Hyperiondev

TASK

CSS Overview

Visit our website

Introduction

WELCOME TO THE CSS OVERVIEW TASK!

In the previous task you added HTML elements to a webpage, but did not learn how to make the page look good to a viewer. Don't worry - in this task, you will learn to use CSS to position and style the elements that you have added to your webpage to make it much more attractive and exciting! You will also learn how to create responsive and flexible web designs using the grid and flexbox layout.

INTRODUCTION TO CSS

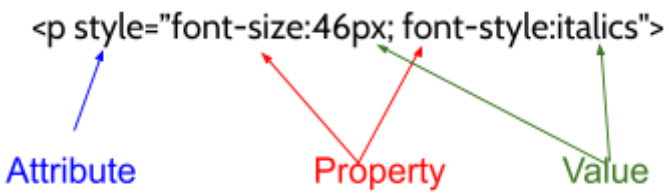
Cascading Style Sheets (CSS) is a language that is used to change the presentation and look of a particular document that has been written in a markup language such as HTML. CSS is usually applied to web pages, but can also be used in other areas, such as to format XML documents.

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, including font-family (Arial, Times New Roman, etc.), font-style (bold, 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 and after a colon specify the value for that property.



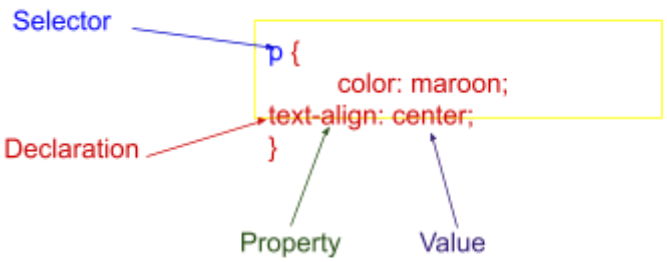
When you style an element individually by changing that element's properties, it is known as **inline styling**. Inline styling allows you to specify the style of an individual element in the line where that element is declared. What if you wanted to apply similar styles to all elements of a certain type? For example, 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. The example below 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.



- 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.

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

```
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.

Open `example.html`. Notice that in this file we have several elements for which either the `class` or `ID` attribute has been defined. For example, notice how there are several `<a>` elements that have the class attribute set to "`moreButton`" as shown below.

```
<a href="" class="morebutton">more</a>
```

However, there are also `<a>` elements that do not have a class attribute specified.

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

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

```
.moreButton {
  font-weight: bold;
}
```

When you use a class selector, the name of the class will always be preceded by a dot, '`.` '. The style rule will cause all elements where the class attribute has been set to `class="moreButton"` to have bold text.

Similarly, you could create a stylesheet that uses ID selectors. ID selectors start with a hash, '`#`', as is shown in the example below.

```
#firstmorebutton {
  font-weight: bolder;
  font-family: cursive;
}
```

The rule above would cause the text of the element where the ID attribute equals "`firstmorebutton`" to be bold and cursive.

Note: Although you can have many elements that have a class attribute with the same value, each ID name must be unique within the document!

You could use a combination of internal CSS (declared in the head of your HTML document) and inline style. How would the style rules apply? Essentially, the closer to the element the style is, the higher the precedence. For example, if you had the internal CSS rule shown in the code above in your page but you wanted one paragraph to be styled differently from the rest, you would simply use inline style for that one paragraph and that would overwrite the rule specified by the internal CSS.

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. Here, "`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 stylesheet. "`type`" tells the browser what type of file it is and how to interpret it.

INTERNAL, EXTERNAL OR INLINE: WHICH APPROACH IS THE BEST?

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. Is it better to use internal or external CSS? Generally, it is ***better to use external CSS wherever possible***. Why? *Readability* is an important factor. Imagine trying to read through different languages (CSS, HTML, Python) simultaneously in one file. Rather separate them — it's much easier to follow what's happening, especially when building fancy websites with plenty of different styles.

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

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

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

You may find, though, that it is necessary to use a combination of external, internal, and inline styles. In this case, it is important to understand the concept of cascade.

Extra resource

When creating websites, remember that you have the flexibility to design the web pages using a wide range of CSS properties. You can customise the appearance of the font, adjust the dimensions of images, or even add interactive hover effects when moving the mouse over HTML elements.

Here are two cool CSS tricks:

1. If you would like to apply a certain style rule to an element when it is 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). Explore a **list of pseudo-classes.**

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. Learn more about using a **descendant combinator**.

Note: CSS enhances the visual appearance and usability of the webpage which improves user experience. Therefore, it is good practice to always attractively style your web pages to keep users engaged. Here are useful resources to further understand **CSS basics** and help you to apply various **CSS properties** when styling HTML elements.

RESPONSIVE AND FLEXIBLE DESIGN LAYOUTS

Now that you have understood how to use the different methods of applying CSS styles to HTML elements, let's explore how we can arrange and lay out elements on the webpage using flexible and responsive layouts. Responsive design is a method

of creating a web application that is able to adapt to different screen resolutions while maintaining interactivity. In other words, the exact layout will vary, but the relationship between elements and the reading order remains the same. Here are two flexible designs to ensure that the layout remains consistent:

GRID LAYOUT

A CSS grid is like a table that is designed to make it easier to position elements on a webpage. The grid usually contains 12 columns. The position of an element is described in terms of which row it is in and how many columns it takes up.

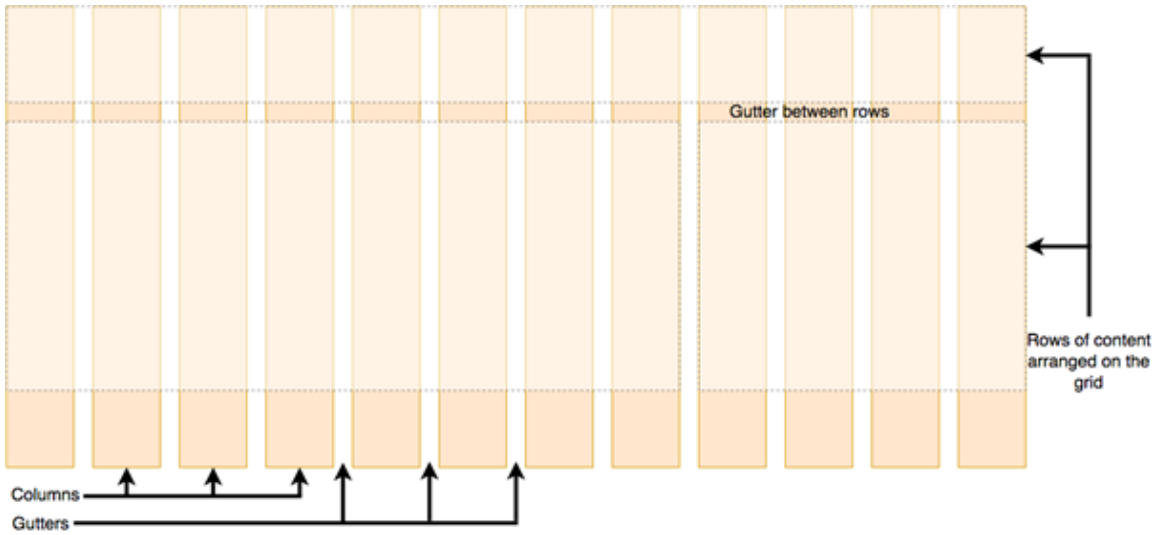


Image source:
https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids#A_CSS_Grid_grid_framework

The grid layout is a powerful tool for creating fluid grids. A fluid grid breaks down the width of the page into several equally sized and spaced columns. You can then position the page content according to these columns. Each fluid column expands accordingly when the viewport expands horizontally, as does the content within the columns.

In the example below, we have defined a grid layout using the following properties:

- **display:** to define a grid we use the grid value of the display property. In the code below, the parent element which has a class of "`grid-container`" will define rows and columns within a container. The children elements which have a class of "`grid-items`" will occupy the cells within this container.
- **grid-template-columns:** this property is used to specify that there are three equal columns in the grid. Note: `fr` represents a fraction of space taken up by each column.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Grid Example</title>
    <style>
      .grid-container {
        display: grid;
        grid-template-columns: repeat(3, 1fr);
        background-color: purple;
      }
      .grid-item {
        background-color: plum;
        text-align: center;
        border: 1px solid black;
        padding: 10vh;
        font-size: 3vh;
      }
    </style>
  </head>
  <body>
    <div class="grid-container">
      <div class="grid-item">1</div>
      <div class="grid-item">2</div>
      <div class="grid-item">3</div>
      <div class="grid-item">4</div>
      <div class="grid-item">5</div>
      <div class="grid-item">6</div>
      <div class="grid-item">7</div>
    </div>
  </body>
</html>
```

FLEXBOX LAYOUT

Flexbox is a CSS module designed to more efficiently position multiple elements, even when the size of the contents inside the container is unknown. Items in a flex container expand or shrink to fit the available space.

Unlike grid layouts which use columns, a flexbox uses a single-direction layout to fill the container. A flex container expands items to fill the available free space or shrinks them to prevent overflow.

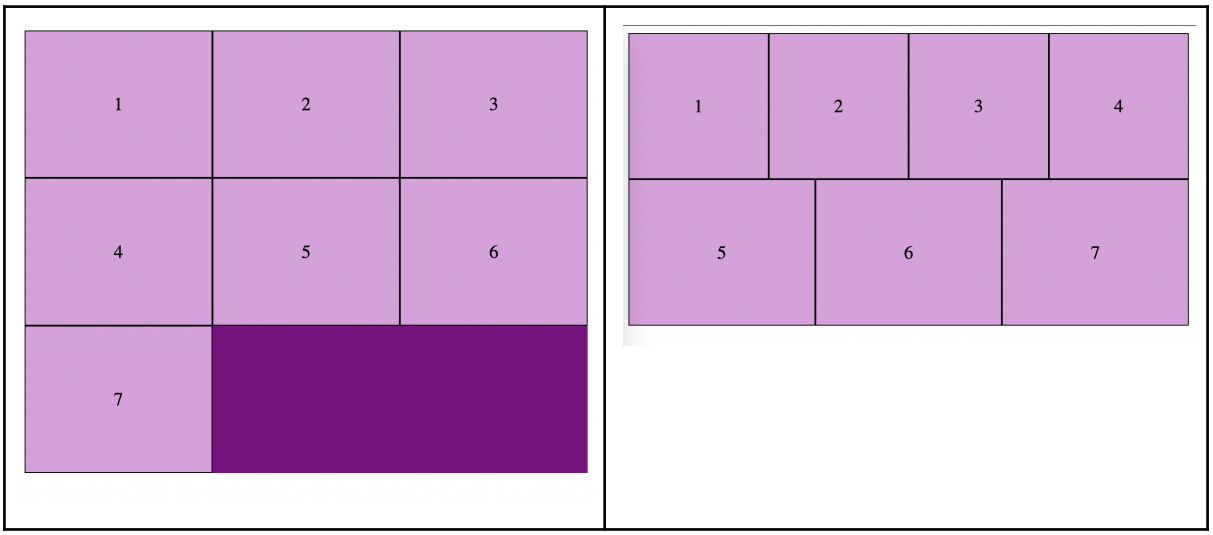
In the example below, we have defined a flexbox grid using the following properties:

- **display:** to define a flexbox, the "`flex`" value of the display property is used. In the code below, the parent element which has a class of "`flex-container`" will contain flexible children elements. The children elements, which have a class of "`flex-item`", will be aligned in one direction.
- **flex-flow:** the `flex-flow` property is a combination of the `flex-direction` and `flex-wrap` properties as it allows us to specify direction and wrapping. The `flex-direction` property establishes the main axis/direction by row or column, whereas the `flex-wrap` property allows items to wrap as needed, which is useful as by default flex items will all try to fit on one line.
- **flex:** We add the rule "`flex: 1`" to our flex item. The flex property is a unitless proportion value that dictates how much space each flex item will take up along the central axis compared to other flex items. In this case, we're giving each "`.flex-item`" element the same value of 1, which means they'll all take up an equal amount of the spare space left after properties like padding and margin have been set. An element with a flex value of 2 will take up twice as much of the available space.

```
<!DOCTYPE html>
<html>
  <head>
    <title>FlexBox Example</title>
    <style>
      .flex-container {
        display: flex;
        flex-flow: row wrap;
        background-color: purple;
      }
      .flex-item {
        background-color: plum;
        text-align: center;
        border: 1px solid black;
        padding: 10vh;
        font-size: 3vh;
        flex: 1;
      }
```

```
    </style>
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item">1</div>
      <div class="flex-item">2</div>
      <div class="flex-item">3</div>
      <div class="flex-item">4</div>
      <div class="flex-item">5</div>
      <div class="flex-item">6</div>
      <div class="flex-item">7</div>
    </div>
  </body>
</html>
```

Grid Container **Flex Container**



Let us now observe how each of these layouts are displayed in comparison to one another. The image on the left is a grid container made up of 7 grid items. We can see that the items are aligned with the three columns that have been declared. Take note of the available grid space, represented by a darker purple.

The image on the right is a flex container made of seven items. All items in a row share the available space equally and succeed in filling the box regardless of the screen dimensions.

Extra resource

Getting the layout of elements correct is one of the trickiest aspects of CSS. You will be required to do some research in this regard to complete the Compulsory Task successfully. Feel free to use any resources you like, but **Introduction to CSS layout** and the **CSS layout cookbook** are excellent places to find help.

CASCADE

As we know, CSS stands for cascading style sheets. You may have wondered why they are called *cascading* style sheets. Cascading has to do with how the rules are applied.

If your website contains external, internal and inline CSS, inline CSS overrides internal CSS rules and external CSS files. Internal CSS overrides external CSS rules. If there are conflicting rules regarding properties, 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 example, in a stylesheet that uses element selectors, class selectors and ID selectors, *element selectors are the least specific* (because they could match the most elements in a page) whereas ID selectors are the most specific. Therefore, ID selectors will be applied over class selectors and element selectors.

CSS VALIDATOR

As you have no doubt come to realise, as a web developer it is very important to follow syntax rules. Nowhere is this more applicable then when using CSS. You need to follow the rules for formatting your CSS exactly, or unexpected errors will occur when you try to view your web page in the browser. Examples of common errors include spelling the name of an element incorrectly, 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. 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 **tool**.

Extra resource

The additional reading for this task includes two excellent resources:

- An eBook entitled "**CSS notes for professionals**"
- **Web Style Sheets CSS tips & tricks: Centering things** by W3C.

Instructions

Read and run the accompanying example files provided before doing the Compulsory Task, to become more comfortable with the concepts covered in this task.

Compulsory Task

Follow these steps:

- Use CSS to style and position the elements on your webpage that you created in the HTML task (**index.html**) as attractively as possible.

- Ensure that you apply the following styling to your resume:
    - **Colour:** change the background colour of the entire page.
    - **Font:** modify the font according to your preference. You are welcome to change the font family, colour, size, weight and style.
    - **Image size:** adjust the size of the image. Feel free to modify the width, height and border-radius properties as you please.
    - **Positioning:** centralise the navbar and place the elements horizontally next to each other. Ensure to adjust and space out the content appropriately in a presentable manner. You may consider using the margin and padding property.
    - **Layout:** use a flexbox OR grid layout to make your resume responsive to different screen sizes.

- **Note before you choose a layout:** if you opt to use the flexbox layout, you could apply the `flex-flow` property with a value of `row-wrap` to arrange the children elements vertically. If you use the grid layout, you can apply the `grid-template columns` property to specify the number of columns in the grid. This will assist with arranging the children elements within the parent container.

- Ensure all properties are visually pleasing in appearance as this plays a role in positively influencing user experience.

Completed the task(s)?

Ask an expert code reviewer to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.