2024/2025

CSS IN FULL STACK

Presented to

Rajesh Sir

TOPS Technologies

HTML in Python

Que.1
A CSS selector is a pattern used to select and style HTML elements. It tells the browser which HTML elements you want to apply styles to.

Types of CSS Selectors with Examples:

- **Element Selector:** Selects all HTML elements of a specific type.
- **Class Selector**: Selects elements with a specific class attribute. Prefix with a . (dot).
- **ID Selector**: Selects an element with a specific ID. Prefix with # (hash).

Que.2

CSS specificity is the set of rules that determines which CSS rule is applied when multiple rules target the same HTML element.
When conflicting styles apply to the same element, the browser uses specificity to decide which style wins.

📊 Specificity Hierarchy (from lowest to highest):

Selector Type	Example	Specificity Value
Universal selector	`*`	0,0,0,0
Element selector	`p`, `div`	0,0,0,1
Class selector	`.class`	0,0,1,0
Attribute selector	`[type="text"]`	0,0,1,0
Pseudo-class	`:hover`	0,0,1,0
ID selector	`#id`	0,1,0,0
Inline styles	`style=""`	1,0,0,0

```
p {
  color: blue;          /* Specificity: 0,0,0,1 */
}

.text {
  color: green;         /* Specificity: 0,0,1,0 */
}

#para1 {
  color: red;           /* Specificity: 0,1,0,0 */
}
```

<html>

- The root element that wraps the entire HTML document.
- All other elements are nested inside this.

<head>

- Contains meta-information about the document (not visible to users).
- Includes <title>, styles, scripts, and meta tags.

<title>

- Sets the title of the webpage (displayed in browser tab).
- It is placed inside the <head> section.

<body>

- Contains all the visible content of the webpage (text, images, links, etc.).

Que.3

1. Inline CSS:

- CSS is written inside the HTML element using the style attribute.
- This is red text.

- **Advantages**:
  - Quick and easy for single element styling.
  - Useful for testing or debugging.

- **Disadvantages**:
  - Not reusable — must be repeated for every element.
  - Makes HTML messy and hard to maintain.
  - Cannot use pseudo-classes (:hover, :focus) effectively.

2. Internal CSS:

- CSS is written inside a <style> tag in the <head> section of the HTML document.

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

- **Advantages**:
  - Styles are in one place within the same file.
  - Useful for small projects or single-page websites.
- **Disadvantages**:
  - Not reusable across multiple pages.
  - Increases page load time if used excessively.

3. External CSS:

- CSS is written in a separate .css file and linked using the <link> tag.

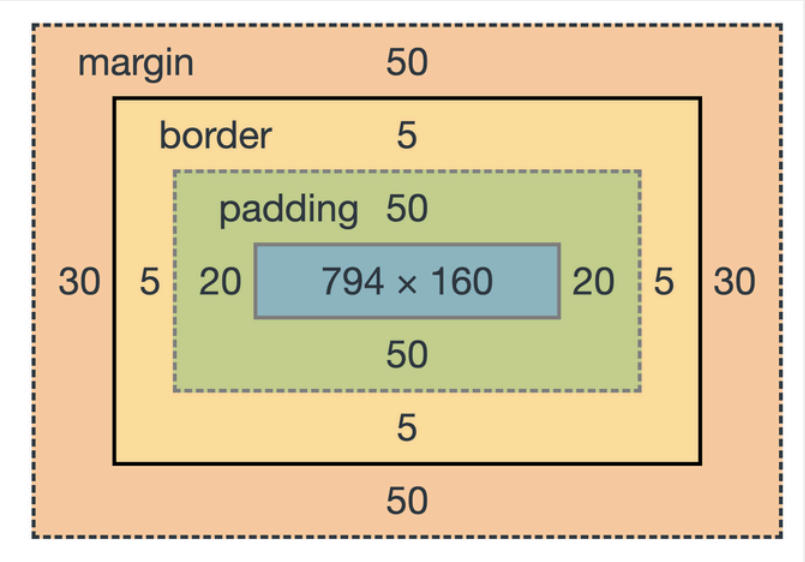
```
<head>
  <link rel="stylesheet" href="styles.css">
</head>

p {
  color: green;
}
```

- **Advantages**:
  - Reusability: Same CSS file can be used across multiple pages.
  - Keeps HTML clean and organized.
  - Browser can cache CSS file, improving load time.

- **Disadvantages:**
  - Extra HTTP request to fetch CSS file.
  - Requires proper file linking; if link breaks, styles won't apply.

Que.4

The CSS Box Model describes how every HTML element is a rectangle box made up of four parts



- **Components of the Box Model**
  - **Content**
    - The actual content (text, image, etc.).
    - Size is set by properties like width and height.
  - **Padding**
    - Space inside the element, between content and border.

- **Border**
  - A line surrounding the padding and content.
  - Adds to the element’s size.
  - Example: border: 2px solid black;
- **Margin**
  - Space outside the border, separating the element from others.
  - Does not increase the box size — it pushes other elements away.
  - Example: margin: 20px;

Example :

```
div {
  width: 200px;
  padding: 10px;
  border: 5px solid black;
  margin: 20px;
}
```

- Content width: 200px
- Padding: 10px × 2 = 20px
- Border: 5px × 2 = 10px
- Margin: 20px × 2 = 40px
- Total space taken:
- Width = 200 + 20 + 10 = 230px
- Height behaves similarly

Que.5

1. content-box (Default):
Only the content area is counted in the width and height.
Padding and border are added outside the specified width/height.

Example:
box-sizing: content-box;
width: 200px;
padding: 10px;
border: 5px solid;

Total width = 200 + 10×2 (padding) + 5×2 (border) = 230px

2. border-box:
The width and height include content, padding, and border.
The content area shrinks to make room for padding and border.

Example:
box-sizing: border-box;
width: 200px;
padding: 10px;
border: 5px solid;

Que.6
Flexbox (Flexible Box Layout) is a one-dimensional layout model in CSS that makes it easier to design flexible, responsive layouts. It is especially useful for aligning elements horizontally or vertically, distributing space, and handling dynamic or unknown sizes.

1. Flex Container
The parent element that holds flex items. You turn an element into a flex container by applying:
display: flex;

Example:

```
<div class="container">
  <div class="box">A</div>
  <div class="box">B</div>
</div>
```

css:

```
.container {
  display: flex; /* makes it a flex container */
}
```

2. Flex Item
These are the direct children of a flex container. Flex items automatically become flexible and can be aligned, spaced, or ordered using Flexbox properties.

Property	Descriptiom
Flex - direction	Sets direction
justify - content	Align items horizontally
align items	Algin items vertically
flex wrap	Allows item to wrap onto multiple lines

flex	Sets grow/shrink behavior and size basis
order	Controls the order of the item
align-self	Overrides vertical alignment per item

```
.container {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

Que.7

Grid
CSS Grid is a powerful 2-dimensional layout system in CSS that allows you to layout items in rows and columns. It gives you full control over both axes (horizontal and vertical) simultaneously.

Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr;
  grid-template-rows: auto auto;
  gap: 10px;
}
```

Flexbox
Flexbox (Flexible Box Layout) is a 1-dimensional layout system. You can control layout in one direction at a time — either as a row or as a column.

Example:

```
.container {
  display: flex;
  flex-direction: row; /* or column */
  gap: 10px;
}
```

Features	CSS Grid	Flexbox
Layout Direction	Two-dimensional (rows and columns)	One-dimensional (row or column)
Alignment Control	Full control on both axes	Better for aligning items in one axis
Content vs Layout	Good for designing overall page layout	Ideal for distributing space in a line
Item Placement	Precise control using grid lines and areas	Items placed based on content order
Use Cases	Whole page or section layout	Nav bars, cards in a row, buttons, etc.

Que.8

1.grid-template-columns:

This property defines the number and size of columns in a grid layout.

Syntax:
grid-template-columns: <track-size> <track-size> ...;

Example:
```
.container {
  display: grid;
  grid-template-columns: 100px 200px auto;
}
```

This creates 3 columns:

- 1st column: 100px wide
- 2nd column: 200px wide
- 3rd column: auto (fills remaining space)

2.grid-template-rows:

This property defines the number and size of rows in the grid.

Syntax:
grid-template-rows: <track-size> <track-size> ...;

Example:

```
.container {
  display: grid;
  grid-template-rows: 100px auto 50px;
}
```

This creates 3 rows:

- 1st row: 100px tall
- 2nd row: auto (based on content)
- 3rd row: 50px tall

3. grid-gap (Now split as row-gap and column-gap in modern CSS)

This property defines the space between grid rows and columns.

Syntax:

```
grid-gap: <row-gap> <column-gap>;
```

Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: auto auto;
  grid-gap: 20px 10px;
}
```

- 20px gap between rows
- 10px gap between columns

Que.9

A media query is a CSS technique used to condition styles based on the user's device or screen size.

Syntax:

```
@media media-type and (condition) {
  /* CSS rules here */
}
```

Example:

```
/* Apply styles when screen width is 600px or less */
@media screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

Responsive design aims to make web pages look good on all devices — desktops, tablets, and phones. Media queries help with:

- Adjusting layout for different screen sizes (e.g., single column on phones, multi-column on desktop).
- Improving readability by resizing text or hiding/showing elements.
- Enhancing user experience without needing separate mobile/desktop websites.

Que.10

```
@media screen and (max-width: 600px) {
  body {
    font-size: 14px;
  }
}
```

Explanation:

- @media screen: Targets screen devices (like phones, tablets, desktops).
- (max-width: 600px): Applies the styles only when the screen width is 600px or less.
- Inside the block, we set body { font-size: 14px; }, which reduces the font size for smaller screens to improve readability and layout.

Que.11

Web-Safe Fonts:

- Definition: Web-safe fonts are a set of standard fonts that are pre-installed on most operating systems and devices.
- Examples: Arial, Times New Roman, Verdana, Georgia, Courier New, Trebuchet MS.
- Usage: Since they are already installed on user devices, the browser doesn’t need to download them.
- Performance: Fast loading; no additional HTTP requests.

Custom Web Fonts:

- Definition: Fonts that are not commonly installed on devices and are loaded from external sources (like Google Fonts, Adobe Fonts, or custom files).
- Examples: Roboto, Open Sans, Lato, Montserrat (when served via web).
- Usage: They are linked via @font-face in CSS or external font libraries.
- Performance: Slower to load because they require downloading; may cause a "flash of unstyled text" (FOUT) or layout shift.

Que.12

The font-family property in CSS specifies the typeface (font) to be used for an element's text.

- You can list multiple fonts as a fallback system, in case the browser can't load the first one.
- Fonts can be generic (like serif, sans-serif) or specific (like Arial, Roboto).

Syntax:

```
selector {
  font-family: "Font Name", fallback-font, generic-family;
}
```

css:

```
body {
 font-family: "Arial", "Helvetica", sans-serif;
}
```