Bo consists of Width and height (or in the absence of that, default values and the content inside), padding, borders, margin.

- **Content:** Actual Content of the box where the text or image is placed.
- **Padding:** Area surrounding the content (Space between the border and content).
- **Border:** Area surrounding the padding.
- **Margin:** Area surrounding the border.

> What is the difference between inline, inline-block, and block?

Block Element: The block elements always start on a new line. They will also take space for an entire row or width. List of block elements are

,

.

Inline Elements: Inline elements don't start on a new line, they appear on the same line as the content and tags beside them. Some examples of inlir elements are , , , and

tags.

Inline Block Elements: Inline-block elements are similar to inline elements, except they can have padding and margins and set height ar width values.

> **What property is used for changing the font face?**

We can use the font-family property for achieving this. The font-family property is used for specifying what font needs to be applied on th targetted DOM element. It can hold several font names as part of "fallback" mechanism in case the browser does not support the fonts.

> **How is border-box different from content-box?**

`content-box` is the default value box-sizing property. The height and the width properties consist only of the content by excluding th border and padding.

The `border-box` property includes the content, padding and border in the height and width properties.

Thank You !