CSS THEORY

CSS SELECTORS & STYLING

Question 1: What is a CSS selector? Provide examples of element, class, and ID selectors.

Answer:

A **CSS selector** is used to target (select) HTML elements that you want to style.
It tells the browser *which element(s)* the CSS rules should apply to.

- **Types of Selectors:**

1. **Element Selector**
    - It selects HTML elements by their tag name.
    - Example:
    - p {
    - color: blue;
    - font-size: 18px;
    - }

This will style **all (paragraph)** elements.

2. **Class Selector**
    - It selects elements that have a specific **class** attribute.
    - A class name always starts with a **dot (.)** in CSS.
    - Example:
    - .highlight {
    - background-color: yellow;
    - font-weight: bold;
    - }

Apply in HTML as:

This text is highlighted.

3. **ID Selector**

- It selects an element with a unique **id**.
- An id name always starts with a **hash (#)** in CSS.
- Example:
- #main-title {
- color: red;
- text-align: center;
- }

Apply in HTML as:

<h1 id="main-title">Welcome to My Website</h1>

Question 2: Explain the concept of CSS specificity. How do conflicts between multiple styles get resolved?

Answer:

CSS specificity is a rule that determines **which CSS style is applied** when multiple rules target the same element.

When two or more CSS rules apply to the same element, **the rule with higher specificity wins**.

- **Specificity Order (Highest to Lowest):**
  1. **Inline CSS** → highest priority
  2. **ID Selector** → next priority
  3. **Class, attribute, and pseudo-class selectors**
  4. **Element (tag) selectors** → lowest priority
  5. **Universal selector (*)** → very low priority
- **Example:**

<h1 id="title" class="heading">Welcome!</h1>

h1 { color: blue; } /* Element selector */

.heading { color: green; } /* Class selector */

#title { color: red; } /* ID selector */

Result: The text will be **red**, because ID has the **highest specificity**.

Question 3: What is the difference between internal, external, and inline CSS? Discuss the advantages and disadvantages of each approach.

Answer:

Type	Definition	Example	Advantages	Disadvantages
Inline CSS	CSS written inside an HTML tag using the style attribute.	Hello	Quick and easy for single elements No external files needed	Not reusable Difficult to maintain Mixes HTML and CSS
Internal CSS	CSS written inside the <style> tag in the <head> section of an HTML document.	html <style> p {color: blue;} </style>	Easy to style one page Keeps CSS separate from HTML content	Not reusable for multiple pages
External CSS	CSS written in a separate .css file and linked using <link>.	html <link rel="stylesheet" href="style.css">	Best for large websites Reusable across multiple pages Easy to maintain	Needs internet or file link One extra file to load

---

CSS Box Model

Question 1: Explain the CSS box model and its components (content, padding, border, margin). How does each affect the size of an element?

Answer:

The CSS Box Model is the basic layout structure of every HTML element.
It describes how the space around an element is calculated — including its content, padding, border, and margin.

Each HTML element on a webpage is considered as a rectangular box made up of these four parts

---

1. Content

- **The main area where text, images, or other elements appear.**
- **You can set its size using width and height.**

Example:

```
div {
  width: 200px;
  height: 100px;
```

}

---

2. Padding

- The space between the content and the border.
- It creates inner spacing inside the box.
- Increases the total element size (unless box-sizing: border-box is used).

Example:

```
div {
 padding: 20px;
}
```

Total width = content width + left padding + right padding

---

3. Border

- The line around the padding and content.
- Adds thickness to the box and increases its overall size.

Example:

```
div {
 border: 5px solid black;
}
```

---

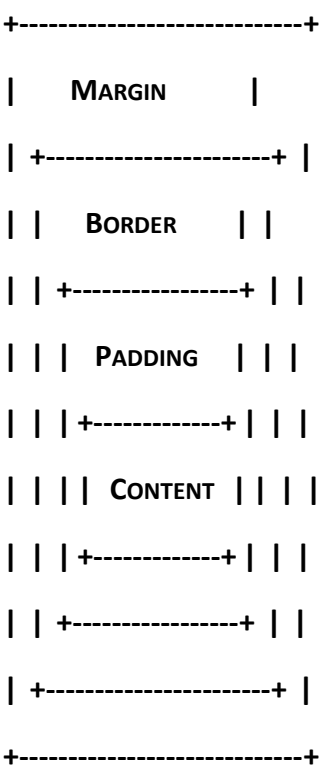
4. Margin

- The space outside the border.
- It creates distance between the element and surrounding elements.
- Margins do not add to the element's actual box size — they create external space.

Example:

```
div {
 margin: 15px;
}
```

Box Model Diagram (Structure)



Effect on Total Element Size

When using the default content-box model:

Total Width = width + padding-left + padding-right + border-left + border-right

Total Height = height + padding-top + padding-bottom + border-top + border-bottom

---

Example:

```
div {
  width: 200px;
  padding: 10px;
  border: 5px solid;
  margin: 15px;
}
```

Total Width = 200 + 10 + 10 + 5 + 5 = 230px

Question 2: What is the difference between border-box and content-box box-sizing in CSS? Which is the default?

Answer:

The box-sizing property controls how the total size of an element is calculated (whether padding and borders are included in width/height or not).

- **1. content-box (Default)**
  - The width and height apply only to the content area.
  - Padding and border are added outside the content, increasing the total element size.

Example:

```
div {
 box-sizing: content-box;
 width: 200px;
 padding: 10px;
 border: 5px solid;
}
```

Total Width = 200 + 10 + 10 + 5 + 5 = 230px

- **2. border-box**
  - The width and height include content, padding, and border.
  - The total size stays fixed, which makes layout easier and consistent.

Example:

```
div {
 box-sizing: border-box;
 width: 200px;
 padding: 10px;
 border: 5px solid;
}
```

Total Width = 200px (padding + border included inside)

Comparison Table

Property	Includes Padding & Border in Width/Height?	Total Size Changes?	Default?
content-box	No	Yes	Yes
border-box	Yes	No	No

CSS Flexbox

Question 1:

What is CSS Flexbox, and how is it useful for layout design? Explain the terms flex-container and flex-item.

Answer:

- **What is CSS Flexbox?**

Flexbox (Flexible Box Layout) is a modern CSS layout system that helps to design responsive and flexible layouts easily.

It allows elements to align, distribute space, and adjust automatically on different screen sizes 📱💻.

Why Flexbox is Useful:

Makes layouts responsive
Easy vertical & horizontal alignment
Simplifies spacing and ordering
Replaces float & position methods
Great for modern web design

Key Terms:

- **1. Flex Container**

It is the parent element that holds all the flex items.
We make a container flexible using:

display: flex;

- **2. Flex Items**

These are the child elements inside the flex container that are arranged according to Flexbox rules.

Example:

```
<style>
.container {
  display: flex;
  background: lightgray;
  justify-content: center;
}
.item {
  background: skyblue;
  padding: 20px;
  margin: 10px;
}
</style>

<div class="container">
  <div class="item">Box 1</div>
  <div class="item">Box 2</div>
  <div class="item">Box 3</div>
</div>
```

Boxes are aligned in one row and centered automatically.

Question 2:

Describe the properties justify-content, align-items, and flex-direction used in Flexbox.

Answer:

Flexbox uses special properties to align and position items inside a container.

- **1. flex-direction**

Defines the direction of the flex items.

```
flex-direction: row | row-reverse | column | column-reverse;
```

Value	Description
row	Items placed left → right *(default)*
row-reverse	Items placed right → left
column	Items placed top → bottom
column-reverse	Items placed bottom → top

- **2.** justify-content

Controls horizontal alignment (main axis).

justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;

Value	description
flex-start	Align left
flex-end	Align right
center	Center horizontally
space-between	Equal space between items
space-around	Equal space around items
space-evenly	Equal spacing on both sides

- **3.** align-items

Controls vertical alignment (cross axis).

align-items: flex-start | flex-end | center | baseline | stretch;

Value	Description
flex-start	Align top
flex-end	Align bottom
center	Center vertically
stretch	Stretch items to container height

Value	Description
baseline	Align based on text baseline

Practical Example:

```
<style>
.container {
  display: flex;
  flex-direction: row;
  justify-content: center;
  align-items: center;
  height: 200px;
  background-color: lightgray;
}
.item {
  background: tomato;
  color: white;
  padding: 20px;
  margin: 10px;
}
</style>

<div class="container">
  <div class="item">Box 1</div>
  <div class="item">Box 2</div>
  <div class="item">Box 3</div>
</div>
```

The boxes are aligned in a row, centered both horizontally and vertically.

CSS GRID

Question 1:

Explain CSS Grid and how it differs from Flexbox. When would you use Grid over Flexbox?

Answer:

- **What is CSS Grid?**

CSS Grid is a powerful **2D layout system** in CSS used to design web page layouts in **rows and columns**.
It gives full control over the layout structure — both **horizontal (rows)** and **vertical (columns)** alignment.

Think of it like an invisible **graph paper**, where you can place elements in different grid cells.

How CSS Grid Works:

1. The **parent element** becomes a **grid container** using display: grid;.
2. The **child elements** become **grid items** that can be placed inside rows and columns using grid properties.

Difference Between CSS Grid and Flexbox:

Feature	CSS Grid	Flexbox
Layout Type	**2D layout** (Rows + Columns)	**1D layout** (Row *or* Column)
Main Use	For **overall page structure**	For **arranging items in a single line**
Axis	Works on **both axes** (horizontal & vertical)	Works on **one axis** at a time
Item Placement	Items can be placed in specific grid cells	Items flow in one direction only
Alignment	Perfect for **complex layouts**	Perfect for **simple alignment**

When to Use Grid Over Flexbox:

Use **Grid** when you need:

- Multi-row and multi-column layouts (e.g., web page sections)
- Fixed and flexible track sizes together
- Complex or 2D responsive design

Use **Flexbox** when you need:

- Simple one-direction alignment (like a navbar or button group)
- Content-based layout instead of structure-based

Example:

```
<style>
.container {
  display: grid;
  grid-template-columns: 200px 200px 200px;
  grid-template-rows: 100px 100px;
  gap: 10px;
}
.item {
  background: skyblue;
  padding: 20px;
  text-align: center;
}
</style>

<div class="container">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>
```

This creates a grid with **3 columns × 2 rows**, and gaps between boxes.

Question 2:

Describe the grid-template-columns, grid-template-rows, and grid-gap properties. Provide examples of how to use them.

Answer:

- **1. grid-template-columns**

Defines the **number and width of columns** in a grid container.
You can use pixels (px), percentage (%), or the flexible fr unit (fractional space).

Syntax:

grid-template-columns: 100px 200px 100px;

This creates 3 columns — 100px, 200px, and 100px wide.

Example:

grid-template-columns: 1fr 2fr 1fr;

The middle column takes **twice as much space** as the others.

- **2. grid-template-rows**

Defines the **number and height of rows** in the grid.

Syntax:

grid-template-rows: 100px 150px;

This creates two rows — one 100px tall and one 150px tall.

- **3. grid-gap (or gap)**

Specifies the **space between rows and columns** in the grid.

Syntax:

grid-gap: 20px;

Adds a 20px gap between all rows and columns.

Or separately:

row-gap: 10px;

column-gap: 20px;

Example:

```
<style>
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;  /* 3 equal columns */
  grid-template-rows: 100px 100px;     /* 2 rows */
  gap: 15px;                           /* space between boxes */
}
.item {
  background-color: lightcoral;
  color: white;
  text-align: center;
  padding: 20px;
  font-size: 20px;
}
</style>

<div class="container">
  <div class="item">Box 1</div>
  <div class="item">Box 2</div>
  <div class="item">Box 3</div>
  <div class="item">Box 4</div>
  <div class="item">Box 5</div>
  <div class="item">Box 6</div>
</div>
```

Creates a 3×2 grid layout with even spacing and equal column widths.

.

Responsive Web Design with Media Queries

Question 1:

What are media queries in CSS, and why are they important for responsive design?

Answer:
Media queries in CSS are rules that apply different styles depending on the device's characteristics such as **screen width, height, resolution, or orientation**.
They help developers create **responsive web designs** — layouts that automatically adjust and look good on all devices like **mobiles, tablets, and desktops**.

Importance of Media Queries:

1. **Mobile-friendly websites** – Ensure content fits smaller screens properly.
2. **Better user experience** – Adapts design according to screen size.
3. **No horizontal scrolling** – Keeps layout clean and readable.
4. **SEO benefits** – Google prefers responsive sites.

Example use case:
When you want a website's layout or font to change automatically on smaller screens.

Question 2:

Write a basic media query that adjusts the font size of a webpage for screens smaller than 600px.

Answer (with Example):

```
/* Default style */
body {
  font-size: 18px;
}

/* Media query for screens smaller than 600px */
@media screen and (max-width: 600px) {
  body {
    font-size: 14px; /* Smaller font for mobile screens */
  }
}
```

Explanation:

- @media screen and (max-width: 600px) means — apply these styles only when the screen width is **600 pixels or less**.
- Inside this block, the font-size changes to **14px**, making text easier to read on small screens.

Typography and Web Fonts

Question 1:

Explain the difference between web-safe fonts and custom web fonts. Why might you use a web-safe font over a custom font?

Answer:
Web-safe fonts are the **standard fonts** that are already installed on most operating systems (Windows, macOS, Android, etc.).
They ensure that text looks the same across different devices and browsers **without needing an internet connection** to load extra fonts.

Examples of Web-safe fonts:

- Arial
- Times New Roman
- Georgia
- Verdana
- Courier New

Custom web fonts, on the other hand, are **downloaded from the web** (like Google Fonts or Adobe Fonts).
They allow designers to use more stylish, unique fonts that may not be pre-installed on a user's system.

Examples of Custom Web Fonts:

- Poppins
- Roboto
- Open Sans
- Lato
- Montserrat

Why use a web-safe font?
Because web-safe fonts:

1. Load **faster** (no external download).

2. Improve **performance and accessibility**.
3. Are useful when **internet connection is slow or unavailable**.
4. Provide **consistent rendering** across all platforms.

Question 2:

What is the font-family property in CSS? How do you apply a custom Google Font to a webpage?

Answer:

The **font-family** property in CSS specifies the **typeface** to be used for text on a webpage.
It can include one or more font names, and the browser will use the first available one.

Syntax:

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

Example:

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

Here, the browser will try to use **Arial** first; if it's not available, it will use a **sans-serif** font.

How to Apply a Custom Google Font:

Step 1: Go to Google Fonts and choose a font (e.g., *Poppins*).
Step 2: Copy the <link> tag provided by Google and paste it inside the <head> section of your HTML file.

```
<link href="https://fonts.googleapis.com/css2?family=Poppins&display=swap" rel="stylesheet">
```

Step 3: Use the font in your CSS with font-family property:

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

Explanation:

- The Google Fonts link **imports the font** from the internet.
- The font-family property **applies it** to the entire webpage or selected elements.