

# CSS

Morteza Ghodoosi

# What is CSS?

---

CSS stands for Cascading Style Sheets.

CSS allows you to define styles for web pages.

It can be used to style text, change colors, font size, etc.,  
as well as create layout - for example, create columns, sidebars, navigation  
menus, etc.

# What is CSS?

---

CSS uses rules to style elements.

A rule is a group of style declarations.

An example rule could define all paragraphs to be red and large.

The style declarations have the following syntax:

```
property: value;
```



# What is CSS?

---

Each declaration specifies a property of the element(s) we are targeting, then a value that we'd like to give the property.

There are a number of ways we can apply CSS rules to our elements.

The first way is to define the style declarations right on the element we target, using the style attribute.

# What is CSS?

---

For example:

```
<p style="color:red">some text</p>
```

Run the code to see the style applied to the paragraph.

We can have multiple declarations in the style rule:

```
<p style="color:red; font-size:26px">some text</p>
```

Note: The declarations are separated using a semicolon.

Task: Change the code to apply a blue color to the paragraph and increase its font size to 42px.

# What is CSS?

---

Using the style attribute on HTML elements applies the style to that element only.

But what if we wanted to apply the same style to multiple paragraphs?

We would have to copy the same style attribute and define it on all the paragraphs.

This would make a lot of redundant code, and make it very hard to change the style in the future.



# What is CSS?

---

It is common practice to define the CSS rules in a separate document and link it with the HTML.

CSS allows you to target elements using selectors, which allows you to target specific elements and apply the same style to them.

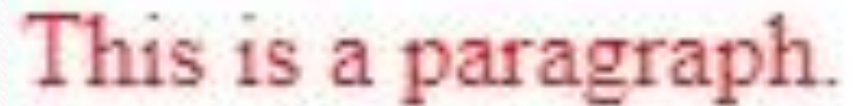
# CSS Selectors

---

The style rules are defined in a CSS file using selectors.

Let's look at an example:

```
p {  
  color: red;  
}
```



This is a paragraph.

The selector `p` targets all the paragraph elements of our HTML.



# CSS Selectors

---

Now, all the paragraphs of our HTML page will have a red color.

A rule can have multiple styles, for example:

```
p {  
  color: red;  
  font-size: 42px;  
}
```

Task: Open the code and modify the style to make all paragraphs green.

# CSS Selectors

---

We can target multiple selectors at once, by separating the selectors with a comma.

For example, let's make all h1 headings and paragraphs red:

```
h1, p {  
  color: red;  
}
```

Task: Add another paragraph to the HTML code and see the CSS style in action.

# Task

---

In the CSS code, select first header (your name) element and set the font size for it to 28px using the font-size property.



# Class & ID Selectors

---

So far we have seen element selectors, which use the element names and target all of them in the HTML.

Often we need to target only a subset of elements, without changing the others. This can be done using a class selector.

# Class & ID Selectors

---

We first need to add a class attribute to the elements we want to target:

```
<p class="intro">CSS is awesome</p>
```

```
<p>some other text goes here</p>
```

Now, we can create the class selector in CSS:

```
.intro {  
  color: red;  
  background-color: yellow;  
}
```

# Class & ID Selectors

---

Note, that the class selector needs to start with a dot, followed by the class name that we gave to our element.

You can apply the `.intro` style rule to any element on your page, by simply adding the class attribute.

Task: Modify the code and add the `.intro` rule to the last paragraph of the page, too.



# Class & ID Selectors

---

We can also create selectors based on the id attribute of elements.

This is done by using the hash (#) symbol before the id value:

```
#intro {  
  color: red;  
  background-color: yellow;  
}
```

This will target the element with id="intro" in our HTML.

You can have only one element with the same id on the page, while the class value can be used multiple times.

# Class & ID Selectors

---

We can also target elements based on their location.

For example, there are two `<strong>` elements, one inside a paragraph and the other inside a list item.

We can target only the one inside the list item by using the following rule:

```
li strong {  
  color: purple;  
}
```

The space denotes that the `<strong>` element needs to be inside an `<li>` element.

# Class & ID Selectors

---

CSS styles set on parent elements are inherited by the child elements.

For example, if you set a color and font on an element, every element inside it will also be styled with that color and font, unless you've applied different color and font values directly to them:

```
header {  
  color: red;  
  font-size: 24px;  
}  
<header>  
  <h1>some title</h1>  
  <p>some text</p>  
</header>
```



**some title**

some text



# Task

---

1. In the CSS code, select the element with id="job" (your job) and set its font-size to 18px.
2. Set the font size of the element with class="summary" (your summary) to 16px.

# Text color & Size

---

Let's look at some properties that are used to style text.

We have already seen the color property in action, which is used to change the color of the text.

```
.intro {  
  color: red;  
}
```

# Text color & Size

---

Besides color names, a color value can also be set using hex and rgb values.

Colors are displayed in combinations of red, green, and blue light (RGB).

Hex values are written using the hashtag symbol (#), followed by six hex characters, which define the components:

```
.intro {  
    color: #47759b;  
}
```



# Text color & Size

---

RGB defines the individual values for Red, Green, and Blue:

```
.intro {  
    color: rgb(71, 117, 155);  
}
```

Each component can take the value 0 to 255.

0 is the absence of the component,

so `rgb(0, 0, 0)` corresponds to black, while `rgb(255, 255, 255)` is the color white.

# Text color & Size

---

The font-size property is used to set the size of the text.

It can take values using pixels (px):

```
.intro {  
    font-size: 32px;  
}
```

# Text color & Size

---

The em size unit is another way to set the size of the text.

1em is equal to the font size set on the parent element of the current element we are styling.

By default, 1em = 16px



# Task

---

In the CSS code, define font colors for the heading elements and also, set font size of them to 20px.

# Styling Text

---

You can change the font of the text using the font-family property.

```
.intro {  
  font-family: Arial;  
}
```

The given font needs to be available on the user's computer to work.

There are only a certain number of fonts that are generally available across all systems.

These are called web safe fonts and include: Arial, Courier New, Georgia, Times New Roman, Trebuchet MS, Verdana.

# Styling Text

---

It's a common practice to define multiple fonts in the font-family property, so that if the first one is not available on for the user, the browser will try the next one:

```
.intro {  
    font-family: Arial, "Helvetica Neue", Helvetica, sans-serif;  
}
```



# Styling Text

---

If the browser does not support the font *Arial*, it tries the next font (*Helvetica Neue*, then *Helvetica*).

If the browser doesn't have any of them, it will try the generic sans-serif.

Note: If the font name consists of multiple words, it has to be in quotes.

# Styling Text

---

The font-style property is used to make text italic:

```
.intro {  
    font-style: italic;  
}
```

Task: Open the code and add multiple properties to the .intro class, making it Arial, 32px in size and red in color.

# Styling Text

---

The font-weight property is used to make the text bold.

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

You can also define the font weight with a number from 100 (thin) to 900 (thick). 400 is the same as normal, and 700 is the same as bold.

Task: Modify the code to use the number format for the font-weight property to make the text bold.



# Task

---

1. In the CSS code, select the body element and set the font you prefer using font-family property. This will specify the font for all of the content.
2. Make your job id bold using the font-weight property.

# Font Styles

---

The text-transform property allows you to transform text with the following values:

uppercase: Transforms the text to capitals.

lowercase: Transforms the text to lower case.

capitalize: Transforms all words to have the first letter capitalized.

# Font Styles

---

For example:

```
.intro {  
  text-transform: uppercase;  
}  
  
.outro {  
  text-transform: capitalize;  
}
```

THIS IS A PARAGRAPH.

This Is Another Paragraph.



# Font Styles

---

The text-decoration property supports the following values:

none: removes any decoration.

underline: underlines the text.

overline: gives the text an overline.

line-through: puts a strikethrough over the text.

# Font Styles

---

For example:

```
.intro {  
  text-decoration: underline;  
}  
  
.outro {  
  text-decoration: line-through;  
}
```

This is a paragraph.

~~This is another paragraph.~~

Task: Add a class to the middle paragraph and make its text-decoration overline.

# Font Styles

---

The text-shadow property allows you to set a shadow for the text.

It takes 4 values, like this:

```
h1 {  
  text-shadow: 3px 2px 3px red;  
}
```

The image shows the text "This is a Heading1." in a bold, black, serif font. It has a prominent red shadow effect applied, which is slightly offset to the right and bottom, giving it a 3D appearance. The shadow is a vibrant red color.

The first value is the horizontal offset of the shadow.

The second value - the vertical offset.

The third value is the blur radius: a higher value means the shadow is dispersed more widely.

The last one is the color of the shadow.

Task: Modify the code to change the color of the shadow to green.



# Font Styles

---

You can add multiple shadows, by separating them with a comma.

For example:

```
h1 {  
    text-shadow: 3px 2px 3px red,  
                2px 2px 1px blue;  
}
```

# Font Styles

---

The text-align property is used to control how text is aligned within its container.

The supported values are: left, right, center, justify.

For example:

```
h1 {  
    text-align: center;  
}
```

Task: Modify the code to align the last paragraph to the right.

# Font Styles

---

The line-height property sets the height of each line of text.

For example, let's set our line height to be 1.8 times the height of the font:

```
.intro {  
  line-height: 1.8;  
}
```

Lorem ipsum dolor sit amet consectetur, adipisicing elit. Sed, in natus. Dolorem quasi quibusdam  
tempora a quisquam. Quae minima adipisci consequuntur facilis, qui, fuga suscipit placeat,  
quibusdam quas sint labore.

The recommended line height is 1.5 - 2 (double spaced.).



# Font Styles

---

The letter-spacing and word-spacing properties allow you to set the spacing between letters and words in your text.

For example:

```
.intro {  
  letter-spacing: 3px;  
  word-spacing: 3px;  
}
```

Lorem ipsum dolor sit amet consectetur, adipisicing elit. Sed, in natus.  
Dolorem quasi quibusdam tempora a quisquam. Quae minima adipisci  
consequuntur facilis, qui, fuga suscipit placeat, quibusdam quas sint  
labore.

Task: Add a rule to make the letter spacing of the heading text 2px.

# Task

---

1. change the text of the h1 element to be uppercase using the text-transform property.
2. Set the line height of the summary section to the value 20px.
3. Align the text of the header and footer elements to the center.

# The Box Model

---

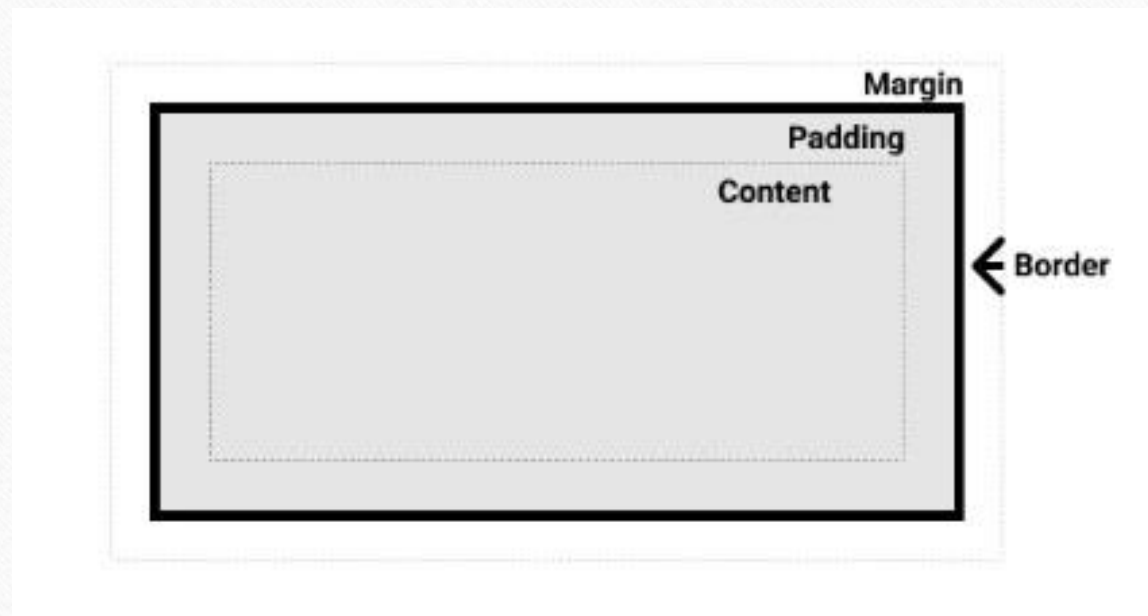
HTML elements appear as boxes.

The box model defines how the different parts of the box - margin, border, padding, and content, work together to create the box.



# The Box Model

Here are the parts of the box:



# The Box Model

---

The content box is the area where the content is displayed.

The padding box is the white space around the content.

The border box comes after the padding.

The margin box is the last one and defines the space between this box and other elements.

We will see how to define these boxes and control their size.

# The Box Model

---

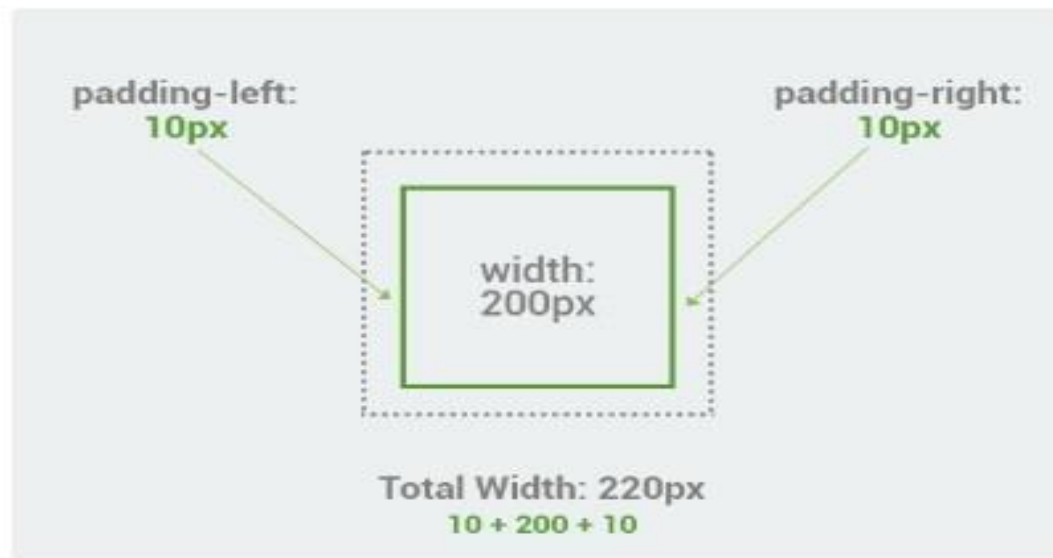
In the standard box model, the height and width attributes define the size of the content box.

Any padding and border is then added to that width and height to get the total size of the box.



# The Box Model

For example, the total width of the box with paddings will be the sum of width plus padding left and padding right.



# The Box Model

---

The margin is not counted towards the actual size of the box, it is the space outside the box.

The margin is the space around your box.

We can control all margins of an element at once using the margin property, in the following order: top, right, bottom, and left.





# The Box Model

---

Now, let's give the paragraph some margins and see how it behaves:

```
p {  
  margin: 5px 10px 5px 10px;  
  border: 3px solid black;  
}  
.box {  
  border: 3px solid red;  
}
```



# The Box Model

---

Margins can have positive or negative values.

We defined borders for our boxes to see them and their margins.

The margins can be provided using separate properties.

# The Box Model

---

The following style:

```
margin: 5px 10px 5px 10px;
```

is equivalent to:

```
margin-top: 5px;
```

```
margin-right: 10px;
```

```
margin-bottom: 5px;
```

```
margin-left: 10px;
```

Task: Modify the code to also give margins to the outside box.




# The Box Model

---

The padding is the area between the border and the content.

It is defined similar to the margin:

```
p {  
  margin: 5px 10px 5px 10px;  
  padding: 4px 7px 4px 7px;  
  border: 3px solid black;  
}
```

A diagram illustrating the box model for a paragraph. It shows a rectangular box with a thick black border. Inside the box, the text "some text" is centered. The box represents the padding area, which is the space between the border and the content.

some text

Task: Change the padding to be 10px on all sides.

# Box Size

---

Similar to using HTML attributes, the width and height can be given using CSS, with the width and height properties.

They can take both pixel and percentage values.

For example, let's give the container box 50% of the page width:

```
.box {  
  width: 50%;  
  border: 3px solid red;  
}
```

Remember, the final width and height of the box are calculated by adding the padding and border widths.

# Box Size

---

Similar to the width, we can also define the height of the elements.

For example:

```
.box {  
  width: 50%;  
  height: 200px;  
  border: 3px solid red;  
}
```



# Box Size

---

It can be inconvenient to calculate the actual size of the box, because of the padding and border widths being added to it.

For this reason, CSS provides an alternative box model where the given width is taken by the box and includes the padding and border.

The model is turned on using box-sizing: border-box:

```
p {  
  margin: 5px 10px 5px 10px;  
  padding: 4px 7px 4px 7px;  
  width: 200px;  
  box-sizing: border-box;  
  border: 3px solid black;  
}
```

# Box Size

---

Now, the paragraph box has the width 200px, irrespective of the padding and border widths.

Without the box-sizing property it would have the width:  $200 + 7 + 3 + 7 + 3 = 220\text{px}$ .

Let's compare the two box models and create two paragraphs, one using the standard box model and the other one using border-box.

# Box Size

---

We will use the same styles for both boxes:

```
.box1 {  
  border: 3px solid red;  
  padding: 10px 10px 10px 10px;  
  margin: 5px 5px 5px 5px;  
  width: 200px;  
  height: 100px;  
}
```


```
.box2 {  
  border: 3px solid red;  
  padding: 10px 10px 10px 10px;  
  margin: 5px 5px 5px 5px;  
  width: 200px;  
  height: 100px;  
  box-sizing: border-box;  
}
```



# Box Size

---

We will use the same styles for both boxes:



BOX 1

BOX 2

# Task

---

1. Add a padding of 10px from left and right, 2px from top and bottom for the section and footer elements.
2. Set the margin of the ul element to 0, as it has some default margins.

# Borders

---

The border property allows you to set the border style for the elements.

```
.box {  
  border: 3px solid red;  
}
```

The property takes 3 values:

- the width in pixels,
- the border style,
- the border color.

If not defined, the border is not visible.



# Borders

---

The border style can take one of the following values:

dotted, dashed, solid, double, groove, ridge, inset, outset.

For example:

```
.box {  
  border: 3px dashed green;  
}
```



Task: Modify the code and set the style of the border to groove.

# Borders

---

Similar to paddings and margins, the border supports separate properties to set the border for each side, e.g.: border-top, border-bottom, etc.

For example:

```
.box {  
  border-top: 3px dashed green;  
  border-bottom: 5px solid red;  
}
```

Each side of the border can have its own style.

# Borders

The border-radius property allows you to round the corners of elements.

For example:

```
.box {  
  border: 3px solid green;  
  border-radius: 10px;  
}
```



BOX

A single value provided uses that value for all 4 corners.

You can also set the border radius for each corner separately, by separating them with spaces.

The radius applies to the whole element, even if it has no border set.



# Task

---

1. Add a 1px solid border to the bottom of the heading elements.
2. To make them visually more appealing, add a 2px padding to the bottom of them.

# Background

---

The background-color property sets the background color for an element.

```
.box {  
  background-color: red;  
  height:100px;  
  padding: 10px;  
}
```

You can set the color using color names, hex values and rgb.

Task: Open the code and change the background color of the box to blue.

# Background

---

You can also set an image as the background of an element, using the background-image property.

It has the following syntax:

```
.box {  
  background-image: url('sl-logo.png');  
  height:150px;  
  padding: 10px;  
}
```



# Background

---

The image address has to be put in quotes and enclosed by the `url()` attribute.

A small image is, by default, tiled to fill the box.

In case the image is larger than the element, it is not scaled down by default and we will see only the part that fits in the box.

# Background

---

The background-repeat property can control how the image is tiled (or repeated) in the box.

It can take the following values:

no-repeat: does not tile the image.

repeat-x: repeat horizontally.

repeat-y: repeat vertically.

repeat: repeat in both directions (default value).

# Background

---

Let's change our background to repeat only horizontally:

```
.box {  
  background-image: url('sl-logo.png');  
  background-repeat: repeat-x;  
  height: 150px;  
  padding: 10px;  
}
```

Task: Modify the code to remove the tiling of the background image.



# Background

---

The background-position property sets the position in which the background image appears.

It uses a coordinate system in which the top-left corner of the box is at position (0,0).

As its value, it can take both pixel/percentage units and keywords (top, left, right, bottom, center).

# Background

---

Let's position our background image at the horizontal and vertical center of the box:

```
.box {  
  background-image: url('sl-logo.png');  
  background-repeat: no-repeat;  
  background-position: center center;  
  height: 150px;  
  padding: 10px;  
  border: 1px solid black;  
}
```

The default background-position value is (0,0): the top-left corner.

# Background

---

The background-size property allows you to define how the image should fit inside the box.

It has two possible values:

cover: The image will completely cover the box, while maintaining its aspect ratio.

contain: The image will fit inside the box completely.



# Background

---

For example:

```
.box {  
  background-image: url('sl-logo.png');  
  background-repeat: no-repeat;  
  background-size: contain;  
  height:150px;  
  padding: 10px;  
  border: 1px solid black;  
}
```

Task: Modify the code to use the other value of the background-size property and see how it changes the result.

# Task

---

1. Set the background color of the header and footer elements.
2. To make the header titles breathe, add top and bottom paddings of 2px.

# Styling Lists

---

By default, unordered list items are marked with round bullets, while ordered lists are numbered.

The `list-style-type` property allows you to change the markers to circle, square, decimal, disc, lower-alpha, lower-roman, etc.

```
ul {  
  list-style-type: square;  
}
```

- [Telegram](#)
- [Instagram](#)
- [Twitter](#)
- [LinkedIn](#)
- [GitHub](#)

Task: Modify the code to use roman numerals for the list using the lower-roman value.



# Styling Lists

---

The list-style-position property specifies the position of the bullets: inside or outside.

```
ul {  
    list-style-type: circle;  
    list-style-position: inside;  
}
```

The default value is outside.

- 
- [Telegram](#)
  - [Instagram](#)
  - [Twitter](#)
  - [LinkedIn](#)
  - [GitHub](#)

# Styling Lists

---

The list-style-image property can be used to specify an image to be used as the list item marker.

```
ul {  
  list-style-image: url("bullet.jpg");  
}
```

However, this property does not allow you to control the position and size of the image.

If you want to use an image as the marker,

it's better to set the list-style-type to none and add a background image to the list items with the marker image,

positioning it to the left.

# Styling Lists

---

The list-style property is a shorthand for setting list-style-type, list-style-image and list-style-position, all in one declaration:

```
ul {  
  list-style: square outside none;  
}
```

This would be the same as:

```
ul {  
  list-style-type: square;  
  list-style-position: outside;  
  list-style-image: none;  
}
```



# Task

---

1. In the CSS code, set the color of the text of the bold elements.
2. Add a bottom margin of 5px to the list items.

# Links

---

A link is by default blue in color and underlined.

We can style links with CSS properties (e.g., color, font-family, background, etc.).

```
a {  
    color: green;  
}
```

# Links

---

In addition, links can be styled differently, depending on what state they are in. The following selectors are available:

`a:link` - defines the style for normal unvisited links.

`a:visited` - defines the style for visited links.

`a:active` - a link becomes active once you click on it.

`a:hover` - a link is hovered when the mouse is over it.

These are called pseudo selectors.



# Links

---

Let's add a background color to the links, when they are hovered:

```
a:hover {  
    background-color: yellow;  
}
```

Task: Modify the code to give a black border on the links, when they are hovered over.

# Links

---

By default, all links are underlined.

We can remove the underline using the text-decoration property:

```
a {  
    color: green;  
    text-decoration: none;  
}
```

Task: Modify the code to add the underline to the links when they are hovered.

# Links

---

It is common to style links as buttons and use them for navigation menus.

First, we put the links inside an unordered list:

```
<ul>  
  <li><a href="#">Home</a></li>  
  <li><a href="#">About</a></li>  
  <li><a href="#">Contacts</a></li>  
</ul>
```



# Links

---

Now, the CSS:

```
li {  
  display: inline;  
}  
a:hover {  
  background: blue;  
}
```

We set the display property to inline for list items to make them sit on the same line with the others, acting as inline elements.

```
a {  
  outline: none;  
  text-decoration: none;  
  display: inline-block;  
  margin-left: 1%;  
  text-align: center;  
  padding: 10px;  
  box-sizing: border-box;  
  width: 31%;  
  color: white;  
  background-color: green;  
}
```

# Task

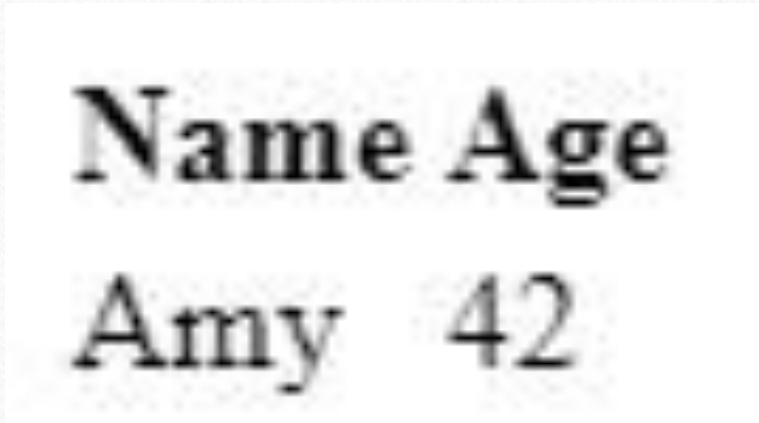
---

1. In the CSS code, set the color of the links.
2. Also, remove the underline using the text-decoration property.
3. For the hover effect, set the text-decoration to underline.

# Styling Tables

An HTML table doesn't look appealing by default:

```
<table>
  <tr>
    <th>Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Amy</td>
    <td>42</td>
  </tr>
</table>
```



| Name | Age |
|------|-----|
| Amy  | 42  |



# Styling Tables

---

Let's add some basic styles to our table:

```
table {  
  border-collapse: collapse;  
  width: 100%;  
}  
td, th {  
  border: 1px solid black;  
  text-align: center;  
}
```

| Name | Age |
|------|-----|
| Amy  | 42  |

# Styling Tables

---

The table cells have spacing between them by default, making double borders appear.

The border-collapse property sets the borders to collapse into one, making the table look a lot better.

We can also provide some padding to the cells, to give the content some space to "breathe":

```
td, th {  
    border: 1px solid black;  
    padding: 10px;  
    text-align: center;  
}
```

# Styling Tables

---

We have also aligned the text in all cells to the center.

Now, we can assign a background color to our heading cells:

```
th {  
    background-color: #d3d3d3;  
}
```

Task: Modify the code to make the heading texts appear white with a black background.



# Styling Tables

---

We can make the rows even more interesting!

CSS provides the `nth-child()` pseudo selector, which allows you to select specific rows and style them.

For example, we can style the 3rd row like this:

```
tr:nth-child(3) {  
    background-color: #b0d1e4;  
}
```

The rows are counted from the top to the bottom, including the heading rows.

The `nth-child` selector allows you to also define rules for the rows, instead of providing the number.

# Styling Tables

---

For example, we can style the odd and even rows separately, giving them different background colors:

```
tr:nth-child(even) {  
    background-color: #b0d1e4;  
}  
tr:nth-child(odd) {  
    background-color: #6aabd1;  
}
```

This will automatically apply the assigned styles to the odd and even rows.

Task: Add some new rows to the table and see how they take the style automatically.

# Task

---

1. Set the background-color of the table header cells and the text color. Also, align the text to the center.
2. Set the border-collapse property to collapse on the table.
3. Add a 1px solid border to the td and th elements. Also, add a padding of 5px for all sides.



# Styling Forms

Let's take a sample form and add some style to it.

Here is the HTML:

```
<form>  
  <label for="name">Name: </label>  
  <input type="text" id="name" placeholder="John Smith">  
  <br><br>  
  <label for="email">Email: </label>  
  <input type="email" id="email" placeholder="name@domain.com">  
  <br><br>  
  <input type="button" value="Send">  
</form>
```



Name: John Smith

Email: name@domain.com

Send

# Styling Forms

---

The text fields have a default size and a black border.

We can add styling to the text fields and labels by selecting them using their name:

```
input {  
  border: 1px solid green;  
  width: 100%;  
  padding: 5px;  
}  
label {  
  font-weight: bold;  
  color: green;  
}
```

**Name:**

John Smith

**Email:**

name@domain.com

Send

# Styling Forms

---

The style will also apply to the button, as it is also an input element.

We can target the input elements by their type using the following syntax:

```
input[type='text'] {  
  border: 1px solid green;  
  width: 100%;  
  padding: 5px;  
}
```

This will only apply the style to `<input type="text">` elements.

Task: Modify the code to also apply the style to the email type inputs.



# Styling Forms

---

We can also style the text inside the text fields, just like any other text.

Let's make it green and bold:

```
input[type='text'], input[type='email'] {  
  border: 1px solid green;  
  width: 100%;  
  padding: 5px;  
  color: green;  
  font-weight: bold;  
}
```

# Styling Forms

Similarly, we can apply a style to our button:

```
input[type='button'] {  
  background-color: green;  
  border: none;  
  border-radius: 20px;  
  padding: 10px 20px;  
  font-weight: bold;  
  color: white;  
}
```



**Name:**  
John Smith

**Email:**  
name@domain.com

**Send**

We used `border:none` to remove the default border from the button.

# Styling Forms

---

Just like link states, we can use the `:hover` pseudo selector to style the button, when it is hovered:

```
input[type='button']:hover {  
    background-color: gray;  
}
```

Now, the button will turn gray when we move the mouse over it.

Task: Modify the code and round the corners of the text fields by adding the `border-radius` property, similar to the button.



# Task

---

1. Add a 1px solid border to the text and email inputs. Set a width of 150px to them, a padding of 5px and a bottom margin of 5px.
2. Apply the same style to the textarea field, and set its width to 203px.
3. For the label elements, set the text color and make them bold.
4. For the submit button, set the border and set a border-radius of 7px. Make the button's text white and bold and set a bottom margin of 7px.

# CSS Layout

---

As we now know, there are block and inline elements.

The block elements span the entire width of their parent and start from a new line,

while the inline elements are placed inside block elements and act similar to their content, sitting on the same line.

# CSS Layout

---

For example, let's create paragraphs of text and make some words in them bold:

`<p>Learning <b>CSS</b> is fun!</p>`

`<p>some text </p>`

`<p>another <b>text</b></p>`

`<p>` is a block element, while `<b>` is inline.



# CSS Layout

Let's add background colors and borders to the <p> and <b> elements to see how they are laid out on the page:

```
p {  
  background-color: red;  
  border: 1px solid blue;  
  padding: 5px;  
}  
  
b {  
  background-color: yellow;  
  border: 1px dashed black;  
}
```

Learning **CSS** is fun!

some text

another **text**

# CSS Layout

The display property can be used to change how elements behave, making them behave as block or inline elements. For example, let's make the inline `<b>` elements behave as block elements:

```
b {  
  background-color: yellow;  
  border: 1px dashed black;  
  display: block;  
}
```



Learning  
**CSS**  
is fun!

some text

another  
**text**

Task: Make the paragraphs act as inline elements and check out the result!

Recall, in a previous example we used the display property to change the list items behavior and make them inline.

# CSS Layout

---

Another value of display is flex.

It is used to arrange elements in rows and columns.

Consider three article elements on the page, wrapped inside a section:

```
<section>
```

```
  <article>First section</article>
```

```
  <article>Second section</article>
```

```
  <article>Third section</article>
```

```
</section>
```

They are block elements and will appear on separate lines, each of them taking the whole width of the section.



# CSS Layout

---

We have also added some colors and styles to the CSS, so we can distinguish the article elements.

Now, if we add `display:flex;` to the section, it will make the articles inside it act as flex items and arrange in columns:

```
section {  
  display: flex;  
}
```

First sectionSecond sectionThird section

The flex items take equal widths, to fill up the width of their container.

Task: Remove one of the articles and see how the layout is changed.

By default, the flex items take equal space.

We can control what proportion of space flex items take up compared to the other items using the flex property.

# CSS Layout

---

For example, let's give the first article the flex value of 2, while the other two will have the flex value of 1:

```
article {  
  background-color: blue;  
  color: white;  
  padding: 20px;  
  margin: 10px;  
  flex: 1;  
}  
.special {  
  flex: 2;  
}
```



Now, the first article will take 2 times as much space as the other ones.

# CSS Layout

---

Let's create a real website layout. It will include a header and footer, spanning the whole width of the page, and two columns for the main content, one taking up 3 times as much space as the other.

Here is the HTML:

```
<header>My Header</header>
<main>
  <section class="content">Main Section</section>
  <section class="side">Side Section</section>
</main>
<footer>My Footer</footer>
```

The layout will also act responsive and maintain the structure for any browser width.

Task: Open the code and check out the CSS styles to understand how it works.



# CSS Layout

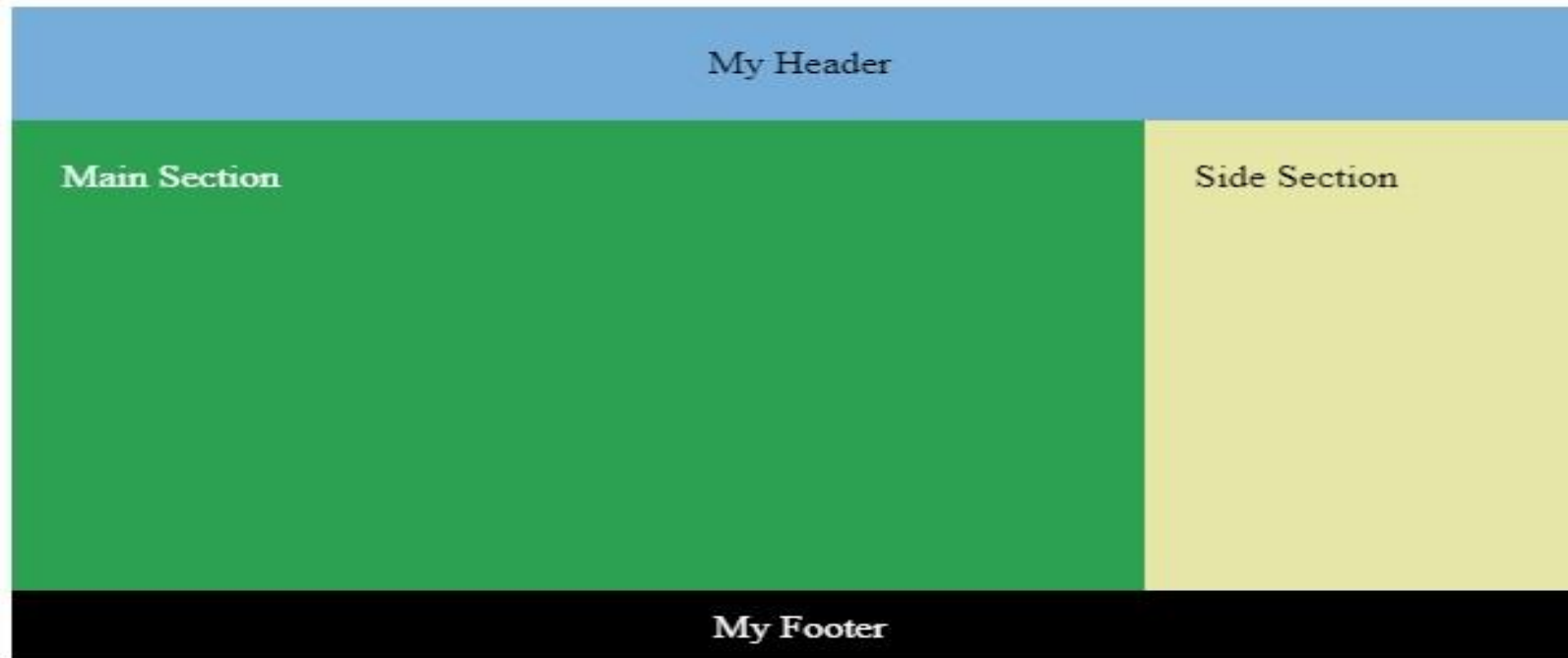
---

```
header {  
  background-color: #77adda;  
  text-align: center;  
  padding: 20px;  
}  
  
footer {  
  background-color: black;  
  color: white;  
  text-align: center;  
  padding: 10px;  
}
```

```
.content {  
  background-color: #2ca152;  
  color: white;  
  padding: 20px;  
  height: 200px;  
  flex: 3;  
}  
  
.side {  
  background-color: #e6e6a5;  
  padding: 20px;  
  flex: 1;  
}  
  
main {  
  display: flex;  
}
```

# CSS Layout

---



# Positioning

---

The float property allows you to float an element inside a container.

For example, we can float an image in a text, to make the text wrap on one side of it:

```
img {  
    float: left;  
}
```

The supported values are left and right.

You can float any other element with the same technique.



# Positioning

---

To fine-tune the position of elements on the page, the position property can be used.

There are a number of different values the position property supports, changing the way the element is positioned.

By default, all elements have the position:static, which is the normal flow.

With position: relative; the element is positioned relative to its normal position.

The properties top, right, bottom, and left can be used to specify how the element will be shifted.

# Positioning

For example:

```
.special {  
  position: relative;  
  bottom: 30px;  
  background-color: orange;  
}
```

First section

Second section

Third section

This will shift the element 30px from the bottom.

The content of relatively positioned elements can be moved and overlap other elements, but the reserved space for the element is still preserved in the normal flow.

# Positioning

An absolutely positioned element is removed from the normal flow of the page.

This is useful, for example, to create popup boxes, control menus, panels, etc.

For example:

```
.special {  
  position: absolute;  
  top: 30px;  
  left: 20px;  
  background-color: orange;  
}
```



This will make the element position 30px from the top of the page and 20px from the left.

The element's position is relative to the first positioned parent element: for example,

you could absolutely position an element inside a relatively positioned container.

Task: Modify the code to position the element at the bottom right corner.



# Positioning

Fixed positioning makes the element stick to a part of the visible area, which means it always stays in the same place even if the page is scrolled.

For example:

```
.special {  
  position: fixed;  
  bottom: 30px;  
  background-color: orange;  
}
```



This makes the element stick to the bottom of the page, shifted by 30px.

Task: Modify the code to make the element span the entire width of the page and stick at the top.

# Positioning

---

When elements are positioned outside the normal flow, they can overlap other elements.

The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).

For example:

```
article {  
  position: relative;  
  z-index: 2;  
}  
.special {  
  z-index: 1;  
  position: absolute;  
  top: 30px;  
  left: 50%;  
}
```

# Positioning

---

By default, the orange box would be above the article elements, however we made it go under them using a lower z-index value.

Task: Provide a class name for the third article and make it go under the orange box.



# Task

---

1. Change the position of the header element to be fixed, stick it to the top and left of the page and set a width of 100%.
2. To make the other sections appear below it, give the body a bottom margin of 100px.
3. Another styling tweak we need to do is float the profile image in the summary text to the right.

# Transforms

---

CSS transforms allow you to translate, rotate, scale, and skew elements.

A transformation is an effect that lets an element change shape, size, and position.

Let's take a look at the rotate transformation.

Here is a div element:

```
div {  
  width: 200px;  
  height: 100px;  
  margin-top: 30px;  
  background-color: #32CD32;  
}
```



# Transforms

---

Now let's rotate the div by 10deg:

```
div {  
  width: 200px;  
  height: 100px;  
  margin-top: 30px;  
  background-color: #32CD32;  
  transform: rotate(10deg);  
}
```





# Transforms

---

The `rotate()` method rotates an element clockwise or counter-clockwise, according to a given degree. Negative value will result in a counterclockwise rotation.

```
div.positive {
```

```
  ...
```

```
  transform: rotate(10deg);
```

```
}
```

```
div.negative {
```

```
  ...
```

```
  transform: rotate(-10deg);
```

```
}
```

# Transforms

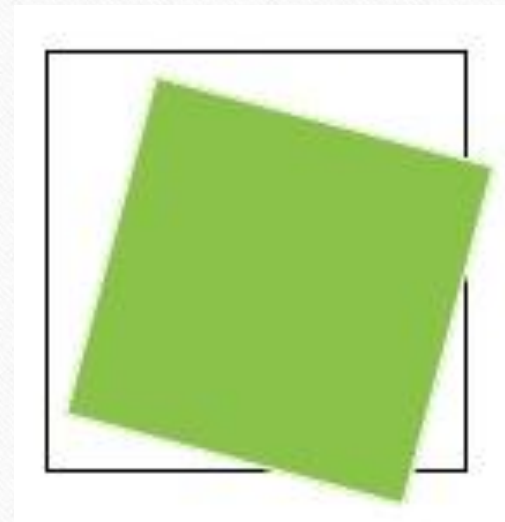
The transform-origin property allows you to change the position of transformed elements.

For example:

```
div.empty-div {  
  position: relative;  
  ...  
}  
  
div.green-div {  
  position: absolute;  
  ...  
  transform: rotate(15deg);  
  transform-origin: 25% 75%;  
}
```

In the example above, we use the transform-origin property together with transform-rotate.

The origin of the x-axis (horizontal) is set to 25% from the left. The origin for the y-axis (vertical) is set to 75% from above.



# Transforms

---

The default value for the property is 50% 50%, which corresponds to the center of the element.

The transform-origin property must be used together with the transform property.

The translate() method moves an element from its current position (according to the parameters given for the x-axis and the y-axis).



# Transforms

---

In this example below, the div element is moved 100px to the right and 50px down:

```
div {  
  width: 200px;  
  background-color: palegoldenrod;  
  transform: translate(100px, 200px);  
}
```

Positive values will push an element down and to the right of its default position, while negative values will pull an element up and to the left of its default position.

An element can also be moved by setting the margins or by positioning the element, although translate is a better choice for animating elements.

# Transforms

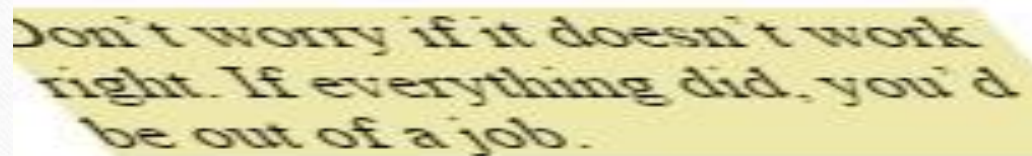
---

The `skew()` method skews an element along the x-axis and the y-axis by the given angles.

The following example skews the `div` element by 30 degrees along the X-axis:

```
transform: skew(30deg);
```

If the second parameter is not specified, it has a zero value.



Don't worry if it doesn't work  
right. If everything did, you'd  
be out of a job.

# Transforms

---

The `scale()` method increases or decreases the size of an element, according to the parameters given for the width and height. 1 stands for the original size, 2 for twice the original size, and so on.

```
div.first {  
    ...  
    transform: scale(0.7, 0.7);  
}  
div.second {  
    ...  
    transform: scale(1.5, 1.5);  
}
```



# Transforms

---

In the example below, we decreased the first div by the factor 0.7 both horizontally and vertically,

and increased the second div by a factor of 1.5.

Multiple transforms can be used at once.

Applying multiple transforms to an element is simple; just separate them using spaces.

For example, let's rotate and translate at the same time:

```
transform: rotate(45deg) translate(100px);
```

You can use as many transforms as you want.

# Transitions

---

CSS transitions allow you to change from one property value to another over a given duration.

Transition effects can be applied to a wide variety of CSS properties, including background-color, width, height, opacity, and many more.

# Transitions

---

In the example below, the div element has width and height of 50px, with a green background. We specified a transition effect for the width property, with a duration of 3 seconds:

The CSS will look like this:

```
div {  
  width: 50px;  
  height: 50px;  
  background: #32CD32;  
  transition: width 3s;  
}  
div:hover {  
  width: 250px;  
}
```

If you hover over the div element, it will start to change its width.

When the cursor is moused out of the element, it will gradually change back to its original style.



# Transitions

---

The transition-timing-function property specifies the speed curve of the transition effect.

It can have the following values:

ease - the animation starts slowly, then accelerates quickly.

ease-in - starts slowly, then accelerates, and stops abruptly.

ease-out - starts quickly, but decelerates to a stop.

ease-in-out - similar to ease, but with more subtle acceleration and deceleration.

linear - constant speed throughout the animation; often best for color or opacity changes.

# Transitions

---

For example:

```
transition-timing-function: ease-in;
```

Run the code and hover the div to see the result.

If no timing function is specified, the default value is ease.

The transition timing function can also be specified in the transition property, like this:

```
transition: width 3s ease-in;
```

# Transitions

---

We can add a transition on a transform property!

Let's make our div rotate when the its hovered:

```
div {  
  ...  
  transition: transform 1s ease-in-out;  
}  
div:hover {  
  transform: rotate(90deg);  
}
```

We specify the transform to rotate the div by 90 degrees, and then declare the transition on our transform property.



# Task

---

1. Add a transform transition for our image element in the summary section which uses the ease function and has a duration of 1 second.
2. For the hover state, add a scale transform with the factor of 1.3

# Keyframes & Animations

---

CSS transitions are generally best for simple from-to movements, while CSS animations are for more complex series of movements.

With animations, you can change as many CSS properties as you want to, as many times you want to.

An animation is defined using keyframes which hold the styles the element will have at certain times.

Keyframes are defined using the `@keyframes` keyword, followed by a name of the animation.

# Keyframes & Animations

---

For example:

```
@keyframes colorchange {  
  [b]0%[/b] {background-color: red;}  
  [b]50%[/b] {background-color: yellow;}  
  [b]70%[/b] {background-color: blue;}  
  [b]100%[/b] {background-color: green;}  
}
```

The example animation will change the background color of an element three times: when the animation is 50% complete, 70% complete, and when the animation is 100% complete. colorchange is the name of the animation. You can choose any name for your animation.



# Keyframes & Animations

---

As an alternative to using percentages, you can use from and to keywords, where:

from is a starting offset of 0%

to is an ending offset of 100%.

The two examples below are equivalent, and produce the same result:

```
@keyframes colorchange {  
  0% {background-color: red;}  
  100% {background-color: green;}  
}  
  
@keyframes colorchange {  
  from {background-color: red;}  
  to {background-color: green;}  
}
```

To get an animation to work, you must bind the animation to an element.

# Keyframes & Animations

---

In the example below, the animation is set for the div element and lasts three seconds:

```
div {  
  ...  
  animation-name: colorchange;  
  animation-duration: 3s;  
}  
@keyframes colorchange {  
  0% {background-color: red;}  
  50% {background-color: green;}  
  100% {background-color: blue;}  
}
```

The animation-name property specifies the animation to be used for the element.

The animation-duration property specifies the duration of the selected animation.

# Animation

---

There are a number of properties we can set for the animation.

The animation-timing-function specifies the speed curve of an animation. It can have the following values:

ease - specifies an animation with a slow start, then fast, then end slowly (this is default)

linear - specifies an animation with the same speed from start to end

ease-in - specifies an animation with a slow start

ease-out - specifies an animation with a slow end

ease-in-out - specifies an animation with a slow start and end.



# Animation

---

For example:

animation-timing-function: ease-in;

animation-delay - defines the delay before an animation starts.

For example:

The animation-delay and animation-duration values can be defined in seconds (s) or milliseconds (ms).

The animation-iteration-count property determines the number of times an animation repeats.

For example:

animation-iteration-count: 5;

To make the animation repeat forever, just use the infinite value:

animation-iteration-count: infinite;

# Animation

---

The animation-direction indicates how the keyframe should be applied.

The values can be set as:

normal - the default value, which means it plays forward from 0 % to 100%.

reverse - plays the keyframe in an opposite direction from 100 % to 0%

alternate - the animation first runs forward, then backward, then forward.

alternate-reverse - the animation first runs backward, then forward, then backward.

For example:

animation-direction: reverse;

# Animation

---

Consider the following example:

```
div {  
  animation-name: colorchange;  
  animation-duration: 3s;  
  animation-timing-function: ease-in;  
  animation-delay: 1s;  
  animation-iteration-count: infinite;  
  animation-direction: reverse;  
}
```



# Animation

---

A single animation property can be used to achieve the same result as the above code:

```
div {  
    animation: colorchange 3s ease-in 1s infinite reverse;  
}
```

The order in which each property is declared in the shorthand declaration is important and cannot be altered,

or the animation will not work properly.