HyperionDev

CSS Overview

Task

Visit our website

Introduction

In the previous task, you added HTML elements to a web page. Once this is done, your web page will contain the information you want, but it probably doesn't look that great! Don't worry, in this task, you will learn to use CSS to style the elements that you have added to your web page to make it much more attractive and exciting.

CSS

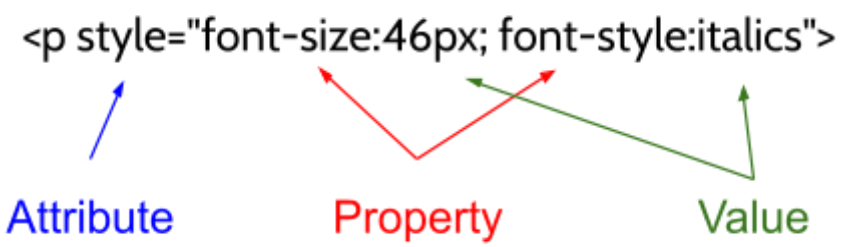
Cascading Style Sheets (CSS) is a language that is used to change the presentation and look of a particular document that's been written in a markup language such as HTML. CSS is usually applied to web pages, but can also be used in other areas, such as XML documents.

Inline style

HTML elements are described using attributes and properties. You can style a web page by changing the properties of the elements that make up that web page. For example, any text that you add to a web page has several properties that you can change. These include: `font-family` (Arial, Times New Roman, etc.), `font-style` (normal, italics, etc.), and `font-size`. An example of using the `style` attribute to change the font of an element is shown below:

```
<p style="font-size: 46px; color: blue">

  This is the paragraph where I describe myself.

</p>
```

Like all other attributes, the `style` attribute goes inside the element's beginning tag, right after the tag name. After specifying that you are changing the `style` attribute, you type =, and then, within double quotes, list the properties you want to change with their values listed after a colon.



Applied CSS inline styling

When you style an element individually by changing that element's properties, it is known as **inline styling**. Inline styling allows you to specify the style of an individual element in the line where that element is declared. What if you wanted to apply similar styles to all elements of a certain type? For instance, what if you wanted to change the font of all paragraphs on your web page? You can do this by creating a CSS rule.

Internal CSS

The example below shows how you can define a CSS rule in the `head` part of your HTML template. This is called **internal CSS**. This example shows a CSS rule that will cause all paragraphs to be in the colour red and be of the `font-family` Arial. If the browser can't find Arial, then it will look for Helvetica. Paragraphs will also have a background colour of blue.

```
<style>
  p {
    color: red;
    font-family: Arial, Helvetica;
    background-color: blue;
  }
</style>
```

CSS syntax consists of a selector and a declaration.

CSS syntax

- The **selector** indicates which HTML element you want to style.
- The **declaration** block contains one or more declarations separated by semicolons. A declaration always ends with a semicolon and is surrounded by curly braces.
- Each declaration includes a **property** and a **value**, separated by a colon.

Extra resource

Explore other **CSS properties** that can be modified.

The CSS below should look familiar. Here, the selector is always an element.

```
/* Selectors for paragraphs and body */
p {
  color: red;
  font-family: Arial, Helvetica;
  background-color: blue;
}

body {
  text-align: center;
}
```

However, you can also use class and ID selectors. A class selector is used when the selector describes the rule for all elements that have a `class` attribute with the same name defined. An ID selector describes the style of an element with a specific `id` attribute defined.

To learn more about class selectors, please consult the following **class selector resource**. To learn more about ID selectors, you might find the following **ID selector resource** useful.

Open **example.html**. Notice that in this file we have several elements in which the `class` or `id` attribute has been defined. For example, notice how there are several `<a>` elements that have the `class` attribute set to `more-button`, as shown below:

```
<a href="" class="more-button">more</a>
```

However, there are also `<a>` elements that do not have a specified `class` attribute, such as the following:

```
<li><a href="contact.html">contact</a></li>
```

Therefore, instead of creating a CSS with an element selector ("`a`"), we could create a rule that is specific to a class selector. See an example of this below.

When you use a class selector, the name of the class will always be preceded by a dot (`.`). The style rule below will cause all elements where the `class` attribute has been set to `more-button` to have bold text.

```
.more-button {
  font-weight: bold;
}
```

Similarly, you could create a style sheet that uses ID selectors. ID selectors start with a hash (`#`), as shown in the example below:

```
#first-more-button {
  font-weight: bold;
  font-family: cursive;
}
```

The rule above would cause the text of the element, where the `id` attribute equals `first-more-button`, to be bold and cursive.

Although you can have many elements that have a `class` attribute with the same value, an ID name must be **unique** in the document.

You can use a combination of internal CSS (declared in the `head` of your HTML document) and inline styles. But how do style rules apply? Essentially, the closer the style is to the element, the higher its precedence. For instance, if you have an internal CSS rule for paragraphs on your page but want one paragraph to be styled differently, you can use inline styling for that paragraph, which will override the internal CSS rule.

External CSS

If your website consists of many HTML files, you are likely to want to be able to apply the same style rules to all the web pages. To accomplish this, use external CSS instead of internal CSS. To do this, create a separate file with the extension **.css**. Within this file, write all the style rules that you would like to specify. You can then link this external CSS file to all the HTML files in which you would like the style rules applied. To link an external CSS file to a specific HTML file, do the following:

```
<link href="examplesCSS.css" rel="stylesheet" type="text/css" />
```

In the `head` part of your HTML, create a reference to your CSS file so that the styles can be used in your web page. `href` refers to the name and path of your CSS file. In the example above, the file **exampleCSS.css** is in the same folder as the HTML page. `rel` says what sort of relation the file is to the HTML – i.e., the style sheet. `type` tells the browser what sort of file it is and how to interpret it.

Internal, external, or inline?

If we were to include the CSS shown below in our CSS file, the result would be the same as if it were in the `<style>` tags in the HTML page.

```
p {
  color: red;
  font-family: Arial, Helvetica;
  background-color: blue;
}

body {
  text-align: center;
}
```

So, is it better to use internal or external CSS? **Generally, it's better to use external CSS wherever possible.** Why? Because readability is an important factor. Imagine trying to read through a whole bunch of different languages at once (CSS, HTML, JavaScript), all in one file. Rather separate them, as it's much easier to follow what's happening, especially when you are building fancy websites where plenty of different styles are being used.

Another important reason to separate CSS from HTML files is to improve the maintainability of your website. If you wanted to update the look and feel of a website, this could easily be done by simply replacing the external CSS file if only external CSS is used for the website. Using external CSS also makes it easier to debug errors since all the CSS is in one place.

You may find, though, that it's necessary to use a combination of external, internal, and inline style. In this case, it is important to understand the concept of cascading, which we will get to later on.

Code hack

Here are two more cool CSS tricks:

1. If you would like to apply a certain style rule to an element when it's in a certain state (e.g., if you hover over it), you can do this as shown below:

```
/* Any button over which the user's pointer is hovering */
button:hover {
  color: blue;
}
```

   In this example, `hover` is a pseudo-class (a keyword that describes the state of the selector). To see a list of pseudo-classes, see **here**.

2. If you would like to apply a certain rule to an element that is a descendant (child) of another element, you can do so as shown below:

```
li li {
  list-style-type: circle;
}
```

   The rule above will make all list items that are descendants of other list items have a circle as a bullet point. To see more about using a descendant combinator, see **here**.

Let's now discuss pixels, a fundamental unit in web design used to set precise sizes for elements on a webpage.

Pixels

Understanding pixels in web development

What is a pixel?

A pixel (px) is a unit of measurement used to define the size of elements on a webpage. It's the smallest unit in digital imaging and provides precise control over element dimensions.

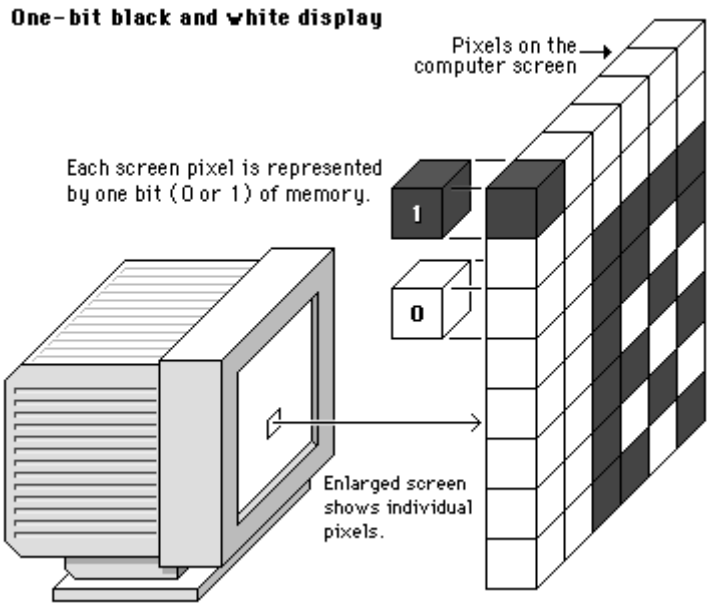


Illustration (Web Style Guide; 1997)

Photograph (Flyme, 2016)

Let's explain what is happening in both deer images:

1. **Full image (left side)**: This is like a high-resolution laptop screen. You can see the deer clearly, including details of its fur, eyes, and ears.

2. **Pixelated image (right side)**: This represents a very low-resolution portion, like zooming in on a tiny part of a low-quality laptop screen. It's so pixelated you can barely make out what it is.

3. **Resolution comparison**:

- The clear deer image might be something like 1000x750 pixels (just an example).
- The pixelated part might be like zooming in on a 20x20 pixel area of the ear.

4. **Pixels**:

- In the clear image, each pixel is so small you can't see individual ones.
- In the pixelated part, each large square is a single pixel, hugely enlarged.

5. **Detail level**:

- High resolution (left): Like a good laptop screen, showing fine details.
- Low resolution (right): Like an old, low-quality screen where you can see individual pixels.

This comparison demonstrates why higher resolution matters for clarity on laptop screens, especially when viewing detailed images or text.

Using pixels in CSS

Pixels are used in CSS to set fixed sizes for elements.

1. **Width and height**: This sets a box element's width to 200px and height to 150px.

```
.box {
  width: 200px;
  height: 150px;
}
```

2. **Margins and padding**: This applies a 20px margin outside and 10px padding inside the container.

```
.container {
  margin: 20px;
  padding: 10px;
}
```

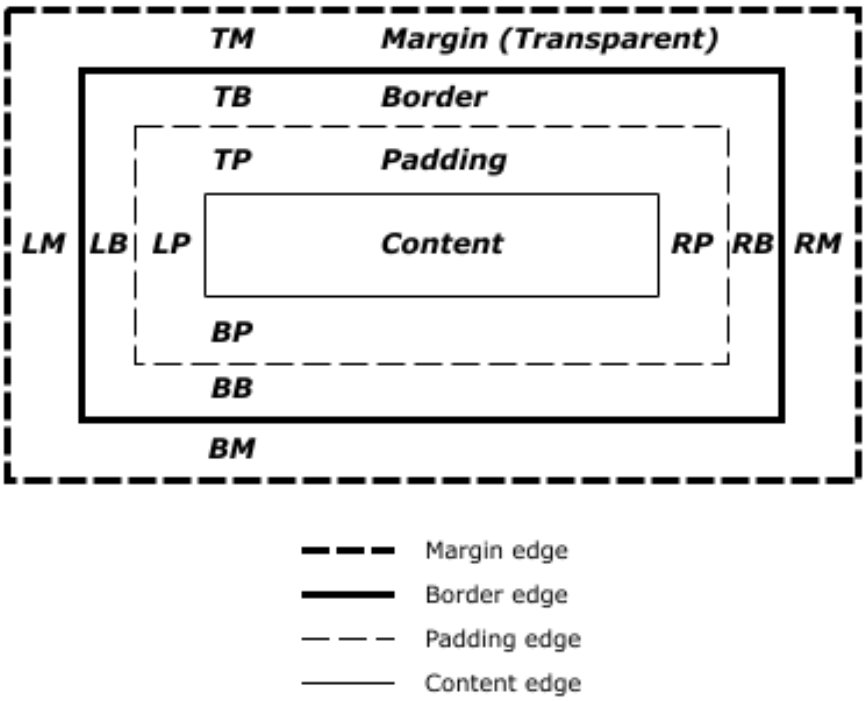
3. **Paragraphs**: This sets paragraph text size to 16px.

```
p {
  font-size: 16px;
}
```

Pixels provide exact sizing, but can affect responsiveness and accessibility. For more flexibility, consider using relative units like percentages (%) or em/rem units.

The box model

An important fundamental concept to understand in CSS is what is called the **box model**. The browser creates a rectangle for each element in the HTML document. The box model describes how the padding, border, and margin are added to the content to create this rectangle. The following diagram is a depiction of the box model for an element.



CSS Box Model (Yesiller, 2012)

Each box has a content area (e.g., text, an image, etc.) and optional surrounding padding, border, and margin areas, with the size of each area specified by properties.

The margin, border, and padding can be broken down into top, right, bottom, and left segments (e.g., in the diagram, “LM” for left margin, “RP” for right padding, “TB” for the top border, etc.).

The perimeter of each of the four areas (content, padding, border, and margin) is called an **edge**, so each box has four edges:

1. **Content edge or inner edge:** The content edge surrounds the rectangle given by the width and height of the box, which often depends on the element's rendered content. The four content edges define the box's content box.
2. **Padding edge:** The padding edge surrounds the box padding. If the padding has zero width, the padding edge is the same as the content edge. The four padding edges define the box's padding box.

3. **Border edge:** The border edge surrounds the box's border. If the border has zero width, the border edge is the same as the padding edge. The four border edges define the box's border box.

4. **Margin edge or outer edge:** The margin edge surrounds the box margin. If the margin has zero width, the margin edge is the same as the border edge. The four margin edges define the box's margin box.

Each edge may be broken down into a top, right, bottom, and left edge.

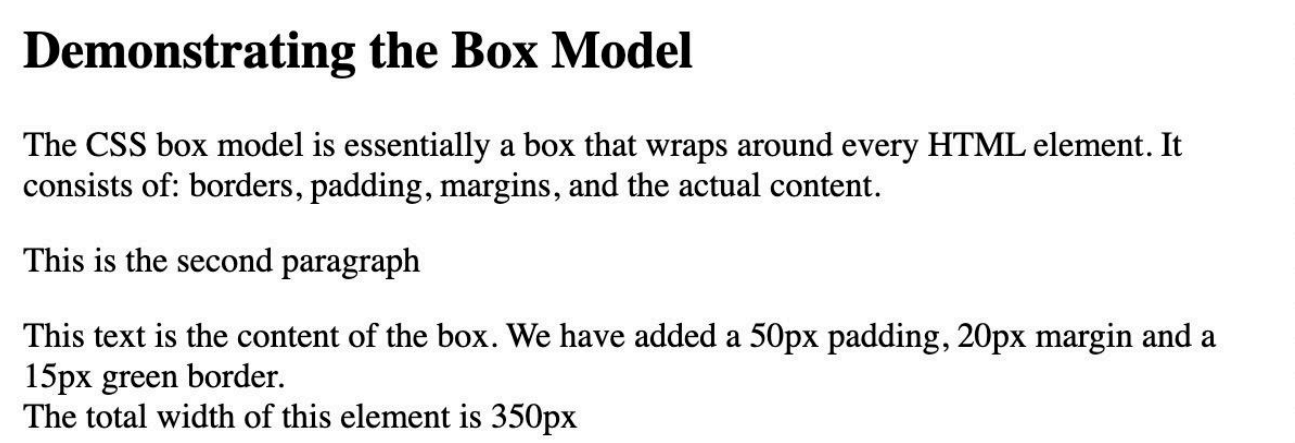
Let's put this into practice. Create a folder named **boxModel**, and in this folder create an HTML file named **index.html**. Copy the following code into the **index.html** file:

```
<!DOCTYPE html>
  <head>
    <title>Static Template</title>
    <link rel="stylesheet" href="style.css" type="text/css"/>
  </head>
  <body>
    <h2>Demonstrating the Box Model</h2>

    <p id="first-paragraph">
      The CSS box model is essentially a box that wraps around every HTML
      element. It consists of: borders, padding, margins, and the actual
      content.
    </p>

    <p class="second-paragraph">This is the second paragraph </p>

    <div class="div1">
      This text is the content of the box. We have added a 50px padding, 20px
      margin and a 15px green border.
    </div>
    <div class="div2">The total width of this element is 350px</div>
  </body>
</html>
```

Now, in the same folder, create a file named **style.css**. We need to double-check that the link to the external stylesheet is in the head section of the **index.html** file, and add it if it is not. Include the following line of code in the head section:

```
<link rel="stylesheet" href="style.css" />
```

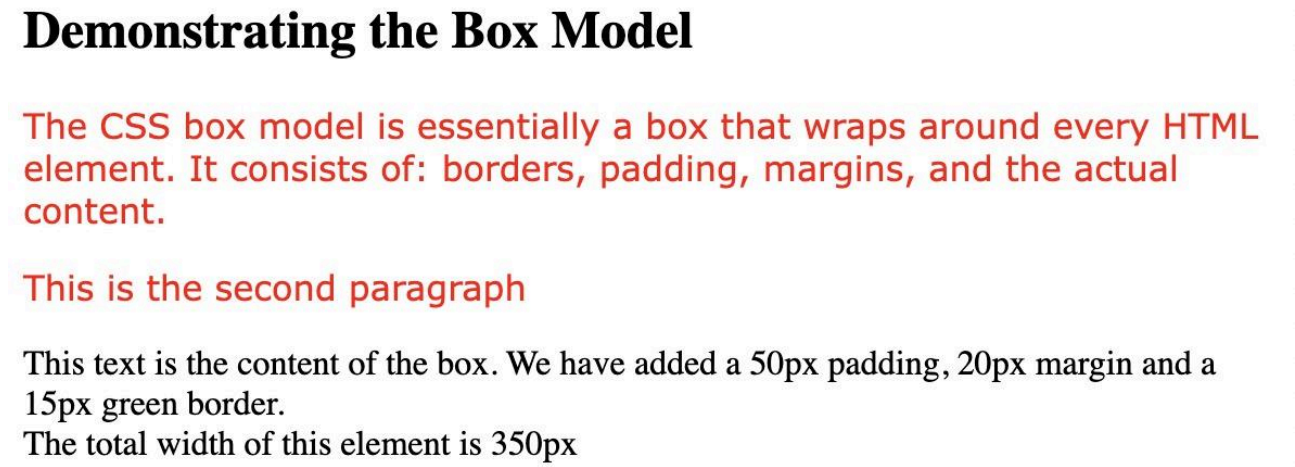
With no CSS code in the CSS file, when **index.html** is opened in the browser, it should look like this:



Let's change the CSS properties of all of the paragraph tags. In the HTML file, we will use an element selector `p`, change the font to Verdana, and the font colour to red. You can copy and paste this code into the CSS file:

```
p {
  font-family: Verdana;
  color: red;
}
```

After saving and refreshing the browser, both paragraphs should change accordingly:

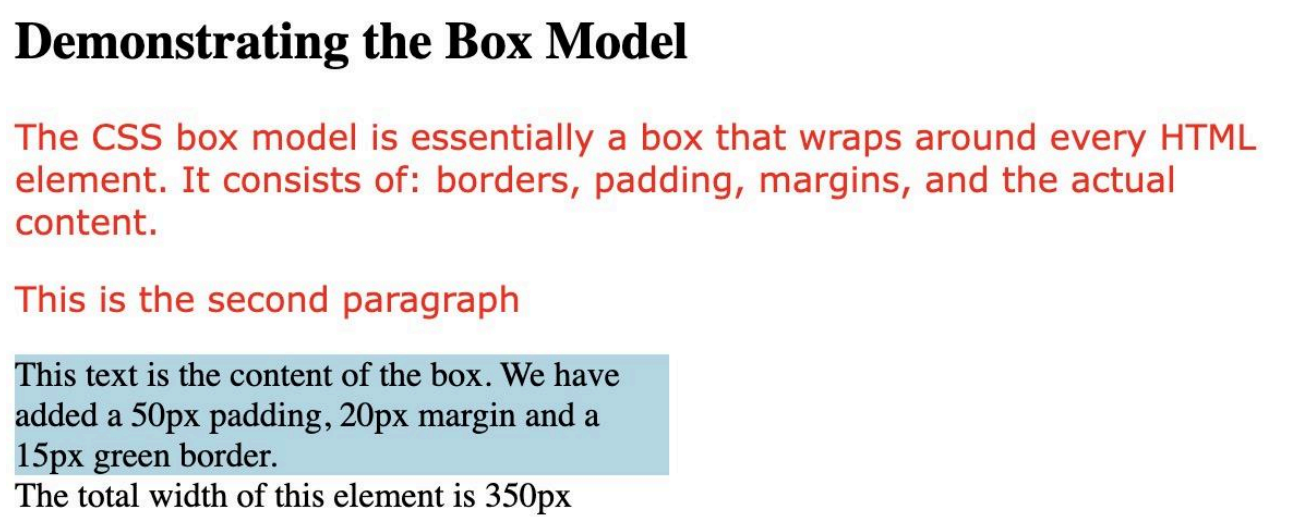


Now let's style the first `div` element. The `div` HTML element is the generic container for flow content. It has no effect on the content or layout until styled in some way using CSS, and it's also known as a content division element.

Let's first set the background colour and width of the element. We will use the class selector for the first `div` element with the class name `div1`, and change the background colour to `lightblue` and the width of the element to `300px`. Copy and paste the following code into the CSS file:

```
.div1 {
  background-color: lightblue;
  width: 300px;
}
```

Save and then look at the changes after reloading the browser. The result should be as follows:

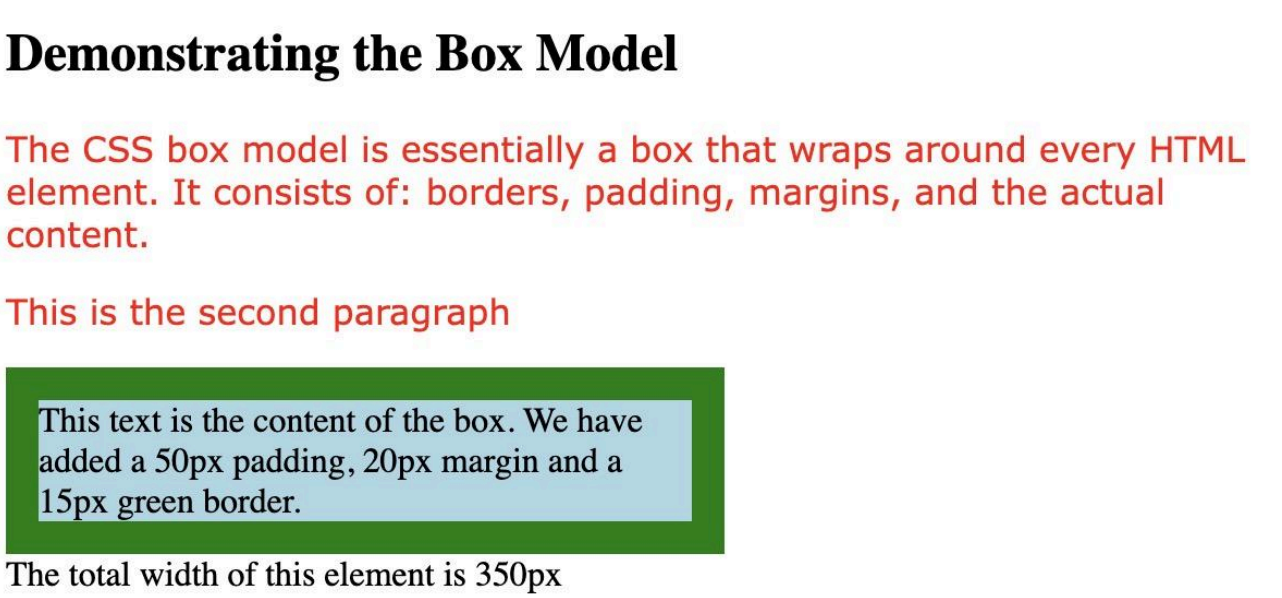


You will now notice that the background colour has changed and the container in which the text is situated now has a width of 350px.

Let's add a border to the same `div` container with the width set to `15px`, a border type `solid`, and the colour set as `lightcoral`, like so:

```
.div1 {

  background-color: lightblue;

  width: 300px;

  border: 15px solid green;

}
```

We should see the following change as this is applied:

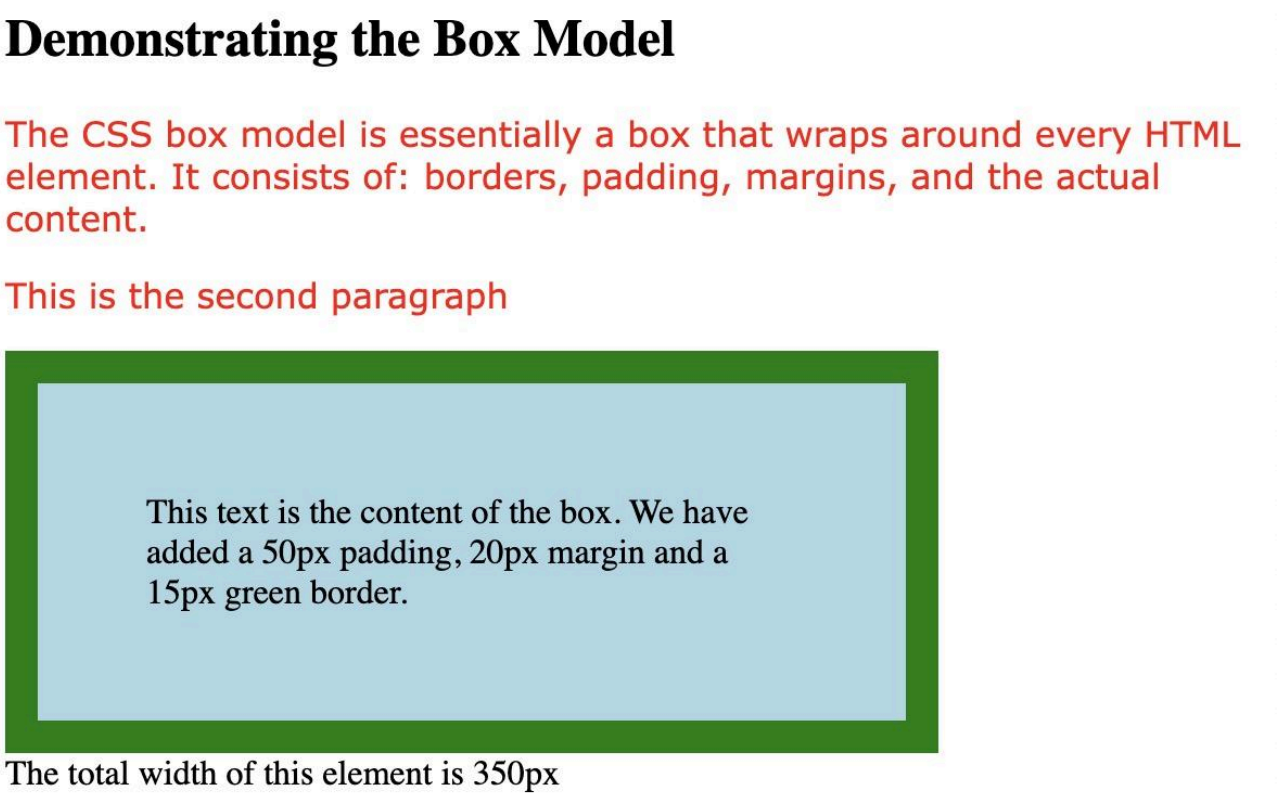


You will notice that there is no spacing between the border and the text (we need to add padding), as well as the border and the `div` element below it (with the content the total width of this element is 330px).

Let's add padding of 50px around the text:

```
.div1 {
  background-color: lightblue;
  width: 300px;
  border: 15px solid green;
  padding: 50px;
}
```

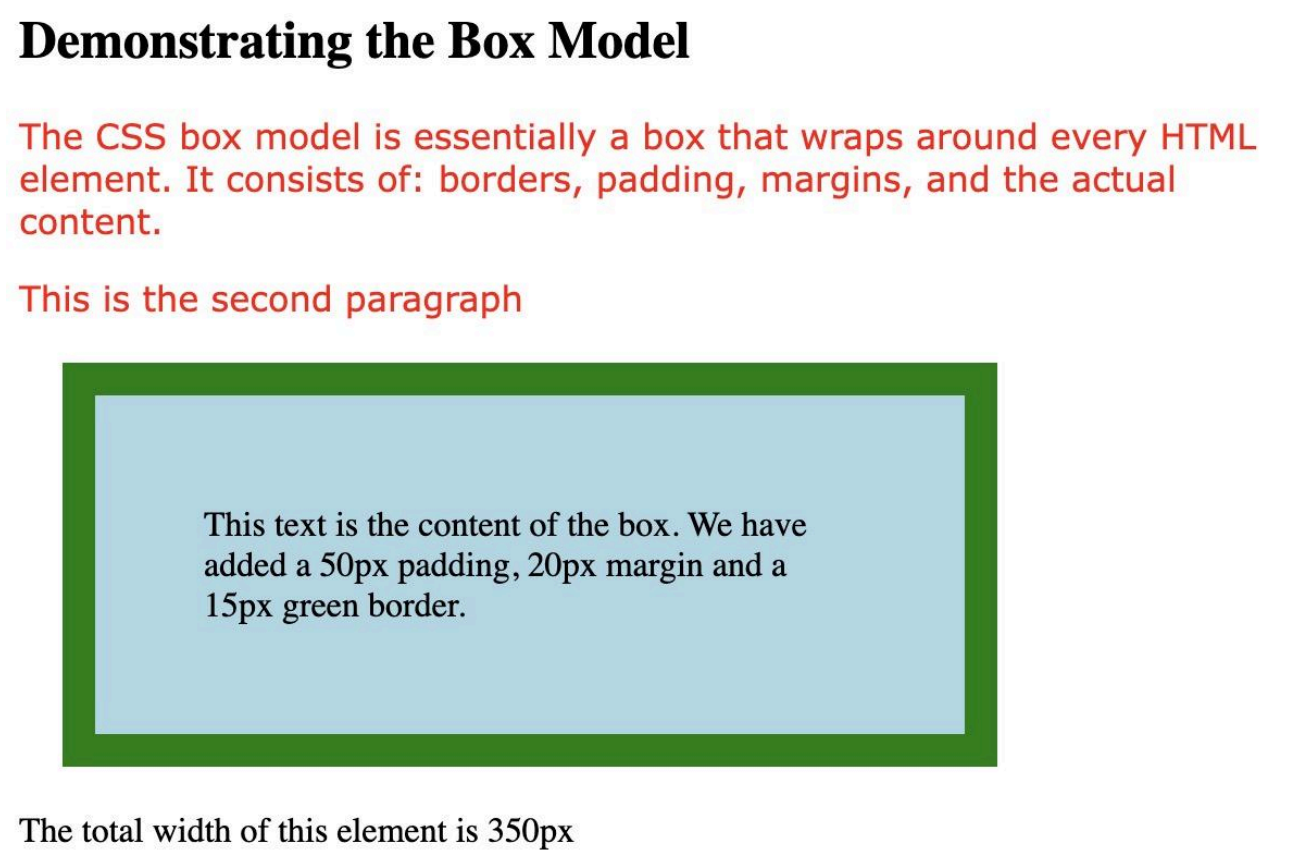
The result:



Now let's add a margin around the border of 20px:

```
.div1 {
  background-color: lightblue;
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
```

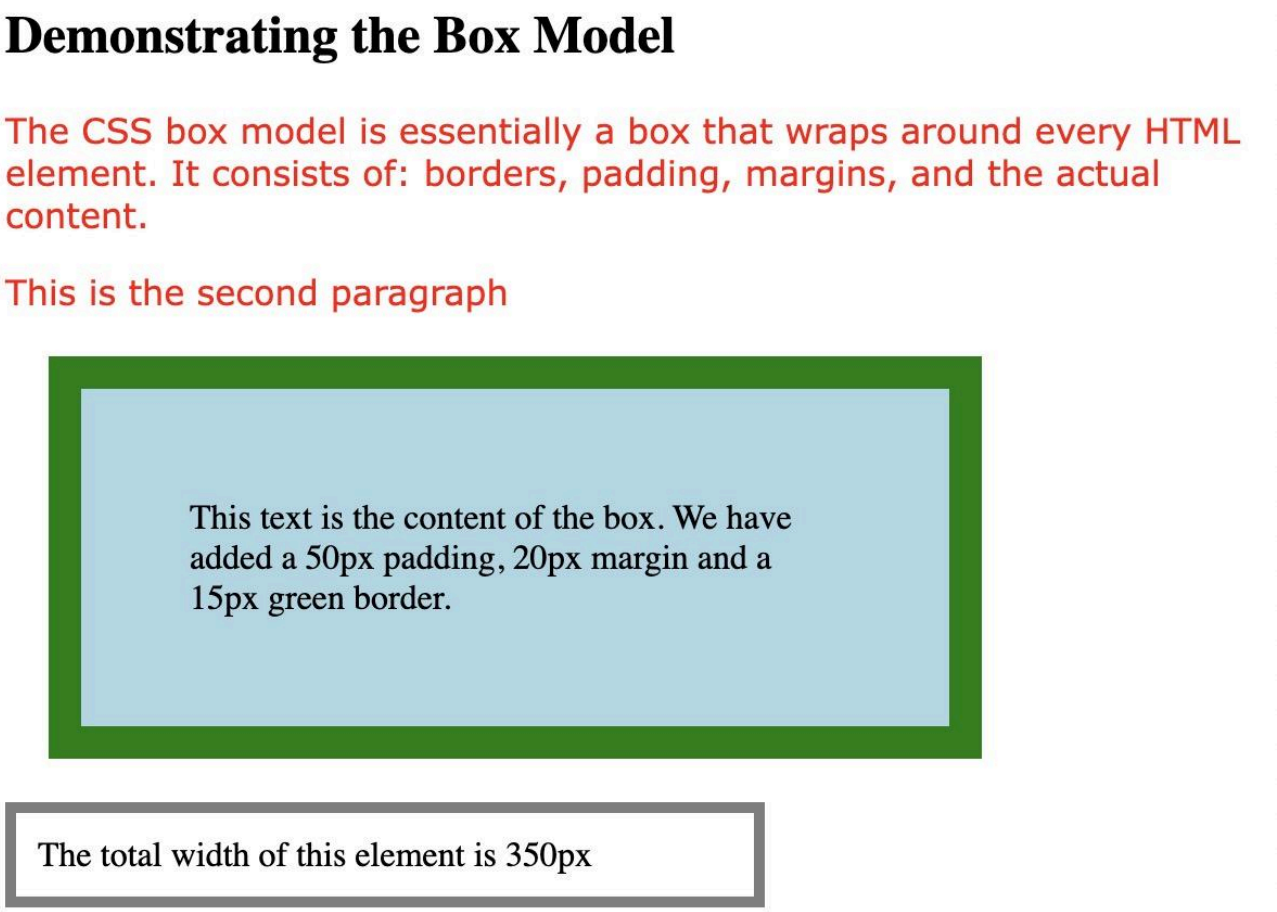
The result has spacing around the element:



Now let's add styling to our second `div` element with the class name `div2`. Add the following to your **style.css** file:

```
.div2 {
  width: 320px;
  padding: 10px;
  border: 5px solid gray;
  margin: 0;
}
```

The result is as follows:



It’s also possible to have different pixel sizes for the left, right, top, and bottom padding and margins by using the following CSS properties:

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

You are encouraged to explore these using the files that you created so that you can see how the changes occur in the browser.

Cascade

As we know, CSS stands for Cascading Style Sheets. You may have wondered what the “cascading” refers to. Cascading has to do with how the rules are applied.

If your website contains external, internal, and inline CSS, the inline CSS overrides internal CSS rules and external CSS files. Furthermore, internal CSS overrides external CSS rules. If there are conflicting rules regarding properties, some properties override other properties, but entire rules don’t override other rules. When several CSS rules match the same element, they are all applied to that element. Only after that are any conflicting properties evaluated to see which individual styles will win over others.

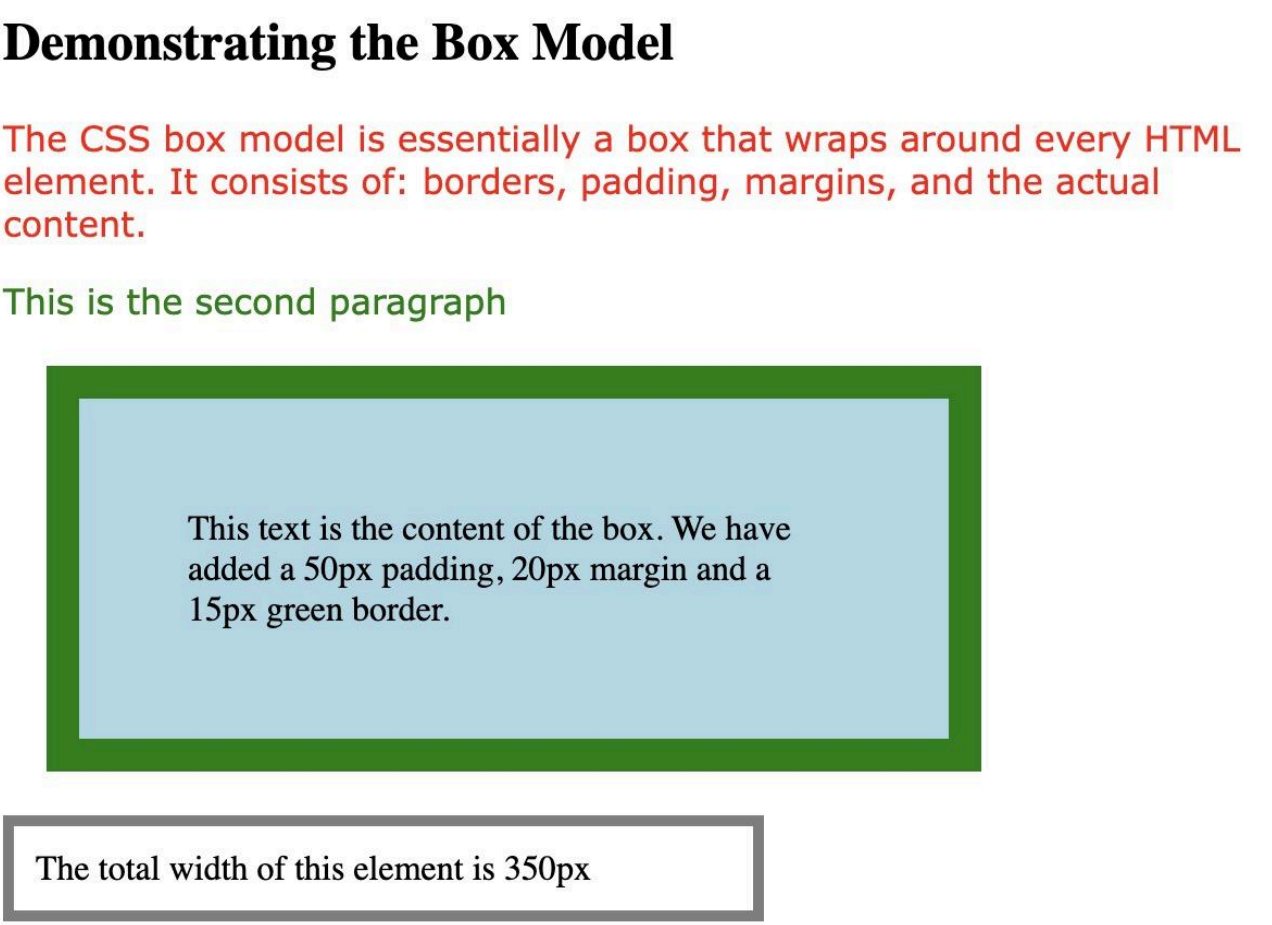
Another important rule to remember is that the more specific a rule is, the higher its precedence. For instance, in a style sheet that uses element selectors, class selectors, and ID selectors, element selectors are the least specific (because they could match the most elements on a page), whereas ID selectors are the most specific. Therefore, ID selectors will be applied over class selectors and element selectors.

Let's see this in action using the files from the box model demonstration that you have created.

We have previously used the element selector `p` to style all paragraph elements red. We can override this by being a bit more specific and changing the colour of all paragraphs of a specific class. Let's do this for the `secondParagraph` class and change it to green:

```
.second-paragraph {

  color: green;

}
```

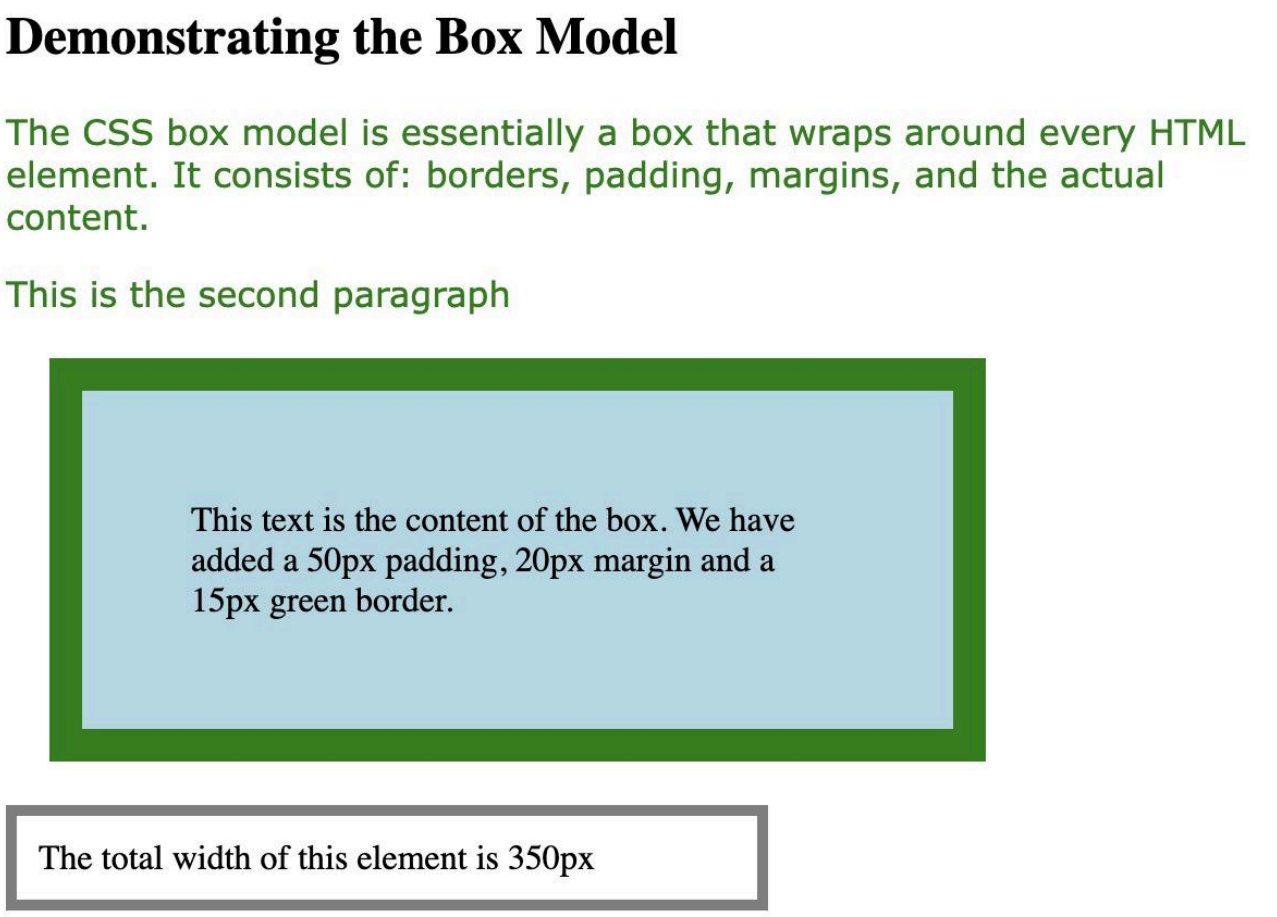
We can now see that the second paragraph colour changed despite the red colour assigned to all paragraphs:



Now let's give the first paragraph the same class name as the second paragraph:

```
<p id="first-paragraph" class="second-paragraph">
  The CSS box model is essentially a box that wraps around every HTML
  element. It consists of: borders, padding, margins, and the actual
  content.
</p>

<p class="second-paragraph">This is the second paragraph </p>
```

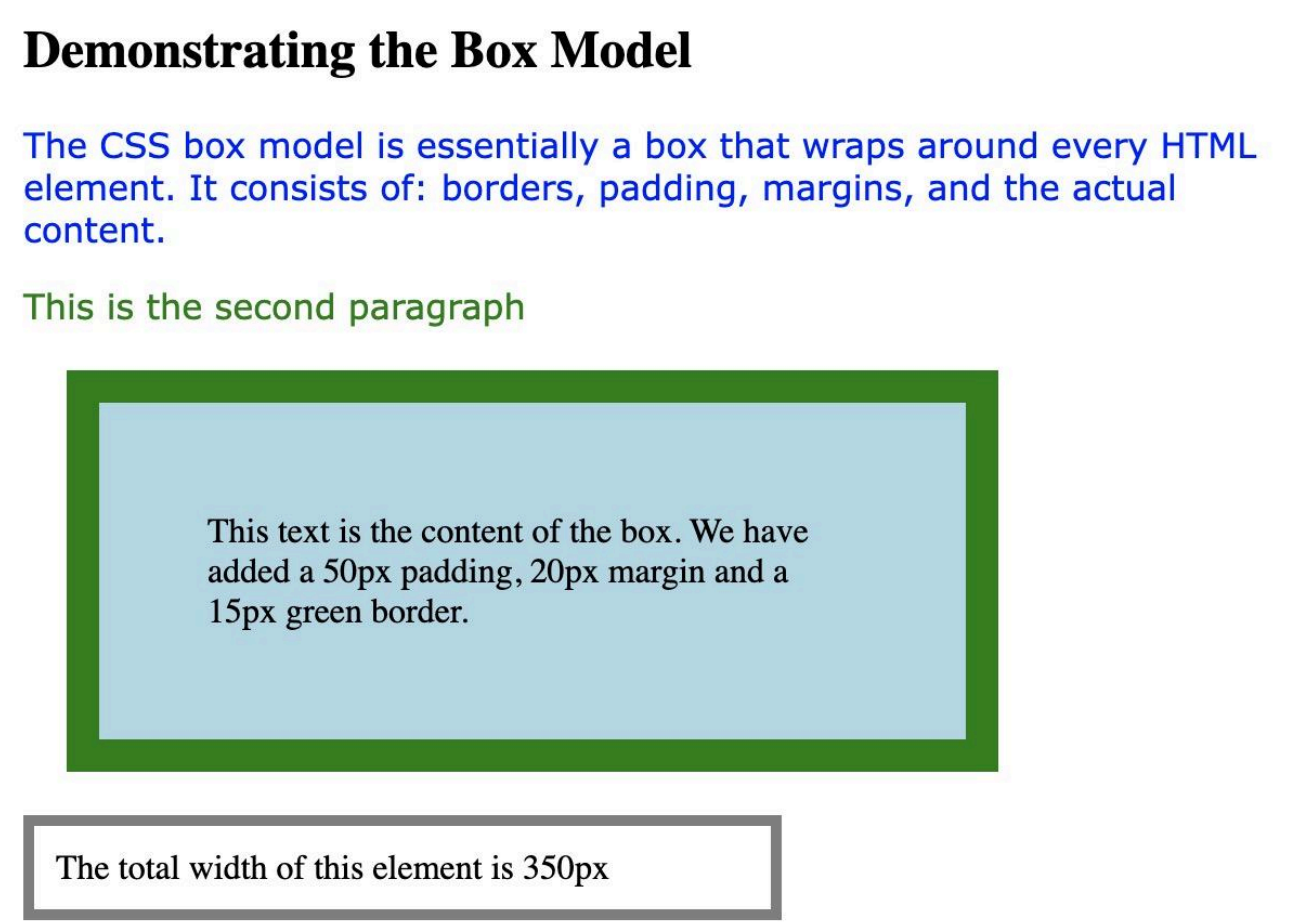
Both paragraphs will now be green:



Lastly, we can see that the first paragraph has the `id` named `first-paragraph`. If we use an `id` selector and change the colour of this `id` to blue, we will see the class colour get overridden. Add the following to your CSS:

```
#first-paragraph {

  color: blue;

}
```

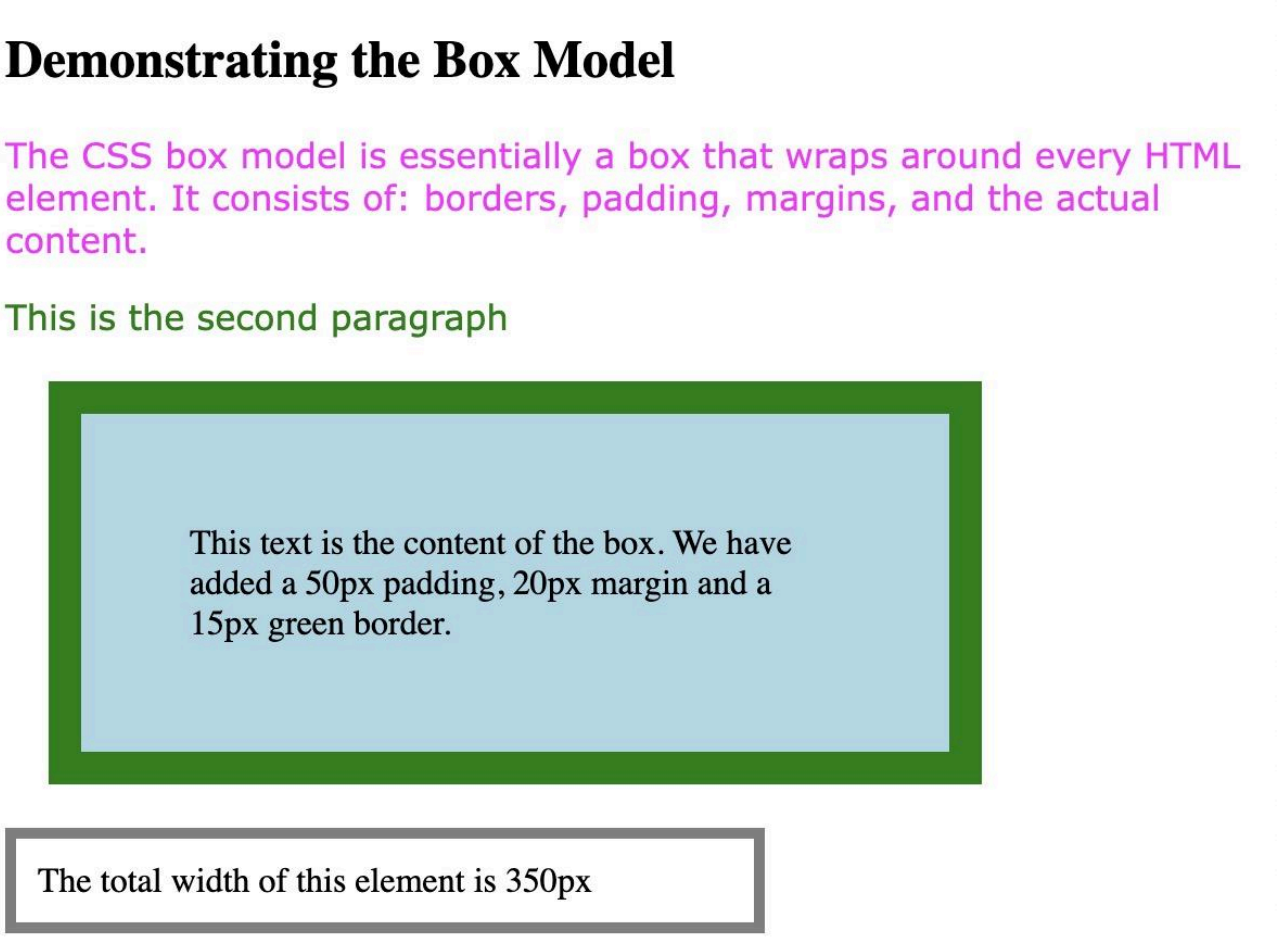
The result:



Lastly, the most specific type of style is inline styling, which will override any other styles applied to an element. Let's add inline styling to the first paragraph by changing the colour to magenta:

```
<p id="first-paragraph" class="second-paragraph" style="color: magenta;">
  The CSS box model is essentially a box that wraps around every HTML
  element. It consists of: borders, padding, margins, and the actual
  content.
</p>
```

The result:



CSS validator

As you have no doubt come to realise, as a developer it is very important to follow the syntax rules. The same is true of CSS. You need to follow the rules for formatting your CSS rules exactly or unexpected errors will occur when you try to view your web page in the browser. Examples of common errors include misspelling the name of an element, not having matching opening and closing braces `{}`, or leaving out semicolons (`;`) or colons (`:`). You are bound to make mistakes that will violate these rules and that will cause problems when you try to view web pages in the browser. We all make syntax errors, and often. Being able to identify and correct these errors becomes easier with time and is an extremely important skill to develop.

To help you identify errors in your CSS, use this helpful **CSS validator tool**.

Extra resource

You will be required to do some research on CSS layouts to complete this task successfully. Feel free to use whatever resources you like, but this **introduction to CSS layout** and the **CSS layout cookbook** are excellent places to find help. You may also consult **Web Style Sheets CSS tips & tricks** by W3C and the additional reading provided in your folder.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select “Request Review”, the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you’ve done that, feel free to progress to the next task.

Auto-graded task

Your task is to enhance the appearance of your webpage by using CSS to style and position the elements on your `index.html` web page that you created in the HTML Overview task.

Follow the guidelines below to ensure that your page meets the expected standards. **You must style at least 50% of the following elements** to meet the task requirements:

- Apply a different font family (e.g., Arial, Verdana, or Times New Roman) to the text on your page.
- Modify the color of your main headings (`<h1>`) and subheadings (`<h2>`). Adjust the font size of the headings to create a visual hierarchy.
- Change the text color of the paragraphs (`<p>`). Ensure the color contrasts well with the background for readability.
- Set a background color for the entire webpage or for specific sections, such as the body or the headings.
- Adjust the alignment of your text (e.g., center, left, or right alignment) to improve the overall layout.
- Style specific words or sentences within your paragraphs using bold (`<b>`) or italics (`<em>`) to emphasise important content.
- Add or adjust the margin and padding around elements like headings and paragraphs to create space and prevent clutter.
- Change the color of your links (`<a>`) and add an underline or remove it to make them stand out or blend in with the text.

Evaluation Criteria:

Your webpage will be evaluated based on the following criteria:

- **Visual Appeal**: How well have you used color, font, and spacing to create an attractive page?
- **Readability**: Is the text easy to read? Have you chosen appropriate colors and text alignment?
- **CSS Application**: Have you correctly applied the required CSS styles to at least 50% of the suggested elements?

Feel free to use style libraries like **Bootstrap** or **Bootstrap Studio**, if you like. This will involve doing a bit of research. A great place to start is **this video tutorial**.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Reference list

Yesiller, G. (2022, July). *CSS box model.* Medium.
https://medium.com/@gokselyesiller/css-box-model-67b8cbbce528

Mozilla Developer Network. (2023, November). *Grid.* mdn web docs.
https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids#A_CSS_Grid_grid_framework

Web Style Guide. (1997). *Color Display Primer*. Web Style Guide.
https://webstyleguide.com/wsg1/graphics/display_primer.html

Flyme. (2016, November). *Pixels and megapixels – explained!* Flyme.
http://forum.flymeos.com/thread-11200-1-1.html