Building Trust & Careers

CSS

CASCADING STYLE SHEETS

COMPLETE CSS NOTES

BASICS OF CSS

WHAT IS CSS & WHY IT'S IMPORTANT?

CSS (Cascading Style Sheets) is a stylesheet language used to control the presentation of HTML documents. It allows developers to style web pages by defining colors, layouts, fonts, and spacing, ensuring better design and user experience.

CSS SYNTAX & STRUCTURE

A CSS rule consists of a selector and a declaration block.

```
selector {
  property: value;
}
```

Example :

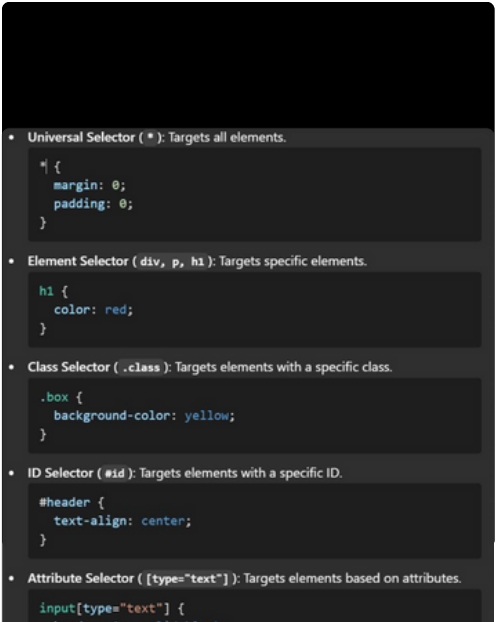
```
p {
  color: blue;
  font-size: 16px;
}
```

COMPLETE CSS NOTES

BASICS OF CSS

CSS SELECTORS

Selectors are used to target HTML elements



COMPLETE CSS NOTES

BASICS OF CSS

CSS SELECTORS

Selectors are used to target HTML elements

COMPLETE CSS NOTES

BASICS OF CSS

ADDING CSS TO HTML

There are three ways to apply CSS:

1. **Inline CSS** (within an HTML tag)

```
<p style="color: red;">This is red text.</p>
```

2. **Internal CSS** (inside `<style>` tag in HTML head)

```
<style>
  p {
    color: blue;
  }
</style>
```

3. **External CSS** (using a separate `.css` file)

```
/* styles.css */
p {
  color: green;
}
```

```
<link rel="stylesheet" href="styles.css">
```

COMPLETE CSS NOTES

BASICS OF CSS

SELECTOR PRECEDENCE (SPECIFICITY & IMPORTANCE)

- Inline styles have the highest specificity.
- ID selectors (#id) have higher priority than class selectors (.class).
- Class selectors have higher priority than element selectors (div, p).
- The !important rule overrides all other styles.

COMPLETE CSS NOTES

BASICS OF CSS

TEXT STYLING PROPERTIES

Font Properties:

- `font-family` : Specifies the font type.

```
p {
  font-family: Arial, sans-serif;
}
```

- `font-style` : Specifies text style (normal, italic, oblique).

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

- `font-weight` : Controls boldness (normal, bold, 100-900).

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

- `line-height` : Adjusts spacing between lines.

```
p {
  line-height: 1.5;
}
```

- `letter-spacing` : Adjusts spacing between letters.

```
p {
  letter-spacing: 2px;
}
```

COMPLETE CSS NOTES

BASICS OF CSS

TEXT STYLING PROPERTIES

Text Properties

- `text-align` : Aligns text (left, center, right, justify).

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

- `text-transform` : Changes case (uppercase, lowercase, capitalize).

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

- `text-decoration` : Adds underline, overline, or line-through.

```
p {
  text-decoration: underline;
}
```

- `text-shadow` : Adds a shadow to text.

```
h1 {
  text-shadow: 2px 2px 5px gray;
}
```

COMPLETE CSS NOTES

COLORS & UNITS IN CSS

COLOR REPRESENTATION

- Named colors: red, blue, green
- RGB format: rgb(255, 0, 0)
- Hex format: #ff0000
- HSL format: hsl(0, 100%, 50%)
- RGBA format: rgba(255, 0, 0, 0.5) (adds transparency)

CSS UNITS

- Absolute Units: px, cm, mm
- Relative Units: %, rem, em
- Viewport Units: vw, vh

COMPLETE CSS NOTES

BOX MODEL & LAYOUT

BOX PROPERTIES

- `margin` : Space outside an element.

```
div {
  margin: 10px;
}
```

- `padding` : Space inside an element.

```
div {
  padding: 10px;
}
```

- `box-sizing` : Defines box model behavior.

```
div {
  box-sizing: border-box;
}
```

COMPLETE CSS NOTES

BOX MODEL & LAYOUT

BORDER :

- border-width: Sets the thickness
- border-style: Defines the type (solid, dashed, dotted)
- border-color: Sets the color
- border-radius: Rounds corners

```
div {
  border: 2px solid black;
  border-radius: 10px;
}
```

COMPLETE CSS NOTES

BACKGROUND & SHADOWS

BACKGROUND PROPERTIES

- background-color: Sets background color
- background-image: Adds an image
- background-size: Defines image size (cover, contain)
- background-position: Positions image
- background-repeat: Repeats image (repeat, no-repeat).

```
body {
  background: url('image.jpg') no-repeat center/cover;
}
```

- linear-gradient: Creates gradient backgrounds

```
div {
  background: linear-gradient(to right, red, blue);
}
```

COMPLETE CSS NOTES

BACKGROUND & SHADOWS

SHADOW EFFECTS

- `box-shadow` : Adds shadow to elements.

```
div {
  box-shadow: 5px 5px 10px gray;
}
```

- `text-shadow` : Adds shadow to text.

```
h1 {
  text-shadow: 2px 2px 5px gray;
}
```

COMPLETE CSS NOTES

UNDERSTANDING BACKGROUND PROPERTIES

CSS provides various background properties to enhance the visual appeal of web pages. These properties allow developers to control the background image, size, positioning, and repeating behavior.

BACKGROUND-SIZE

Defines how a background image is scaled within an element.

- background-size: cover; → The image covers the entire element, maintaining its aspect ratio.
- background-size: contain; → The image scales to fit inside the element while maintaining its aspect ratio.
- background-size: 50% 50%; → The image is resized to 50% of the width and height of the element.

Example :

```
body {
  background-image: url('image.jpg');
  background-size: cover;
}
```

COMPLETE CSS NOTES

BACKGROUND-IMAGE

Used to set an image as the background of an element.

Example :

```
div {
  background-image: url('example.jpg');
  background-size: cover;
}
```

COMPLETE CSS NOTES

BACKGROUND-REPEAT

Controls whether the background image repeats and how.

- background-repeat: repeat; (default) → Image repeats both horizontally and vertically.
- background-repeat: no-repeat; → Image does not repeat.
- background-repeat: repeat-x; → Image repeats horizontally.
- background-repeat: repeat-y; → Image repeats vertically.

Example :

```
div {
  background-image: url('pattern.png');
  background-repeat: no-repeat;
}
```

COMPLETE CSS NOTES

BACKGROUND-POSITION

Defines the starting position of a background image.

- background-position: center; → Centers the image.
- background-position: top left; → Positions the image at the top-left.
- background-position: 50% 50%; → Places the image in the middle.

Example :

```
div {
  width: 100px;
  height: 100px;
  background-image: url('image.jpg');
  background-repeat: no-repeat;
  background-position: center;
}
```

COMPLETE CSS NOTES

LINEAR-GRADIENT

Creates a smooth transition of colors in a straight line.

Example :

```
div {
  background: linear-gradient(to right, red, blue);
}
```

RADIAL-GRADIENT

Creates a circular gradient effect.

Example :

```
div {
  background: radial-gradient(circle, red, blue);
}
```

COMPLETE CSS NOTES

PX VS %

- px (Pixels): A fixed unit of measurement. Useful when you need precise control over an element's size.
- % (Percentage): Relative to the parent container. Useful for responsive designs

Example :

```
div {
  width: 50%; /* 50% of the parent element's width */
  height: 200px; /* Fixed height */
}
```

COMPLETE CSS NOTES

WORKING WITH POSITIONAL PROPERTIES

- CSS position properties define how an element is positioned within its container.

POSITION: ABSOLUTE

- Removes the element from the normal document flow and positions it relative to the nearest positioned ancestor (or the viewport if no positioned ancestor exists).

Example :

```
div {
  position: absolute;
  top: 50px;
  left: 100px;
}
```

COMPLETE CSS NOTES

POSITION: RELATIVE

- Positions an element relative to its normal position.

Example :

```
div {
  position: relative;
  top: 20px;
  left: 30px;
}
```

TRANSFORM: TRANSLATE

- Moves an element without affecting surrounding elements.

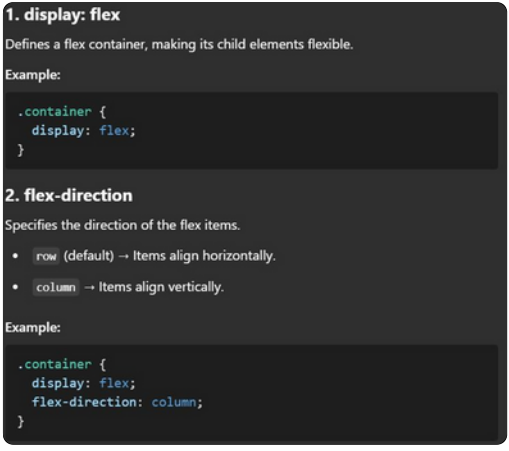
Example :

```
div {
  transform: translate(50px, 100px);
}
```

COMPLETE CSS NOTES

INTRODUCTION TO FLEXBOX FOR ALIGNMENT & STRUCTURE

- Flexbox is a powerful layout system in CSS that helps in aligning and distributing elements efficiently.



1. display: flex

Defines a flex container, making its child elements flexible.

Example:

```
.container {
  display: flex;
}
```

2. flex-direction

Specifies the direction of the flex items.

- `row` (default) → Items align horizontally.
- `column` → Items align vertically.

Example:

```
.container {
  display: flex;
  flex-direction: column;
}
```

COMPLETE CSS NOTES

INTRODUCTION TO FLEXBOX FOR ALIGNMENT & STRUCTURE

3. flex-wrap

Controls whether flex items wrap onto multiple lines.

- `nowrap` (default) → Items stay on a single line.
- `wrap` → Items wrap to the next line if needed.

Example:

```
.container {
  display: flex;
  flex-wrap: wrap;
}
```

4. flex-shrink

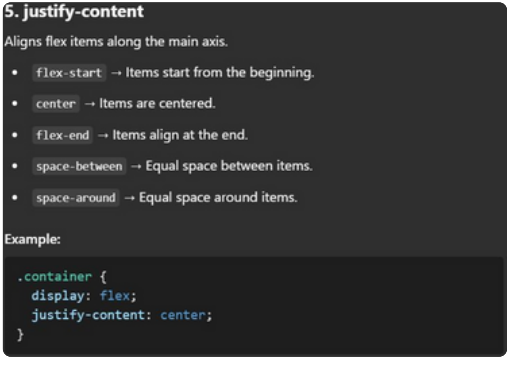
Determines how much a flex item can shrink if needed.

Example:

```
.item {
  flex-shrink: 2; /* Shrinks twice as fast as others */
}
```

COMPLETE CSS NOTES

INTRODUCTION TO FLEXBOX FOR ALIGNMENT & STRUCTURE



5. justify-content

Aligns flex items along the main axis.

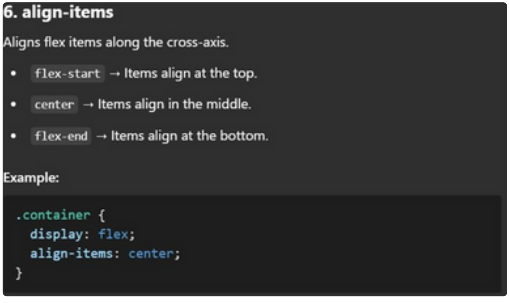
- `flex-start` → Items start from the beginning.
- `center` → Items are centered.
- `flex-end` → Items align at the end.
- `space-between` → Equal space between items.
- `space-around` → Equal space around items.

Example:

```
.container {
  display: flex;
  justify-content: center;
}
```

COMPLETE CSS NOTES

INTRODUCTION TO FLEXBOX FOR ALIGNMENT & STRUCTURE



6. align-items

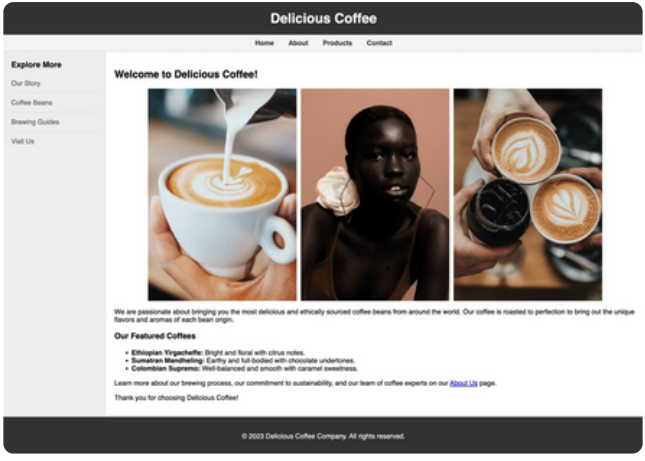
Aligns flex items along the cross-axis.

- `flex-start` → Items align at the top.
- `center` → Items align in the middle.
- `flex-end` → Items align at the bottom.

Example:

```
.container {
  display: flex;
  align-items: center;
}
```

PROJECT EXERCISE : TWO-COLUMN LAYOUT WITH FLEXBOX



INTERESTING THINGS ABOUT CSS

UNDERSTANDING PSEUDO-CLASSES & PSEUDO-ELEMENTS

WHAT ARE PSEUDO-CLASSES?

- Pseudo-classes allow you to apply styles to elements based on their state or user interaction. They start with a colon **:**

EXAMPLE:

```
/* Change color when a button is hovered */
button:hover {
  background-color: blue;
  color: white;
}

/* Style input when focused */
input:focus {
  border: 2px solid red;
}

/* Change link color when active */
a:active {
  color: green;
}
```

INTERESTING THINGS ABOUT CSS

UNDERSTANDING PSEUDO-CLASSES & PSEUDO-ELEMENTS

WHAT ARE PSEUDO-ELEMENTS?

- Pseudo-elements allow you to style specific parts of an element, such as inserting content before or after it. They start with **::**

EXAMPLE:

```
/* Add content before an h1 */
h1::before {
  content: "🔥 ";
  color: orange;
}

/* Add content after an h1 */
h1::after {
  content: " 🎉";
}
```

INTERESTING THINGS ABOUT CSS

LEARNING CSS TRANSITIONS

WHAT ARE TRANSITIONS?

- CSS transitions allow property changes to occur smoothly over a specified duration instead of happening instantly.

EXAMPLE:

```
/* Add content before an h1 */
h1::before {
  content: "🔥 ";
  color: orange;
}

/* Add content after an h1 */
h1::after {
  content: " 🎉";
}
```

INTERESTING THINGS ABOUT CSS

LEARNING CSS TRANSITIONS

WHAT ARE TRANSITIONS?

- CSS transitions allow property changes to occur smoothly over a specified duration instead of happening instantly.

EXAMPLE:

```
/* Add content before an h1 */
h1::before {
  content: "🔥 ";
  color: orange;
}

/* Add content after an h1 */
h1::after {
  content: " 🎉";
}
```

TRANSITION PROPERTIES

- transition-property: Defines which property to animate (e.g., color, background, all)
- transition-duration: Specifies the time the transition takes (e.g., 0.5s, 1s)
- transition-timing-function: Defines the speed curve (ease, linear, ease-

INTERESTING THINGS ABOUT CSS

UNDERSTANDING CSS TRANSFORMATIONS

The transform property allows you to modify the appearance of an element by moving, rotating, scaling, or skewing it.

Transform Functions

- translate(x, y): Moves the element
- rotate(deg): Rotates the element
- scale(x, y): Enlarges or shrinks the element
- skew(x, y): Skews the element along the X and Y axis

EXAMPLE:

```
/* Move element 50px right and 20px down */
.box {
  transform: translate(50px, 20px);
}

/* Rotate element by 45 degrees on hover */
.box:hover {
  transform: rotate(45deg);
}

/* Scale element 1.5 times when active */
.box:active {
  transform: scale(1.5);
}
```

INTERESTING THINGS ABOUT CSS

UNDERSTANDING CSS ANIMATIONS

What are Animations?

CSS animations allow elements to change styles over time without user interaction.

Animation Properties

- @keyframes: Defines the steps of the animation
- animation-name: Specifies the animation name
- animation-duration: Defines how long the animation lasts
- animation-timing-function: Controls the speed curve
- animation-iteration-count: Specifies how many times the animation runs (infinite, 1, 2, etc.)

EXAMPLE:

```
/* Define animation */
@keyframes slide {
  0% { transform: translateX(0); }
  50% { transform: translateX(100px); }
  100% { transform: translateX(0); }
}

/* Apply animation */
.box {
  animation: slide 2s infinite ease-in-out;
}
```

PROJECT EXERCISE : STYLE YOUR FORM!

UNDERSTANDING CSS ANIMATIONS

Description:

Make your form look good with CSS! Add colors, rounded corners, and make it interactive. Learn to style different parts of the form when someone uses it (like when they hover or click).

Hints:

- **Pseudo-classes (:hover, :focus, :active)**
  - Change how things look when you point at them (hover), click in them (focus), or click on them (active). Think about button colors or input field borders.
- **CSS Transitions:**
  - Make style changes smooth, not instant. Use transition to control how styles change over time.
- **Transforms (2D):**
  - Move, rotate, or scale things! Try making your button slightly bigger or move a bit when you hover using transform.
- **Transforms (3D - Challenge):**
  - Make things look like they have depth. This is harder! Look at perspective and rotateX to make elements tilt. (Optional!)
- **CSS Animation (Challenge):**
  - Make things move on their own! Use @keyframes to create simple moving effects, like a pulsing button. (Optional!)

RESPONSIVE WITH CSS

RESPONSIVE DESIGN

Mobile-First Approach

Purpose:

Prioritizes mobile user experience, improves performance, and keeps the codebase cleaner.

Techniques:

- Start styling for mobile devices first, without media queries.
- Use @media (**min-width: value**) for larger screen styles.

EXAMPLE:

```
body {
  font-size: 16px;
  padding: 10px;
}

@media (min-width: 768px) {
  body {
    font-size: 18px;
    padding: 20px;
  }
}
```

RESPONSIVE WITH CSS

RESPONSIVE DESIGN

@media Queries

Purpose:

Apply styles dynamically based on screen width or other device characteristics.

Techniques:

- @media (min-width: value): Styles apply for screens wider than the given value.
- @media (max-width: value): Styles apply for screens narrower than the given value.
- Define breakpoints as SCSS variables for better maintainability.

EXAMPLE:

```
$breakpoint-tablet: 768px;

.container {
  width: 100%;
}

@media (min-width: $breakpoint-tablet) {
  .container {
    width: 80%;
  }
}
```

RESPONSIVE WITH CSS

RESPONSIVE DESIGN

Grid

Purpose:

Provides two-dimensional layouts with rows and columns for complex designs.

Techniques:

- display: grid
- grid-template-columns: repeat(3, 1fr)
- grid-gap: 10px

EXAMPLE:

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 10px;
}
```