CSS

Adding CSS

1. Inline: <tag style="css"/>

```
<html style="background:blue"> </html>
```

2. Internal: <style>css</style>

```
<html>
   <head>
      <style>
         html {
            background:blue;
         }
      </style>
   </head>
</html>
```

3. External: <link href="styles.css"/>

```
<html>
   <head>
```

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


<! Inside styles.css ⇒ html { background: blue; }>
```

CSS Selectors

1. Types of selectors → Element selector, Class selector (.), Id selector (#), Attribute selector and Universal selector (*).

```
// Select paragraph tag with attribute draggable
p[draggable] {
    color: red;
}

// Select paragraph tag with attribute draggable set to false
p[draggable="false"] {
    color: red;
}
```

CSS Properties

- **CSS Colors**

1. https://colorhunt.co/ - Great website for different color palettes
2. MDN Docs: https://developer.mozilla.org/en-US/docs/Web/CSS/named-color
3. *background-color* and *color* properties.

- **Font Properties**

1. font-size

▼ 1 px (pixel) = 1/96 th of inch = 0.26 mm

   1 pt (point) = 1/72 th of inch = 0.35 mm

   1 em (pronounced as m) ⇒ 100% of parent

   1 rem ⇒ 100% of root

2. font-weight

▼ normal, bold - keywords

   lighter (-100), bolder (+100) - Relative to parent

   number - 100 to 900

3. font-family

▼ font-family: Helvetica, sans serif ⇒ Here Helvetica is typeform and sans serif is generic type.

   font-family: “Times New Roman”, serif

4. For many other fonts: https://fonts.google.com

5. text-align: center (left, right, start and end)

CSS Box Model

1. **Content**: The actual content of the element (e.g., text or image).
2. **Padding**: Space between the content and the border, Example: `padding: 10px;`
3. **Border**: The outer line surrounding the element, Example: `border: 2px solid black;`
4. **Margin**: The space outside the border, between elements, Example: `margin: 15px;`

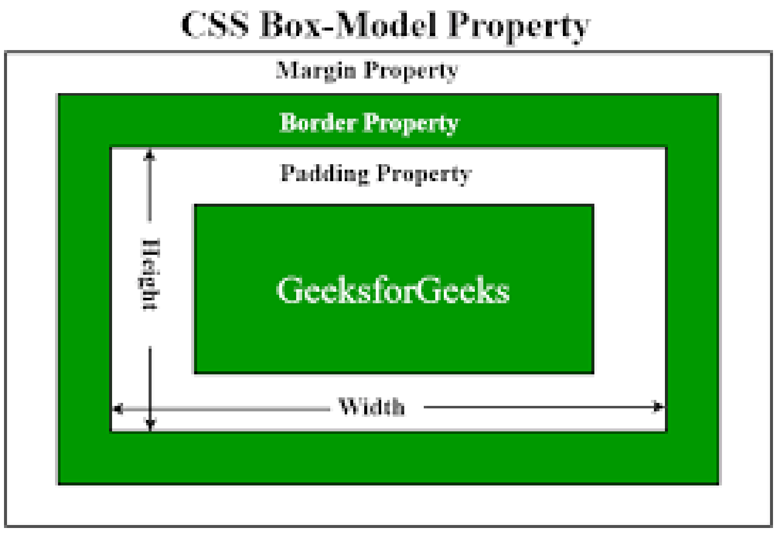
Example Code:

```
<div style="width: 200px; height: 100px;
        padding: 20px; border: 5px solid blue;
        margin: 10px;">Hello Box Model!</div>
```

This creates a box with:

- Content width: 200px
- Padding around content: 20px
- Border width: 5px
- Margin outside border: 10px

5. border-width: 2px 0px 3px 4px; (clockwise direction from top), Always goes after border property.
6. if border-width: 2px 4px; ⇒ (Here, top & bottom would be 2px and left & right would be 4px).
7. padding and margin works in same way as border-width.



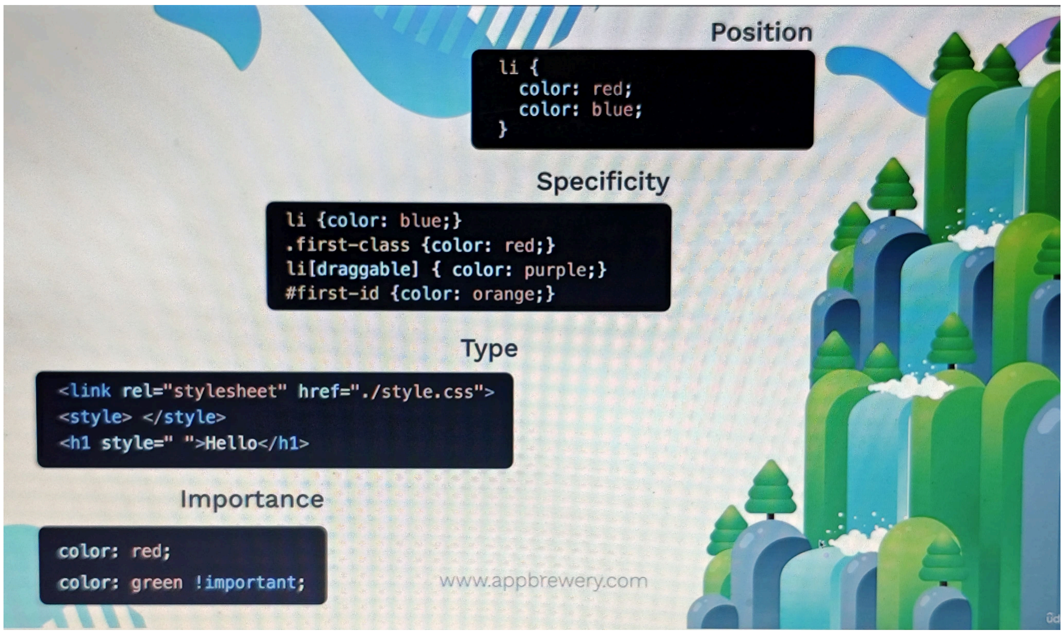
7. <div> ... </div> tag is used to separate the contents into different boxes.

The Cascade - Specificity and Inheritance

1. **Position**: Think of it like a set of rules in a game. If you have two rules, and the second one contradicts the first, the second one usually wins. In CSS, if two rules apply to the same element, **the one that comes later in your stylesheet will take precedence**.
2. **Specificity**: This is about how "specific" a rule is. For example, if you have an ID selector (like `#header`) and a class selector (like `.menu`), the ID selector will override the class selector because it's more specific. So, if an element

has both an ID and a class, the styles from the ID will be applied over the class styles. **(Order Of Importance: element < class < attribute < id)**

3. **Type**: There are different types of CSS styles—inline styles (added directly to an HTML element), internal styles (within a `<style>` tag in the HTML), and external styles (linked from a separate CSS file). Inline styles have the highest priority, so they will override internal and external styles. (**Order Of Importance: inline > internal > external**)
4. **Importance**: By adding `!important` to a CSS rule, you can give that rule the highest priority, making it override all other styles, regardless of their position or specificity. For instance, writing `color: red !important;` will ensure that red is the color applied, no matter what other styles are present.
5. Order of Priority: Importance > Type > Specificity > Position
6. Example



```
li {
  color: red;
  color: blue;
}
```

```
li {color: blue;}
.first-class {color: red;}
li[draggable] { color: purple;}
#first-id {color: orange;}
```

```
<link rel="stylesheet" href="./style.css">
<style> </style>
<h1 style=" ">Hello</h1>
```

```
color: red;
color: green !important;
```

Combining CSS Selectors

Taking below HTML code as example

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <title>Combining CSS Selectors</title>
  <link rel="stylesheet" href="./style.css">
</head>

<body>
  <h1>To Do List</h1>
  <h2>Monday</h2>
  <div class="box">
    <p class="done">Do these things today!</p>
    <ul class="list">
      <li>Wash Clothes</li>
      <li class="done">Read</li>
      <li class="done">Maths Questions</li>
    </ul>
  </div>

  <ul>
    <p class="done">Other items</p>
  </ul>
  <p>The best preparation for tomorrow is doing your best today.</p>

</body>

</html>
```

1. Grouping: elements separated by commas.

```
h1, h2 {
    color: blue;
}
```

2. Child: Apply to direct child of left side (Only to immediate childs).

```
selector > selector {
  color: blue;
}
```

```css
.box > p {
    color: firebrick;
}
```

3. Descendant: Apply to descendant of left side.

```css
selector selector {
/* Ancestor Descendant */
    color: blue;
}

.box li {
    color: blue;
}
```

4. Chaining: Apply where all selectors are true.

```css
selectorselectorselector {
    color: blue;
}

/* Always start with element, Remember to add . for class and # for id, Also there will be no space b/w selectors */
li.done {
    color: green;
}
```

5. Combining Combiners

```css
selector selectorselector {
    color: red;
}

ul p.done{
    font-size: 0.5rem;
}
```

```
ul > p.done {
  font-size: 0.5rem;
}
```

CSS Positioning

1. Static positioning: HTML default positioning.
2. Relative positioning: Position relative to default position.
3. Absolute positioning: Position relative to nearest positioned ancestor or top left corner of webpage. (Remember to keep to position as relative to any one of the ancestor or it will take the top left corner as reference)
4. **z-index**: To determine which one goes above which one. The one with higher z-index is displayed on top of another.
5. Fixed positioning: Position relative to top left corner of the browser window.
6. Example: https://appbrewery.github.io/css-positioning/

CSS Display

1. `block` elements start on a new line and take up the full width available; you can set their width and height.
2. `inline`elements do not start on a new line, flow within surrounding text, and only take up necessary width; you cannot set their width and height directly.
3. `inline-block` elements are a hybrid, flowing with text but allowing you to set their width and height.
4. The `none` value removes an element from the document flow, and it takes up no space.
5. Example: https://appbrewery.github.io/css-display/

CSS Float

1. Main purpose is to wrap text using float and clear.

2. **Property Name**: `float`
   - **Value**: `left` or `right`
3. **Functionality**:
   - When you apply `float: left;` to an element, like an image, it moves that element to the left side of its container. The text will then flow around the right side of the floated element.
   - Conversely, using `float: right;` moves the element to the right side, allowing text to wrap on the left.

Example CSS:

```css
img {
   float: left; /* Moves the image to the left */
}
```

4. **Behavior**: By default, an image is a block-level element and takes up the full width of its parent. When you float it, it's taken out of the normal flow, allowing text to adjust accordingly.
5. **Clearing Floats**: To stop text from wrapping around a floated element.

```css
footer {
   clear: both; /* Stops elements from flowing around the floated elements
*/
}
```

Creating Responsive Websites

It is possible in 4 ways: Media Queries, CSS Grid, CSS Flex and Bootstrap.

Media Queries

1. Adding breakpoints to create responsive websites.
2. Example

```css
@media (max-width: 600px) {  /*Breakpoint: Screen size less than or equal
to 600px, then the CSS inside this @media will be applied*/
```

```css
    h1 {
        font-size: 20px;
    }
}

@media (min-width: 600px) {  /*Breakpoint: Screen size more than or equal to 600px, then the CSS inside this @media will be applied*/
    h1 {
        font-size: 20px;
    }
}

@media (min-width: 600px) and (max-width: 900px) {
/*Breakpoint: Screen size more than 600px and less than 900px, then the CSS inside this @media will be applied*/
    h1 {
        font-size: 20px;
    }
}
```

3. Refer MDN docs: CSS Media Queries for more insights
4. Use **object-fit: cover;** property for images while developing responsive websites.

CSS Flexbox

1. display: flex and display: inline-flex
2. `display: flex`

- **Block-level** flex container.
- Makes the container behave like a **block element**, and its children become **flex items**.
- Takes **full width** by default.
- Used to **layout children in a row or column**.

3. `display: inline-flex`

- **Inline-level** flex container.
- Makes the container behave like an **inline element** while still acting as a **flex container** for its children.
- Container size is **only as wide as its content**.

4. The `flex-direction` property defines **the direction in which flex items are placed** in a flex container.

Possible Values:

Value	Layout Direction	Description
`row`	Left ➡️ Right *(default)*	Items are placed horizontally.
`row-reverse`	Right ➡️ Left	Items are placed in reverse horizontal order.
`column`	Top ⬇️ Bottom	Items are stacked vertically.
`column-reverse`	Bottom ⬆️ Top	Items are stacked in reverse vertical order.

5. `flex-basis` sets the **initial size** of a flex item **along the main axis**.

- When `flex-direction: row` → main axis is **horizontal**

  👉 `flex-basis: 100px` sets the **width** of each item to 100px.
- When `flex-direction: column` → main axis is **vertical**

  👉 `flex-basis: 100px` sets the **height** of each item to 100px.
- flex-basis always applies to the main axis — not width or height directly.
- This property will be applied to children and not the container itself.

6. Refer FLEX LAYOUT example website for more properties on flex.
7. Refer ***CSS GUIDE FOR FLEXBOX*** for all the available properties.
8. `order` (Child Property)

- Controls **the order** in which flex items appear.
- Default is `0`. Lower values appear **first**.
- Can be negative or positive.
- Used on individual flex **items**.

9. `flex-wrap` (Parent Property)

- Determines whether items should **wrap** onto multiple lines.

- Values:
    - `nowrap` (default): All items stay in **one line**.
    - `wrap`: Items **wrap to next line** if needed.
    - `wrap-reverse`: Items wrap in **reverse direction**.
- Applied to the flex **container**.

10. `justify-content` (Parent Property)

- Aligns items **horizontally along the main axis**.
- Common values:
    - `flex-start` (default)
    - `flex-end`
    - `center`
    - `space-between`
    - `space-around`
    - `space-evenly`
- Used on the **container**.

11. `align-items` (Parent Property)

- Aligns items **vertically along the cross axis** (per line).
- Common values:
    - `stretch` (default)
    - `flex-start`
    - `flex-end`
    - `center`
    - `baseline`
- Used on the **container**.

12. `align-self` (Child Property)

- Overrides `align-items` for a **specific item**.
- Same values as `align-items`.

- Useful for **individual alignment** within the group.
- Used on a **single item**.

13. `align-content` (Parent Property)

- Aligns **multiple lines** of items along the **cross axis** (only when wrapping).
- Only works when there are **multiple rows or columns**.
- Common values:
  - `flex-start`, `flex-end`, `center`
  - `space-between`, `space-around`, `space-evenly`
  - `stretch` (default)
- Used on the **container**.

14. Game to understand the above properties: https://appbrewery.github.io/flexboxfroggy/

15. Flex Sizing

- Priority list that flex-box uses internally to get the size of the each child item: Content width < Width < flex-basis < min-width/max-width

▼ `flex-grow` (Child Property)

- Default: `0` (item won't grow).
- Example: `flex-grow: 1` makes item expand if space is available.
- **Defines how much a flex item can grow** to fill available space.

▼ `flex-shrink` (Child Property)

- **Defines how much a flex item can shrink** when space is tight.
- Default: `1` (item can shrink).
- Example: `flex-shrink: 0` prevents item from shrinking.

Example:

```
/*Applied on child items → 0 indicates turned off and 1 indicates turned on
*/
flex-basis: 0;
flex-grow: 1; /*default 0*/
flex-shrink: 1; /*default 1*/
```

```
flex: 1 1 0; /*(grow shrink basis)*/

flex: 1;
```

16. `height: 100vh;` or can be set in percentages (vh - viewport height) makes the `container` stretch to fill the entire height of the browser window, which is crucial for:

- Ensuring that the container takes up the full screen.
- Enabling proper vertical centering of the content using flexbox.
- Providing control over the layout's vertical dimension.

CSS Grid

1. Flexbox is more useful for 1D layouts and grid for 2D layouts
2. https://appbrewery.github.io/grid-vs-flexbox/
3. Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr; /* How much fraction of whole space should be occupied by each column */
  grid-template-rows: 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr; /* How much fraction of whole space should be occupied by each row */
}
```

4. Grid Sizing → https://appbrewery.github.io/grid-sizing/

```
.container {
  display: grid;
  grid-template-columns: 100px auto; /*In the case of columns, auto takes up the entire width of the window*/
  grid-template-rows: 200px auto; /*In the case of rows, auto takes up the required height to fit the content inside it*/
}
```

```
.container {
  display: grid;
  grid-template-columns: 200px minmax(400px, 800px); /*min and max wi
dth*/
}
```

```
.container {
  display: grid;
  grid-template-columns: repeat(2, 100px); /*repeat 2 times with width 100p
x*/
  grid-template-rows: repeat(3, 300px); /*repeat 3 times with height 300px
*/
}
```

```
.container {
  display: grid;
  grid-template-columns: 200px, 100px;
  grid-template-rows: 100px, 200px;
  grid-auto-rows: 300px; /*This is usedful when there are more rows than t
he specified rows in a container. All the extra added rows will take 300px h
eight. This will be the same behaviour for columns as well.*/
  grid-auto-columns: 100px;
}

/* In case if we dont specify the auto functions, they try to take up the exist
ing rows and columns properties */
```

5. Grid items placement

```
/* These numbers indicate the line numbers in a grid*/ /*-1 corresponds to l
ast row or column*/
.item {
    grid-column: span 2;
    /* above property is a shorthand property for below two properties */
    grid-column-start: 1;
    grid-column-end: -1;
}

.item {
   grid-row: span 2;
   /* above property is a shorthand property for below two properties */
   grid-row-start: 1;
   grid-row-end: 2;
}

.item {
   order: 1; /*By default all the items will have order 0*/
   /* Lower values appear first, higher values later. */
}

.item {
   grid-column-start: 1;
   grid-column-end: 3;
   grid-row-start: 2;
   grid-row-end: 3;
   grid-area: 2 / 1 / 3 / 3; /* row start / column start / row end / column end
*/
}
```

CSS Bootstrap framework

1. https://getbootstrap.com/
2. Bootstrap follows 12 columns system

```
<div class="container">
   <div class="row">
      <div class="col-2">Hello</div>
      <div class="col-4">World</div>
      <div class="col-6">Bootstrap</div>
   </div>
</div>
```

```
<div class="col-sm-3">TEST</div>
<!-until small screens take 3 spaces out of 12 spaces>

<div class="col-sm-12 col-md-8 col-l-4">TEST</div>
<!-small screens take 12 spaces, medium screens 8 and large screens 4 ou
t of 12 spaces>
```

3. Available break points: https://getbootstrap.com/docs/5.3/layout/breakpoints/#available-breakpoints
4. Practice Bootstrap layout: https://appbrewery.github.io/bootstrap-layout/
5. Using margin and padding using notation: https://getbootstrap.com/docs/5.3/utilities/spacing/#margin-and-padding

Some points to remember

1. Colors: https://color.adobe.com/create/color-wheel & https://colorhunt.co/ (Use this websites to get the color pallettes for websites).
2. For UI Inspirations: https://collectui.com/.
3. Follow consistency b/w different parts of website.
4. Use fonts that please the users and target audience.
5. Use free tools like https://collectui.com/ to design websites.