CSS Color

Foreground color & Background Color

```
h1 {
   color: Red; #foreground color
   background-color: Blue; #background color
}
```

Hexadecimal

A hex color begins with a hash character (#) which is followed by three or six characters. The characters represent values for red, blue and green.

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

DarkSeaGreen: #8FBC8F
Sienna: #A0522D
SaddleBrown: #8B4513
Brown: #A52A2A
Black: #000000 or #000
White: #FFFFFF or #FFF
Aqua: #00FFFF or #0FF

RGB Colors

```
h1 {
   color: rgb(23, 45, 23);
}
```

Here, each of the three values represents a color component, and each can have a decimal number value from 0 to 255. The first number represents the amount of red, the second is green, and the third is blue. These colors are exactly the same as hex, but with a different syntax and a different number system.

Hue Saturation and Lightness

HSL Syntax
The first number represents the degree of the hue, and can be between 0 and 360. The second and third numbers are percentages representing saturation and lightness respectively. Here is an example:

```
color: hsl(120, 60%, 70%);
```

Hue is the first number. It refers to an angle on a color wheel. Red is 0 degrees, Green is 120 degrees, Blue is 240 degrees, and then back to Red at 360. Saturation refers to the intensity or purity of the color. The saturation increases towards 100% as the point gets closer to the edge of the hue wheel (the color becomes more rich). The saturation decreases towards 0% as the point gets closer to the center of the hew wheel (the color becomes more gray). Lightness refers to how light or dark the color is. Halfway, or 50%, is normal lightness. Imagine a sliding dimmer on a light switch that starts halfway. Sliding the dimmer up towards 100% makes the color lighter, closer to white. Sliding the dimmer down towards 0% makes the color darker, closer to black.

Opacity and Alpha

HSL Scheme
color: hsla(34, 100%, 50%, 0.1);
➔ The first three values work the same as hsl. The fourth value (which we have not seen before) is the alpha. This last value is sometimes called the opacity. Alpha is a decimal number from zero to one. If alpha is zero, the color will be completely transparent. If alpha is one, the color will be opaque. The value for half transparent would be 0.5.

RGB Color scheme
color: rgba(234, 45, 98, 0.33); #last value is opacity

Alpha can only be used with HSL and RGB colors; we cannot add the alpha value to color: green color: #FFFFF.

CSS Typography

Font Family

To change the typeface of text on your web page, you can use the font-family property. The default typeface for many browsers is Times New Roman. It's a good practice to limit the number of typefaces used on a web page to 2 or 3. When the name of a typeface consists of more than one word, it must be enclosed in double quotes (otherwise it will not be recognized), like so:

```
h1 {
   font-family: Garamond;
}
```

These pre-installed fonts serve as fallback fonts if the stylesheet specifies a font which is not installed on a user's computer.

```
h1 {
   font-family: "Garamond", "Times", serif;
}
```

➔ Use the Garamond font for all <h1> elements on the web page.
If Garamond is not available, use the Times font.
If Garamond and Times are not available, use any serif font pre-installed on the user's computer.

Font Weight

Bold ; normal
OR
In numbers:
400 is the default font-weight of most text.
700 signifies a bold font-weight.
300 signifies a light font-weight.

```
p {
   font-weight: bold;
}
```

Font Style

Normal (default) ; italic

```
h3 {
   font-style: italic;
}
```

Word Spacing

```
h1 {
   word-spacing: 0.3em;
}
```

The default amount of space between words is usually 0.25em.

Letter Spacing

```
h1 {
   letter-spacing: 0.3em;
}
```

Text Transformation

Uppercase ; lowercase

```
h1 {
   text-transform: uppercase;
}
```

Text Alignment

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

left - aligns text to the left hand side of the browser.
center - centers text.
right - aligns text to the right hand side of the browser.

Line Height
- A unitless number, such as 1.2. This number is an absolute value that will compute the line height as a ratio of the font size.
- A number specified by unit, such as 12px. This number can be any valid CSS unit, such as pixels, percents, ems, or rems.

```
p {
   line-height: 1.4;
}
```

CSS Grid Essentials

Creating Grid
The grid container will be a parent element that contains grid items as children and applies overarching styling and positioning to them.
To turn an HTML element into a grid container, you must set the element's display property to grid (for a block-level grid) or inline-grid (for an inline grid). Then, you can assign other properties to lay out the grid.

```
.grid {
   border: 2px blue solid;
   width: 400px;
   height: 500px;
   display: grid;
}
```

Creating Columns
By default, grids contain only one column. If you were to start adding items, each item would be put on a new row; that's not much of a grid! To change this, we need to explicitly define the number of rows and columns in our grid.

```
.grid {
   display: grid;
   width: 500px;
```

```
  grid-template-columns: 100px 200px;    # Width of column
}

.grid {
  display: grid;
  width: 100px;
  grid-template-columns: 20px 40% 60px; # percentage of the entire grid's width.
}
```

Creating Rows

```
.grid {
  display: grid;
  width: 1000px;
  height: 500px;
  grid-template-columns: 100px 200px;
  grid-template-rows: 10% 20% 600px;
}
```

Grid Template

Specifying columns and rows at the same time

```
.grid {
  display: grid;
  width: 1000px;
  height: 500px;
  grid-template: 200px 300px / 20% 10% 70%;
}
```

Fraction

By using the fr unit, we can define the size of columns and rows as a fraction of the grid's length and width. This unit was specifically created for use in CSS Grid. Using fr makes it easier to prevent grid items from overflowing the boundaries of the grid.

```
.grid {
  display: grid;
  width: 1000px;
  height: 400px;
  grid-template: 2fr 1fr 1fr / 1fr 3fr 1fr;
}
```

Repeat

```
.grid {
  display: grid;
  width: 300px;
  grid-template-columns: repeat(3, 100px);
}
```

➔ three columns that are each 100 pixels wide

min max

```
.grid {
  display: grid;
  grid-template-columns: 100px minmax(100px, 500px) 100px;
}
```

➔ The second column will always be between 100 and 500 pixels wide.

Grid Gap

```
.grid {
   display: grid;
   width: 320px;
   grid-template-columns: repeat(3, 1fr);
   grid-column-gap: 10px;
}
```
➔ gap between columns : 10px

Finally, there is a CSS property grid-gap that can set the row and column gap at the same time. **grid-gap**: 20px 10px; will set the distance between rows to 20 pixels and the distance between columns to 10 pixels.

Multiple Row items
```
.item {
   grid-row-start: 1;
   grid-row-end: 3;
}
```
➔ In this example, the HTML element of class item will take up two rows in the grid, rows 1 and 2. The values that grid-row-start and grid-row-end accept are grid lines.
Row grid lines and column grid lines start at 1 and end at a value that is 1 greater than the number of rows or columns the grid has.

Grid Row
```
.item {
   grid-row: 4 / 6;
}
```

IS SAME AS

```
.item {
   grid-row-start: 4;
   grid-row-end: 6;
}
```

Grid Column
```
.item {
   grid-column-start: 4;
   grid-column-end: span 2;
}
```
➔ If you know where you want your grid item to start and how long it should be, use span!

Grid Area
```
.item {
   grid-area: 2 / 3 / 4 / span 5;
}
```
➔ 4 values (order important)
grid-row-start
grid-column-start
grid-row-end
grid-column-end

[SUMMARY]

grid-template-columns defines the number and sizes of the columns of the grid
grid-template-rows defines the number and sizes of the rows of the grid
grid-template is a shorthand for defining both grid-template-columns and grid-template-rows in one line
grid-gap puts blank space between rows and/or columns of the grid

grid-row-start and grid-row-end makes elements span certain rows of the grid
grid-column-start and grid-column-end makes elements span certain columns of the grid
grid-area is a shorthand for grid-row-start, grid-column-start, grid-row-end, and grid-column-end, all in one line

Advanced CSS Grid

Grid Template Areas

```
.container {
  display: grid;
  max-width: 900px;
  position: relative;
  margin: auto;
  grid-gap: 10px;
  grid-template-areas: "header header"
                       "nav    nav"
                       "left   right"
                       "footer footer";
  grid-template-columns: 200px 400px;
  grid-template-rows: 150px 200px 600px 200px;
}

h1, h2 {
  font-family: monospace;
  text-align: center;
}

header {
  background-color: dodgerblue;
  grid-area: header;
}

nav {
  background-color: beige;
  grid-area: nav;
}

.left {
  background-color: dodgerblue;
  grid-area: left;
}

.right {
  background-color: beige;
  grid-area: right;
}

footer {
  background-color: dodgerblue;
  grid-area: footer;
}
```

➔ .container
2-column, 4-row layout with these areas:
header (spans two columns in the first row)
nav (spans two columns in the second row)
left (spans one column on the left in the third row)
right (spans one column on the right in the third row)
footer (spans two columns in the fourth row)

justify items and content & align content

justify-items is a property that positions grid items along the inline, or row, axis. This means that it positions items from left to right across the web page.

start — aligns grid items to the left side of the grid area
end — aligns grid items to the right side of the grid area
center — aligns grid items to the center of the grid area
stretch — stretches all items to fill the grid area

```
<main>
   <div class="card">Card 1</div>
   <div class="card">Card 2</div>
   <div class="card">Card 3</div>
</main>

main {
   display: grid;
   grid-template-columns: repeat(3, 400px);
   justify-items: center;
}
```

We can use **justify-content** to position the entire grid along the row axis.

start — aligns the grid to the left side of the grid container
end — aligns the grid to the right side of the grid container
center — centers the grid horizontally in the grid container
stretch — stretches the grid items to increase the size of the grid to expand horizontally across the container
space-around — includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element
space-between — includes an equal amount of space between grid items and no space at either end
space-evenly — places an even amount of space between grid items and at either ends

```
<main>
   <div class="left">Left</div>
   <div class="right">Right</div>
</main>

main {
   display: grid;
   width: 1000px;
   grid-template-columns: 300px 300px;
   grid-template-areas: "left right";
   justify-content: center;
}
```

align-content positions the rows along the column axis, or from top to bottom.

start — aligns the grid to the top of the grid container
end — aligns the grid to the bottom of the grid container
center — centers the grid vertically in the grid container
stretch — stretches the grid items to increase the size of the grid to expand vertically across the container
space-around — includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element
space-between — includes an equal amount of space between grid items and no space at either end
space-evenly — places an even amount of space between grid items and at either end

```
<main>
   <div class="top">Top</div>
   <div class="bottom">Bottom</div>
</main>

main {
   display: grid;
   height: 600px;
   rows: 200px 200px;
   grid-template-areas: "top"
                                "bottom";
   align-content: center;
}
```

justify self & align self
justify-self specifies how an individual element should position itself with respect to the row axis. This property will override justify-items for any item on which it is declared.

align-self specifies how an individual element should position itself with respect to the column axis. This property will override align-items for any item on which it is declared.

align-self and justify-self accept the same values as align-items and justify-items.

Grid Auto Rows and Grid Auto Columns
grid-auto-rows specifies the height of implicitly added grid rows. grid-auto-columns specifies the width of implicitly added grid columns.
grid-auto-rows and grid-auto-columns accept the same values as their explicit counterparts, grid-template-rows and grid-template-columns. [pixels (px), percentages (%), fractions (fr), the repeat() function]

```
<body>
   <div>Part 1</div>
   <div>Part 2</div>
   <div>Part 3</div>
   <div>Part 4</div>
   <div>Part 5</div>
</body>
body {
   display: grid;
   grid: repeat(2, 100px) / repeat(2, 150px);
   grid-auto-rows: 50px;
}
```

Grid Auto Flow
grid-auto-flow specifies whether new elements should be added to rows or columns.

row — specifies the new elements should fill rows from left to right and create new rows when there are too many elements (default)
column — specifies the new elements should fill columns from top to bottom and create new columns when there are too many elements
dense — this keyword invokes an algorithm that attempts to fill holes earlier in the grid layout if smaller elements are added

[Summary]
grid-template-areas specifies grid named grid areas
grid layouts are two-dimensional: they have a row, or inline, axis and a column, or block, axis.
justify-items specifies how individual elements should spread across the row axis

justify-content specifies how groups of elements should spread across the row axis
justify-self specifies how a single element should position itself with respect to the row axis
align-items specifies how individual elements should spread across the column axis
align-content specifies how groups of elements should spread across the column axis
align-self specifies how a single element should position itself with respect to the column axis
grid-auto-rows specifies the height of rows added implicitly to the grid
grid-auto-columns specifies the width of columns added implicitly to the grid
grid-auto-flow specifies in which direction implicit elements should be created

CSS Animation

Transition

```
a {
   display: block;
   width: 300px;
   padding: 31px 5px;
   border-radius: 5px;
   margin: 20% auto;
   background-color: orange;
   text-align: center;
   font-family: Helvetica;
   font-size: 32px;
   color: MintCream;
   transition-property: background-color;
   transition-duration: 2s;
}

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

Join Now! ➔ when you hover on the button ➔ Join Now!

Different properties transition in different ways, for example:
- Color values, like color and background-color, will blend to a new color.
- Length values like font-size, width, and height will grow or shrink.

transition-delay Much like duration, its value is an amount of time. Delay specifies the time to wait before starting the transition.

transition-timing-function
ease-in — starts slow, accelerates quickly, stops abruptly
ease-out — begins abruptly, slows down, and ends slowly
ease-in-out — starts slow, gets fast in the middle, and ends slowly
linear — constant speed throughout