Coding HTML with NVDA -Part 13(The Box Model and CSS Units)

# Introduction

Hey guys what’s up it’s your girl Thee Quinn here and I’m back with another video in the html series. In today’s video I will be explaining the CSS “box model” and introducing you to CSS units. But before we get into it, please be sure to like the video if you love the content, subscribe if you are new and turn on my notification bell to be alerted whenever I post the next video. Also note that the instrumentals for this video have been provided by Inner Sanctum Entertainment Ltd. But without further ado, let’s get right into it.

# Start of tutorial

So, before we get deeper into CSS, there is still some foundation that we have to lay. If we don’t lay this foundation you can find yourself being lost for the rest of the series. In CSS, the term "box model" is used when talking about design and layout. When styling, we should think of html elements as having boxes around them. The box model in CSS is essentially a box that wraps around every HTML element. This box has 4 main features which are margins, borders, padding, and content. Now when you think of an html element being in a box, it has 4 sides. Just imagine it on the screen in front of you. It will have a top, right, bottom, and left side. These sides of the box are referred to as its border. Now for every html element, there is some content inside. For example, the text in a heading tag will be its content. The actual space between the content and the border, is what is called the padding, and this is transparent. And lastly, the space outside of the border of the element is called the margin, and is also transparent. So, think of it this way. Every element has some content. The padding is around the content. The border is around the padding. And the margin is around the border

The box model allows us to add a border around elements, and to define space between elements. Later on in this series I will teach you guys how to change the properties for each part of the box model. It is important to know how the box model works in order to set the width and height of an element correctly in all browsers. With CSS, when you set the width and height properties of an element, you just set the width and height of the content area. you must also add padding, borders and margins to calculate the full size of an element. This is how the total width of an element is calculated:

Total width = width + left padding + right padding + left border + right border + left margin + right margin

This is how the total height of an element is calculated:

Total height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

This means that when you set the width or height of an element, the element often appears bigger than you set it. This is because the element's border and padding are added to the element's specified width or height. Now, the box-sizing property solves this problem. The box-sizing property allows us to include the padding and border in an element's total width and height. If you use the box-sizing property, setting the value as border-box on an element, padding and border are included in the width and height.

So, I’m going to show you an example of using this property. Don’t pay attention to the other properties in the example for now, you will learn them later on:

div {

width: 300px;

height: 100px;

padding: 50px;

border: 1px solid red;

box-sizing: border-box;

}

According to my source, Since the result of using the box-sizing with the border-box value is so much better, many developers want all elements on their pages to work this way. Applying this to all elements is a safe and wise thing to do, and you can easily do it by using the universal selector. Here is how you would do it:

\* {

  box-sizing: border-box;

}

Now that is it for the box model. Let’s move on to CSS units.

# Units

As you will see throughout this series, many CSS properties take a length as their value. CSS has several different units for expressing length. Length is represented by a number followed by a length unit. When using length as a value, a space cannot appear between the number and the unit. However, if the value is 0, the unit can be omitted. Also, for some CSS properties, negative lengths are allowed. So, the basic syntax for using length units is:

selector {

property: 5units;

}

For the purposes of this example, I set the value of the unit to 5. Now, here are two types of length units. These are absolute and relative. Let’s start with absolute.

## Absolute length

The absolute length units are fixed. This means that a length expressed in any of these units will appear as exactly that size. Absolute length units are not recommended for use on screen, because screen sizes vary so much. However, they can be used if the output medium is known, such as for print layout. Here are the possible absolute units:

* Cm: which stands for centimeters

1cm = 37.8px = 25.2/64in

* mm: which stands for millimeters

1mm = 1/10th of 1cm

* in: which stands for inches

1in = 2.54cm = 96px

* px: which stands for pixels

1px = 1/96th of 1in

* pt: which stands for points

1pt = 1/72 of 1in

* pc: which stands for picas

1pc = 1/6th of 1in

Note: Pixels (px) are relative to the viewing device. For low dpi devices (DPI means dots per inch), 1px is one device pixel (dot) of the display. For printers and high-resolution screens 1px implies multiple device pixels. Now let’s move on to relative lengths.

## Relative Lengths

Relative length units specify a length relative to another length property. These units scale better between different rendering medium. Here are all the relative units:

* em

this is relative to the font-size of the element

2em means 2 times the size of the current font

* ex

This is relative to the height of the current font

This is rarely used.

* ch

This is relative to the width of the "0" (zero)

* rem

This is relative to the font-size of the root element

* vw

This is relative to 1% of the width of the viewport

* vh

This is relative to 1% of the height of the viewport

* vmin

This is relative to 1% of the viewport's smaller dimension

* vmax

This is relative to 1% of the viewport's larger dimension

* %

This is relative to the parent element

Note: The em and rem units are practical in creating a perfectly scalable layout. Also, the viewport is the browser window size. If the viewport is 50cm wide, 1vw = 0.5cm.

Now, CSS units are one of the most fundamental aspects to consider while creating a site layout. These units will define how your design will interact on various devices. Pixels are the most commonly used and accepted unit. It’s considered the base of measurement for many other units. It provides the most consistent result among various devices. A size set in pixels will remain the same on all screen sizes. All other absolute units (such as inch, centimeter, etc. have very little to no use cases for screen CSS, but since they are real-world measurement units, they can be effectively used in print CSS.

So, as we said before em depends on the font size of the parent element or the document. By default, 1em is equal to 16px if no font-size is defined. Em inherits size from its immediate parent’s font size. For example, if a parent element has a font size of 18px, then 1em will be measured as 18px for all its children. The advantage of using em is,

if you decide to change the font-size, padding, and margin of each element proportionately, then you just have to change the parent element font size and all other elements will adjust accordingly. That is not the case with absolute units like px, where you have to adjust each element individually

Rem can be really helpful when you want to change an element’s font size without affecting it’s children. This is because they always refer to the root element font size, not the parent element.