### Chapter 1: What is CSS?

* **CSS (Cascading Style Sheets)**: A language to style HTML documents.

### Chapter 2: Selectors

* **Element Selector (e.g.,**p**)**: Styles all <p> elements.
* **Class Selector (e.g.,**.className**)**: Styles all elements with the specified class.
* **ID Selector (e.g.,**#idName**)**: Styles the element with the specified ID.
* **Universal Selector (**\***)**: Styles all elements on the page.
* **Attribute Selector (e.g.,**[type="text"]**)**: Styles elements with a specific attribute.
* **Pseudo-class (e.g.,**:hover**)**: Styles an element when in a certain state (like mouse hover).
* **Pseudo-element (e.g.,**::before**)**: Styles a specific part of an element, like content before it.

### Chapter 3: Colors

* **Color Names (e.g.,**red**)**: Sets the color using predefined names.
* **RGB (e.g.,**rgb(255, 0, 0)**)**: Sets color using red, green, and blue values.
* **RGBA (e.g.,**rgba(255, 0, 0, 0.5)**)**: Sets color with transparency using red, green, blue, and alpha.
* **Hexadecimal (e.g.,**#FF0000**)**: Sets color using a hex code.
* **HSL (e.g.,**hsl(120, 100%, 50%)**)**: Sets color using hue, saturation, and lightness.
* **HSLA (e.g.,**hsla(120, 100%, 50%, 0.5)**)**: Sets color using hue, saturation, lightness, and alpha transparency.

### Chapter 4: Units & Sizes

* **Pixels (e.g.,**16px**)**: Sets a fixed size in pixels.
* **Percent (e.g.,**50%**)**: Sets size relative to the parent element.
* **Em (e.g.,**1.5em**)**: Sets size relative to the font size of the element.
* **Rem (e.g.,**1.5rem**)**: Sets size relative to the root font size.
* **Viewport Width (e.g.,**50vw**)**: Sets size as a percentage of the viewport's width.
* **Viewport Height (e.g.,**50vh**)**: Sets size as a percentage of the viewport's height.

### Chapter 5: Box Model

* **Content**: The area where text or media is displayed.
* **Padding**: Space between the content and the border.
* **Border**: Surrounds the padding and content.
* **Margin**: Space outside the border.
* **Box Sizing (**box-sizing: border-box;**)**: Includes padding and border in the element's total width and height.

### Chapter 6: Typography

* **Font Size (e.g.,**font-size: 16px;**)**: Sets the size of the font.
* **Font Weight (e.g.,**font-weight: bold;**)**: Sets the thickness of the font.
* **Font Style (e.g.,**font-style: italic;**)**: Sets the style of the font (normal, italic, oblique).
* **Font Variant (e.g.,**font-variant: small-caps;**)**: Controls the use of small-caps fonts.
* **Font Stretch (e.g.,**font-stretch: condensed;**)**: Adjusts the width of the font.
* **Font Family (e.g.,**font-family: "Helvetica Neue", Arial, sans-serif;**)**: Specifies the typeface with fallbacks.

### Chapter 7: Styling Links

* **Link States (e.g.,**:link, :visited**)**: Styles links based on their state (unvisited, visited).
* **Hover State (e.g.,**:hover**)**: Styles links when hovered over.

### Chapter 8: List Styles

* **Ordered List (e.g.,**<ol>**)**: Creates a numbered list.
* **Unordered List (e.g.,**<ul>**)**: Creates a bulleted list.
* **List Style Type (e.g.,**list-style-type: disc;**)**: Changes the bullet or numbering style.

### Chapter 10: Display

* **Display Property Values (e.g.,**display: block;**)**: Changes the display behavior of an element (block, inline, flex, grid, etc.).
* **Block**: The element takes up the full width available, starting on a new line.
* **Inline**: The element takes up only as much width as necessary, allowing other elements to sit beside it.
* **Flex**: The element becomes a flex container, enabling flexible layouts for its children.
* **Grid**: The element becomes a grid container, allowing for complex layouts using rows and columns.
* **Inline-block**: The element behaves like an inline element but can have width and height set.

### Chapter 11: Floats

* **Float Property (e.g.,**float: left;**)**: Moves an element to the left or right side of its parent, allowing text and inline elements to wrap around it.
* **Clear Property (e.g.,**clear: left;**)**: Prevents elements from wrapping around floated elements, moving the element below the floated element.

### Chapter 12: Columns

* **Column Count Property (e.g.,**column-count: 3;**)**: Divides content into a specified number of columns.
* **Column Gap Property (e.g.,**column-gap: 10px;**)**: Adjusts the space between columns.

### Chapter 13: Position

* **Position Property Values (e.g.,**position: relative;**)**: Changes the positioning behavior of an element (static, relative, absolute, fixed, sticky).
* **Static**: The default positioning; elements are positioned in the order they appear in the document flow.
* **Relative**: Positions the element relative to its original position.
* **Absolute**: Positions the element relative to the nearest positioned ancestor.
* **Fixed**: Positions the element relative to the viewport, remaining in place when scrolling.
* **Sticky**: Toggles between relative and fixed positioning based on the scroll position.
* **Top, Right, Bottom, Left Properties (e.g.,**top: 10px;**)**: Adjusts the position of an element from its specified position.

### Chapter 14: Flexbox

* **Flex Container Properties (e.g.,**display: flex;**)**: Creates a flexible container that allows for responsive layouts.
* **Flex Direction (e.g.,**flex-direction: row;**)**: Defines the direction flex items are placed in the flex container.
* **Justify Content (e.g.,**justify-content: center;**)**: Aligns flex items along the main axis.
* **Align Items (e.g.,**align-items: stretch;**)**: Aligns flex items along the cross axis.
* **Flex Item Properties (e.g.,**flex-grow: 1;**)**: Controls the growth and shrinkage of flex items.

### Chapter 15: Grid Layout

* **Grid Container Properties (e.g.,**display: grid;**)**: Creates a grid container for layout management using rows and columns.
* **Grid Template Columns (e.g.,**grid-template-columns: repeat(3, 1fr);**)**: Defines the number and size of columns in the grid.
* **Grid Gap (e.g.,**grid-gap: 10px;**)**: Sets the spacing between grid items.
* **Grid Item Properties (e.g.,**grid-column: 1 / 3;**)**: Controls the placement of grid items within the defined grid structure.

### Chapter 16: Images

* **Responsive Images (e.g.,**max-width: 100%;**)**: Ensures images scale properly to fit within their container, maintaining their aspect ratio.

### Chapter 17: Media Queries

* **Media Query Syntax (e.g.,**@media (max-width: 600px) { ... }**)**: Applies styles based on device characteristics, such as screen size or orientation.

### Chapter 18: Card Project

* **Card Layout Design**: A project to design a visually appealing card layout using CSS concepts, integrating various styling techniques.

### Chapter 19: Pseudo

* **Pseudo-classes (e.g.,**:hover**)**: Styles an element based on its state (e.g., when hovered over).
* **Pseudo-elements (e.g.,**::before**)**: Styles a specific part of an element, allowing for additional content without adding extra HTML.

### Chapter 20: Variables

* **CSS Variables (e.g.,**--variable-name: value;**)**: Defines reusable custom properties for consistent styling.
* **Variable Usage (e.g.,**var(--variable-name)**)**: Uses a defined variable in styles to maintain consistency and ease of updates.

### Chapter 21: Functions

* **CSS Functions**: Functions in CSS allow for dynamic calculations and manipulations of property values.
  + calc(): Performs calculations to determine CSS property values (e.g., width: calc(100% - 20px);).
  + min(): Returns the smallest value from a set of values (e.g., width: min(100%, 500px);).
  + max(): Returns the largest value from a set of values (e.g., height: max(200px, 50vh);).
  + clamp(): Sets a value that can grow and shrink within a defined range (e.g., font-size: clamp(1rem, 2vw + 1rem, 2rem);).

### Chapter 22: Animations

* **Keyframes**: Define styles at various points in an animation sequence.
  + @keyframes: Specifies the styles for the animation at different stages (e.g.,

css

VerifyOpen In EditorEditCopy code

1@keyframes fadeIn {

2 from { opacity: 0; }

3 to { opacity: 1; }

4}

5```).

* **Animation Properties**: Control the behavior of animations.
  + animation-name: Specifies the name of the @keyframes animation (e.g., animation-name: fadeIn;).
  + animation-duration: Defines how long the animation should take (e.g., animation-duration: 2s;).
  + animation-timing-function: Specifies the speed curve of the animation (e.g., animation-timing-function: ease-in;).
  + animation-delay: Sets a delay before the animation starts (e.g., animation-delay: 1s;).
  + animation-iteration-count: Defines how many times the animation should repeat (e.g., animation-iteration-count: infinite;).
  + animation-direction: Specifies whether the animation should play forwards, backwards, or alternate (e.g., animation-direction: alternate;).
  + animation-fill-mode: Determines how styles are applied before and after the animation (e.g., animation-fill-mode: forwards;).

### Chapter 23: Organization

* **Best Practices for Organizing CSS Files and Styles**: Effective organization enhances maintainability and readability.
  + **Consistent Naming Convention**: Use a clear and consistent naming scheme for classes and IDs (e.g., BEM methodology: block\_\_element--modifier).
  + **Separate Styles into Different Files**: Organize CSS into multiple files based on functionality (e.g., reset.css, layout.css, components.css).
  + **Use a Preprocessor**: Utilize CSS preprocessors like SASS or LESS to write more maintainable and modular CSS.
    - **SASS Syntax**: Allows for nesting, variables, and mixins (e.g.,

scss

VerifyOpen In EditorEditCopy code

1$primary-color: #333;

2

3.button {

4 background-color: $primary-color;

5 &:hover {

6 background-color: lighten($primary-color, 10%);

7 }

8}

9```).

* + **Commenting**: Use comments to explain complex styles or sections of the stylesheet (e.g., /\* Navigation Styles \*/).
  + **Organize by Structure**: Group related styles together and maintain a logical order throughout the file.

### Chapter 24: Final Project

* **Application of All Concepts**: A comprehensive project to integrate and apply all learned CSS concepts.
  + **Project Planning**: Outline the project structure, components, and styles needed.
  + **Responsive Design**: Ensure the design adapts to various screen sizes using media queries and flexible layouts (e.g., flexbox and grid).
  + **Implementing Animations**: Use animations to enhance user experience (e.g., smooth transitions for buttons).
  + **Utilizing Functions**: Apply CSS functions to create dynamic layouts and responsive designs (e.g., using calc() for spacing).
  + **Organizing Styles**: Maintain a clean and organized stylesheet, following best practices for CSS organization.
  + **Testing and Iteration**: Test the project across different browsers and devices, making adjustments as necessary.