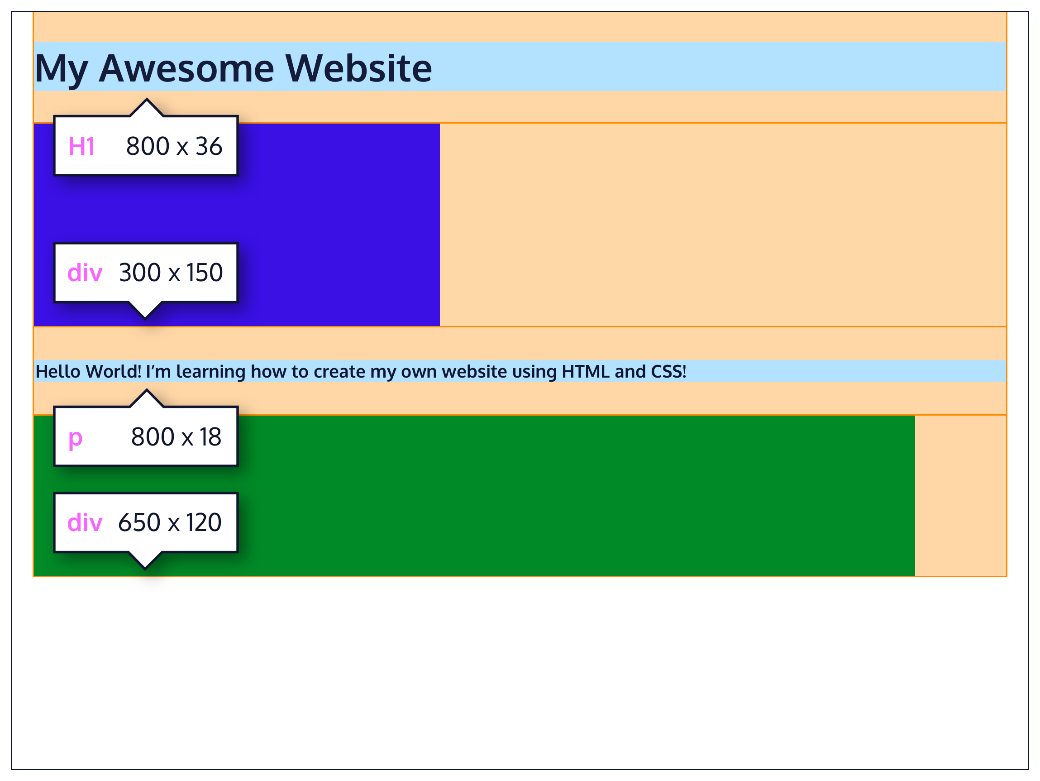
**Positioning**

CSS provides properties to position the elements, like where an element is located, can share lines,…

1. **Position**



Block-level elements like these boxes create a *block* the full width of their parent elements, and they prevent other elements from appearing in the same horizontal space.

Block-level elements always stack on each other, meaning they don’t overlap horizontally. The default position for the blocks are on the left side of the browser.

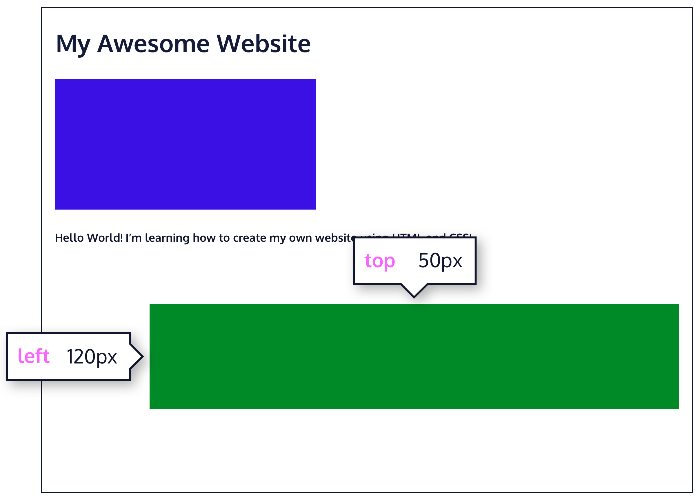
The default position of an element can be changed by setting its **position** property. The **position** property can take one of five values:

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

Now let’s get into each one.

1. **Relative**

This value allows you to position an element *relative* to its default static position on the web page. This is done by accompanying the position declaration with one or more of the following ***offset*** *properties* that will move the element away from its default static position: **top-bottom-left-right (away)**

We can specify values in pixels, ems, or percentages.

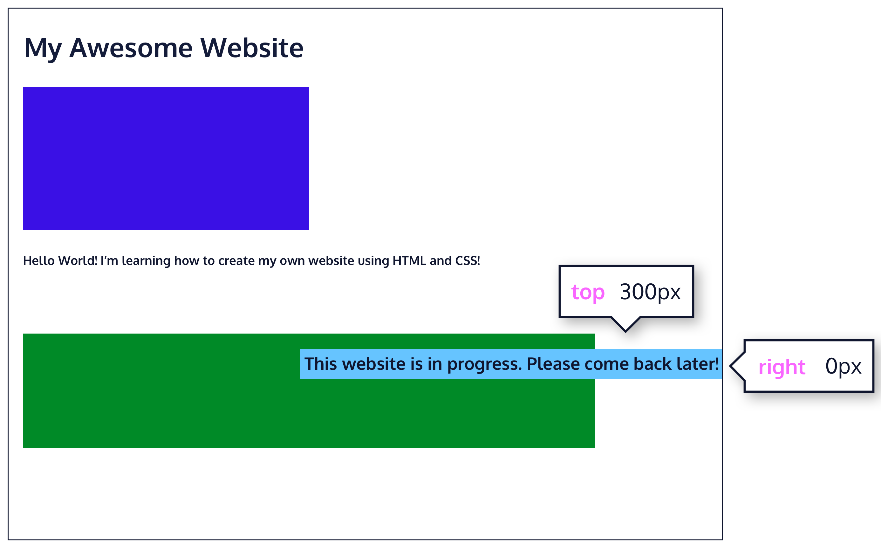
.green-box {  
  background-color: green;  
  position: relative; 🡪   
  top: 50px;  
  left: 120px;  
}

Offsetting the relative element will not affect the positioning of other elements.

1. **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. It is removed from the document flow.

The element will be positioned relative to its closest positioned parent element, while offset properties can be used to determine the final position from there.

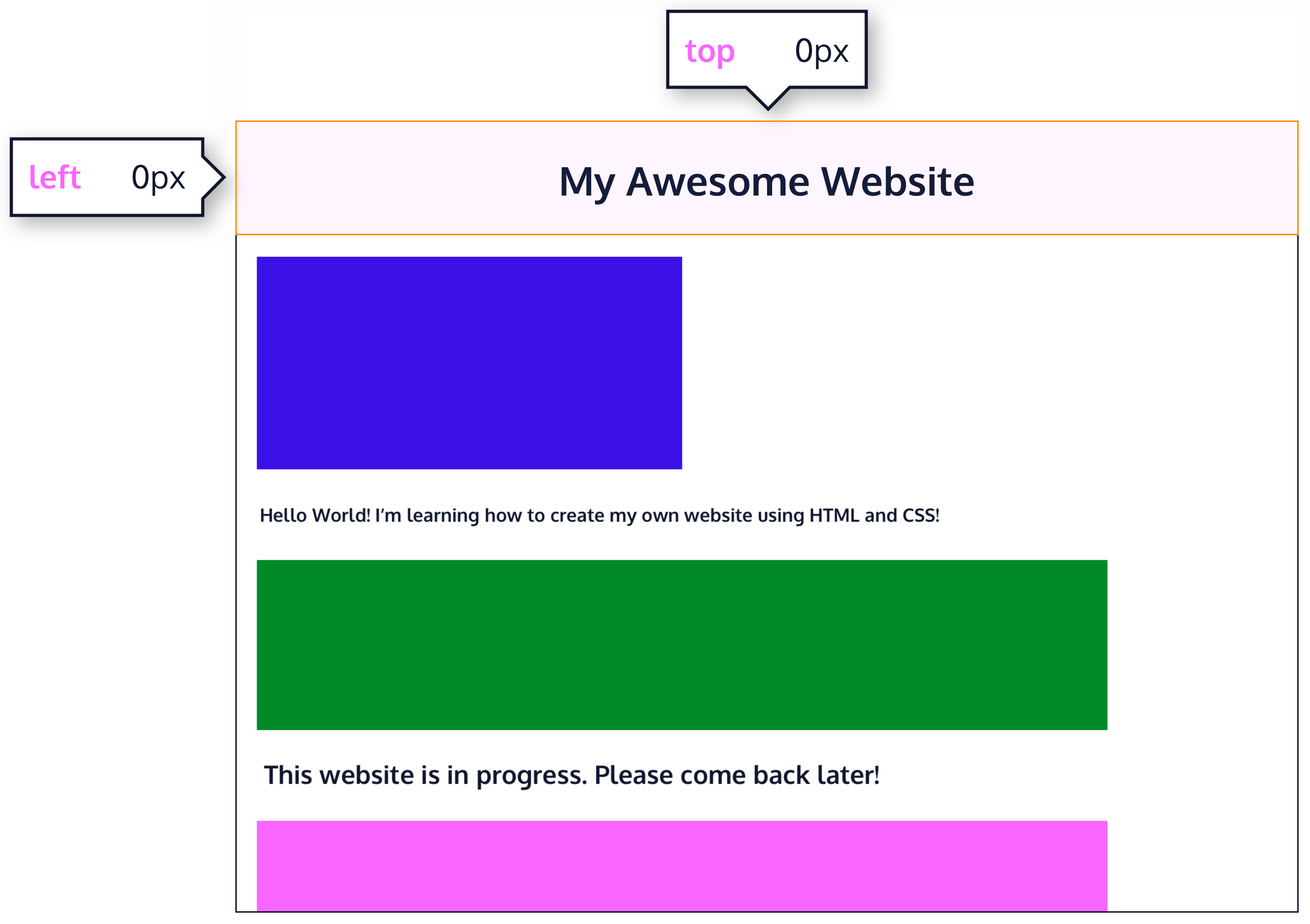


Here, the text is displaced from its static position at the top left corner of its parent container.

1. **Fixed**

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 the familiar offset properties top, bottom, left, and right.

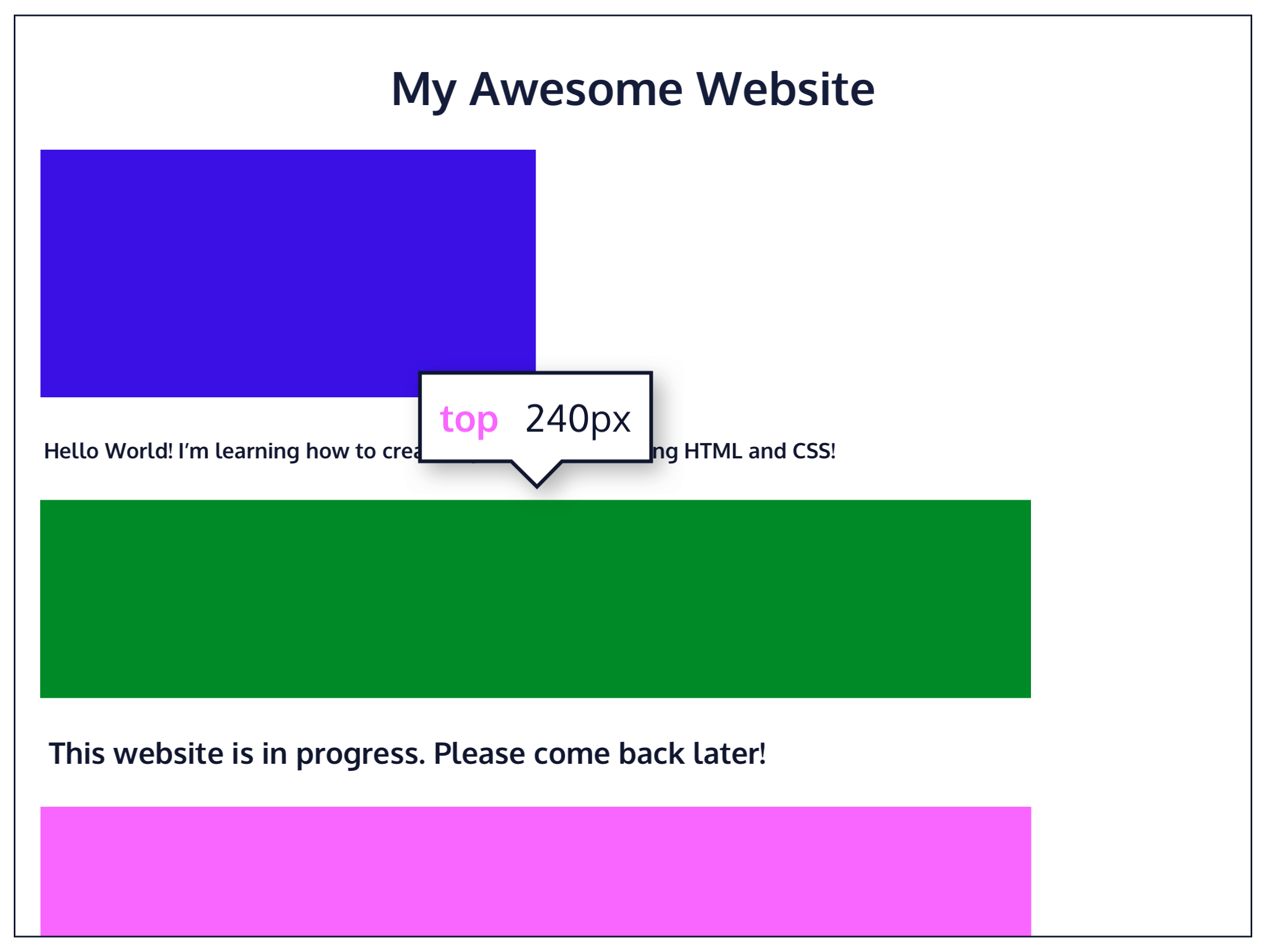
Note that when a position is set to fixed, it is also removed from the flow of the document.

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

This technique is often used for navigation bars on a web page.

1. **Sticky**

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 the 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;  
}

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 will stick to that position until it reaches the bottom of its parent container where it will “unstick” and rejoin the flow of the document.

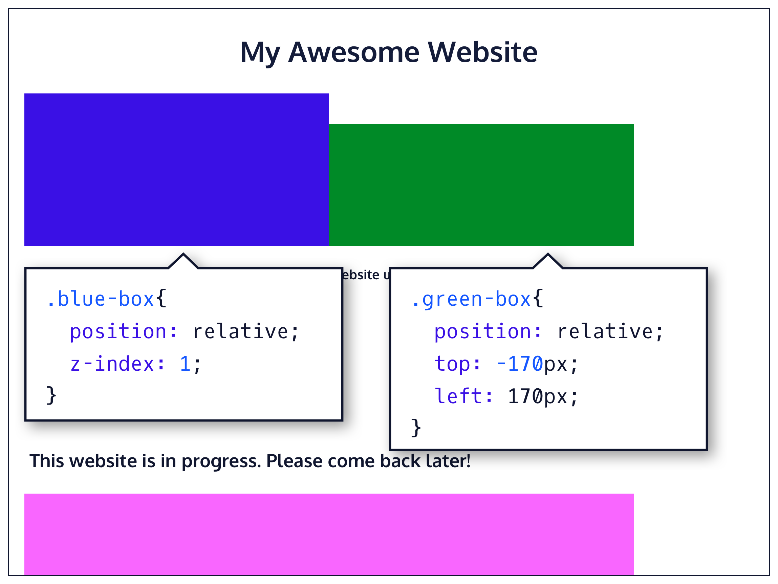
1. **Z-index**

When boxes on a web page have a combination of different positions, the boxes (and therefore, their content) can overlap with each other, making the content difficult to read or consume.

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 be thought of as the *depth* of elements, with deeper elements appearing behind shallower elements.

The **z-index** property accepts integer values. Depending on their values, the integers instruct the browser on the order in which elements should be layered on the web page. **The property does not work on static elements**.

Default z-index = 0, if not specified. We can rank the z-index of the elements, to see which one will be in front and which one will be in the back (🡪 fixed & sticky elemets should have high **z-index**)

.blue-box {  
  background-color: blue;  
  position: relative;  
  z-index: 1;  
}  
.green-box {  
  background-color: green;  
  position: relative;  
  top: -170px;  
  left: 170px;  
}

1. **Float**

If you’re simply interested in moving an element as far left or as far right as possible in the container, you can use the float property. The float property is often set using left or right values, and **must have a width specified**. Else, the element will have full width of its containing element 🡪 no difference.

Note, however, that moving elements left or right for layout purposes is better suited for tools like CSS grid and flexbox, which you’ll learn about later on.

.green-section {  
  width: 50%;  
  height: 150px;  
} 🡪   
  
.orange-section {  
  background-color: orange;  
  width: 50%;  
  float: right;  
}

1. **Clear**

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 any sides (default value)

🡪 If any values are specified, the element cannot be touched on that side, and they will move down so that they cannot be touched.