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

Ans.

A **CSS selector** is a pattern used to select and style specific elements on a web page. It tells the browser which HTML elements the associated CSS rules should apply to.

Common Types of CSS Selectors:

1. Element Selector

- Targets HTML elements by their tag name.
- **Example:**

```
p {
  color: blue;
}
```

2. Class Selector

- Targets elements with a specific `class` attribute.
- Denoted by a period (`.`) followed by the class name.
- **Example:**

```
.highlight {
  background-color: yellow;
}
```

3. ID Selector

- Targets a single, unique element with a specific `id` attribute.
- Denoted by a hash (`#`) followed by the ID name.
- **Example:**

```
#header {
  font-size: 24px;
}
```

(2) Explain the concept of CSS specificity. How do conflicts between multiple stylesget resolved?

Ans.

CSS Specificity — How Conflicts Between Styles Are Resolved

CSS specificity is a set of rules the browser uses to determine **which styles take precedence** when multiple rules target the same element.

What is Specificity?

Specificity is a score (or weight) assigned to a CSS rule based on the type of selectors used. The **more specific** the rule, the **higher priority** it has.

Specificity Hierarchy (From Lowest to Highest)

Selector Type	Specificity Value
Universal selector (`*`)	0,0,0,0
Element selector (`div`, `p`)	0,0,0,1
Class selector (`.class`)	0,0,1,0

Attribute selector (`[type="text"]`)	0,0,1,0
Pseudo-classes (`:hover`, `:focus`)	0,0,1,0
ID selector (`#id`)	0,1,0,0
Inline style (`style=""`)	1,0,0,0
`!important` declaration	Overrides everything (but should be avoided unless necessary)

How Specificity is Calculated

Specificity is usually calculated as a 4-part value: **(a, b, c, d)**

- **a** = Inline styles
- **b** = Number of ID selectors
- **c** = Number of class selectors, attributes, and pseudo-classes
- **d** = Number of element and pseudo-element selectors

> The rule with the **higher value in the leftmost part** wins.

Example:

```
p {
  color: blue;
}
```

```
.text {
  color: green;
}

#main {
  color: red;
}

<p id="main" class="text">Hello!</p>
```

What About `!important`?

The `!important` declaration overrides all other rules **except** another `!important` rule with **higher specificity**.

```
p {
  color: blue !important;
}

#main {
  color: red;
}
```

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

Ans.

Differences Between Internal, External, and Inline CSS

CSS can be applied to HTML documents in **three main ways**. Each method has its own **use cases**, **advantages**, and **disadvantages**.

1. Inline CSS

- **Definition**: CSS written directly in the HTML tag using the `style` attribute.
- **Example**:

  This is inline styled text.

- **Advantages**:
  - Quick and easy for testing or one-off styles.
  - Overrides other styles due to higher specificity.
- **Disadvantages**:
  - Hard to maintain and scale.
  - Mixes content with presentation (violates separation of concerns).
  - Redundant if used repeatedly (no reusability).

2. Internal CSS

- **Definition**: CSS written within a `<style>` tag in the `<head>` section of an HTML document.
- **Example**:

  <head>

<style>

p {

color: blue;

```
  }
 </style>
</head>
```

- **Advantages**:
    - Good for single-page styling.
    - Keeps all styles in one place (for that file).
- **Disadvantages**:
    - Styles are not reusable across multiple pages.
    - Can increase page load time if repeated in multiple files.
    - Still mixes structure and style in the same file.

3. External CSS

- **Definition**: CSS stored in a separate `.css` file and linked using the `<link>` tag.
- **Example**:

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

```
p {
 color: green;
}
```

- **Advantages**:
    - Best for large websites (reusable across multiple pages).
    - Cleaner HTML structure.

    - Easier maintenance and collaboration.

- **Disadvantages**:
    - Requires an extra HTTP request (may affect loading if not optimized).
    - Won't work without internet if the file is hosted externally (e.g., on a CDN).

CSS Box Model

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

Box Model Components (Inside → Out):

1. **Content**
    - The actual text, image, or other content inside the element.
    - Width and height set directly with `width` and `height`.
2. **Padding**
    - Space **between the content and the border**.
    - Expands the size of the box **inside** the border.

Example:
```
padding: 10px;
```

3. **Border**
    - Surrounds the padding (and content).
    - Has thickness, color, and style.

Example:

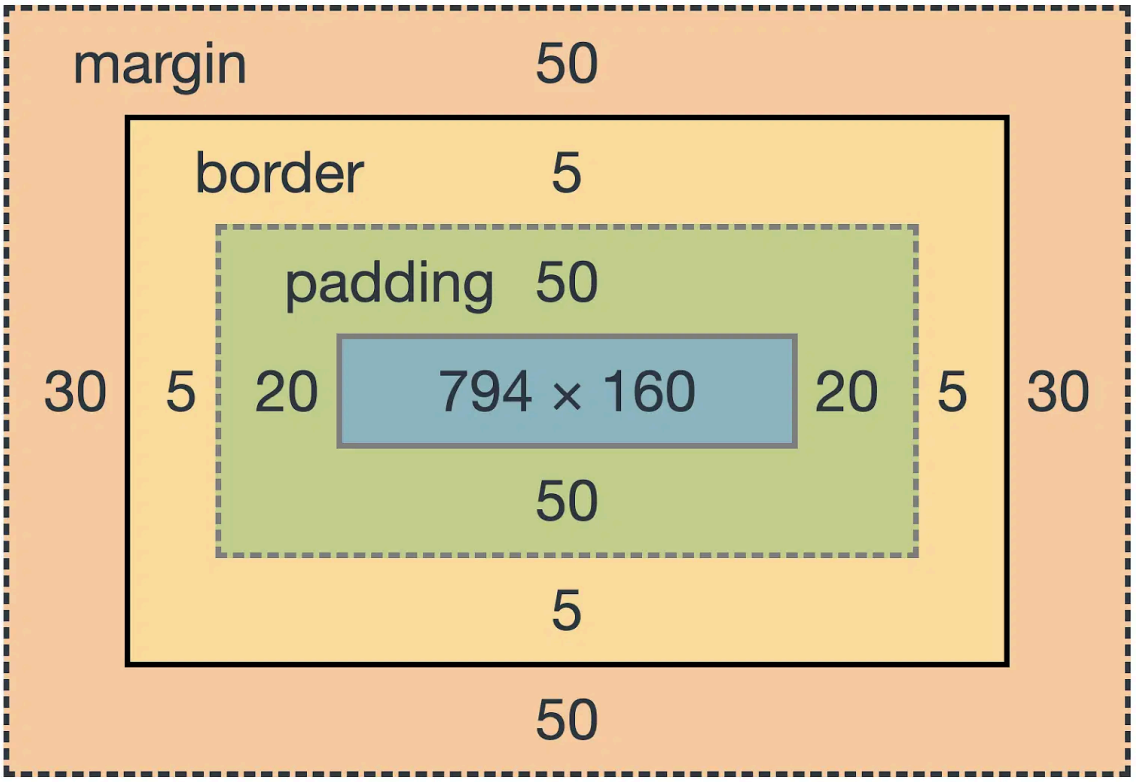
```
border: 2px solid black;
```

4. **Margin**
    - Space **outside the border**.
    - Creates space between this element and others.

Example:

margin: 15px;

Visual Overview:



How It Affects Element Size:

By default (in `content-box` model):

```
Total Width  = content width + left/right padding + left/right
border + left/right margin
```

```
Total Height = content height + top/bottom padding + top/bottom
border + top/bottom margin
```

So **adding padding, border, or margin increases the total space** the element occupies on the page.

Box-Sizing Property

You can control how size is calculated using `box-sizing`:

1. Default:

```
box-sizing: content-box;
```

- Padding and border are **added** to the width and height.

2. Better for layout:

```
box-sizing: border-box;
```

- Padding and border are **included** in the width/height, making layouts more predictable.

(5) What is the difference between border-boxand content-boxbox-sizing inCSS? Which is the default?

Ans.

Difference Between `border-box` and `content-box` in CSS `box-sizing`

The `box-sizing` property in CSS defines **how the total width and height of an element are calculated** — whether the `padding` and `border` are included **inside** or **outside** the specified width/height.

`box-sizing: content-box` (Default Value)

- **How it works**:
    - `width` and `height` only apply to the **content**.
    - Padding and border are added **outside** that size.

Example:

```
div {
  width: 200px;
  padding: 20px;
  border: 10px solid black;
  box-sizing: content-box;
}
```

- **Total width** = 200 (content) + 40 (padding) + 20 (border) = **260px**
  Total height = height + padding + border

box-sizing: border-box

- **How it works**:
    - `width` and `height` include **content + padding + border**.
    - The actual visible box size stays as you set it.

Example:

```
div {
  width: 200px;
  padding: 20px;
  border: 10px solid black;
  box-sizing: border-box;
}
```

- **Total width** stays at **200px**
  The browser automatically **shrinks the content area** to fit padding and border inside the 200px.

Comparison Table

Feature	`content-box`	`border-box`
Includes padding/border in width?	No	Yes
Predictable layout?	Less predictable	More predictable
Default in CSS?	Yes (default)	No (must be set manually)
Preferred for layout?	Often problematic	Commonly recommended

Best Practice

Use this in most modern layouts:

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

CSS Flexbox

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

Ans.

CSS Flexbox (Flexible Box Layout) is a layout model in CSS that allows you to design a flexible and responsive layout structure without using float or positioning. It helps in aligning and distributing space among items in a container, even when their size is unknown or dynamic.

Why Flexbox is Useful for Layout Design:

- Makes it easy to create flexible and adaptive layouts.
- Automatically adjusts the layout to different screen sizes and resolutions.
- Simplifies alignment (vertical and horizontal) of elements.
- Offers powerful control over spacing, ordering, and sizing of items.

Key Terms:

Flex Container

- The parent element that holds the flex items.
- It is created by setting `display: flex` or `display: inline-flex` on an element.
- It controls the layout of its children (flex items) using various flex properties.

Example:

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

Flex Item

- The direct child elements of the flex container.
- These items are laid out according to the rules of the Flexbox model.

- You can control the behavior of each item using properties like `flex`, `order`, `flex-grow`, `align-self`, etc.

Example:

```
<div class="container">
  <div class="item1">Item 1</div>
  <div class="item2">Item 2</div>
</div>
```

Common Flexbox Properties:

On the flex container:

- `flex-direction`: row | row-reverse | column | column-reverse
- `justify-content`: flex-start | center | space-between | space-around
- `align-items`: stretch | center | flex-start | flex-end
- `flex-wrap`: nowrap | wrap | wrap-reverse

On the flex items:

- `flex`: shorthand for `flex-grow`, `flex-shrink`, and `flex-basis`
- `align-self`: override alignment per item
- `order`: controls item order

Example Code:

```
<style>
.container {
  display: flex;
```

```
  justify-content: space-between;
  align-items: center;
}
.item {
  padding: 20px;
  background-color: lightblue;
  margin: 5px;
}
</style>

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

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

Ans.

Here's a clear explanation of the **Flexbox properties**: `justify-content`, `align-items`, and `flex-direction` — all of which are applied to the **flex container**.

1. `flex-direction`

Defines the **main axis** along which flex items are placed in the container.

Values:

Value	Description
`row`	Default. Items are placed left to right.
`row-reverse`	Items go right to left.
`column`	Items are stacked top to bottom.
`column-reverse`	Items go bottom to top.

Example:

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

2. `justify-content`

Aligns **items along the main axis** (defined by `flex-direction`). Controls how the **extra space** is distributed between items.

Values:

Value	Description
`flex-start`	Items align at the start of the main axis.

`flex-end`	Items align at the end of the main axis.
`center`	Items are centered along the main axis.
`space-between`	Equal space **between** items.
`space-around`	Equal space **around** items (half-space on edges).
`space-evenly`	Equal space **between and around** items.

Example:

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

3. `align-items`

Aligns **items along the cross axis** (perpendicular to the main axis). Used for vertical alignment when the `flex-direction` is `row`, and horizontal when it's `column`.

Values:

Value	Description
`stretch`	Default. Items stretch to fill the container.

`flex-start`	Items align to the start of the cross axis.
`flex-end`	Items align to the end of the cross axis.
`center`	Items are centered along the cross axis.
`baseline`	Items align based on their text baseline.

Example:

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

Quick Visualization Example:

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

This layout:

- Places items in a **row**,
- **Evenly spaces** them across the row,
- **Vertically centers** them in the container.

CSS Grid

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

Ans.

What is CSS Grid?

CSS Grid Layout is a powerful layout system in CSS that allows you to create **2-dimensional layouts** using rows and columns. It's ideal for complex designs where you need precise control over both vertical and horizontal alignment.

To use Grid:

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

How Grid Differs from Flexbox:

Feature	CSS Grid	Flexbox

Layout type	**2D** (rows and columns)	**1D** (either row or column)
Direction control	Both rows & columns at once	Main axis (row or column), not both
Item placement	Uses **grid lines**, areas, and coordinates	Items flow in a single direction
Best for	Full-page or section layout grids	Aligning items in a row or column
Alignment tools	Grid-specific (e.g. `place-items`)	Flexbox-specific (e.g. `justify-content`)
Browser support	Fully supported (modern browsers)	Fully supported

When to Use Grid over Flexbox:

Use Grid when:

You need a full layout with **rows and columns**

You want items to span **multiple rows or columns**

You are designing a **complex UI structure**

The layout is not strictly in one direction

Use Flexbox when:

You are aligning items in **one row or column**

You want flexible content that wraps or stacks easily

You need simple alignment and spacing between items

Example: CSS Grid Layout

```
<style>
.container {
  display: grid;
  grid-template-columns: 1fr 2fr;
  grid-template-rows: auto auto;
  gap: 10px;
}
.item {
  background: lightblue;
  padding: 20px;
  text-align: center;
}
</style>

<div class="container">
```

```
  <div class="item">Header</div>
  <div class="item">Sidebar</div>
  <div class="item">Main Content</div>
  <div class="item">Footer</div>
</div>
```

(9) Describe the grid-template-columns, grid-template-rows, and grid-gapproperties. Provide examples of how to use them.

Ans.

Here's a detailed explanation of the **CSS Grid properties**: `grid-template-columns`, `grid-template-rows`, and `grid-gap`, along with examples.

1. `grid-template-columns`

Defines the **number and width of columns** in the grid.

Syntax:

```
grid-template-columns: <column-width> <column-width> ...;
```

Values:

- `px`, `em`, `%`: fixed widths
- `fr`: **fractional unit** of remaining space
- `auto`: adapts to content size
- `repeat(n, value)`: repeats the value `n` times

Example:

```
grid-template-columns: 200px 1fr 2fr;
```

Creates 3 columns: one fixed (200px), one flexible (`1fr`), one larger flexible (`2fr`)

2. `grid-template-rows`

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

Syntax:

```
grid-template-rows: <row-height> <row-height> ...;
```

Example:

```
grid-template-rows: auto 100px 1fr;
```

Creates 3 rows: one based on content, one fixed at 100px, and one flexible

3. `grid-gap` (shorthand for `row-gap` and `column-gap`)

Specifies the **spacing between rows and columns**.

New syntax (preferred): `gap` instead of `grid-gap`

Example:

```
grid-gap: 10px;
```

```
gap: 10px 20px;
```

Full Example Using All Three:

```
<style>
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  grid-template-rows: 100px auto;
```

```
  gap: 20px;
}
.item {
  background: lightblue;
  text-align: center;
  padding: 20px;
}
</style>

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

This creates:

- **3 columns**: 1 fraction, 2 fractions, 1 fraction
- **2 rows**: first is 100px high, second auto-sized
- **20px** space between all grid cells

Responsive Web Design with Media Queries

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

Ans.

What Are Media Queries in CSS?

Media queries are a feature in CSS that allow you to apply styles **only when certain conditions are met**, such as screen width, height, resolution, orientation, or device type. They're essential for **responsive web design**, where the layout adapts to different screen sizes (like mobile, tablet, or desktop).

Why Are Media Queries Important for Responsive Design?

They help you:

- Make websites look good on **all devices** (mobile, tablet, desktop)
- Adjust layout, font size, spacing, etc., based on **screen width or orientation**
- Avoid building separate sites for different devices
- Improve **user experience**, accessibility, and performance

Basic Syntax of Media Query:

```
@media media-type and (condition) {

 }
```

Common Media Types:

- `screen` – computer screens, tablets, smartphones

- `print` – print preview or printed documents
- `all` – applies to all devices (default)

Common Conditions (Media Features):

Feature	Example	Description
`min-width`	`(min-width: 600px)`	Applies if screen is ≥ 600px wide
`max-width`	`(max-width: 768px)`	Applies if screen is ≤ 768px wide
`orientation`	`(orientation: landscape)`	Applies if device is in landscape
`resolution`	`(min-resolution: 300dpi)`	High-resolution (e.g., retina)

Example: Responsive Layout Using Media Queries

```
body {
  background-color: lightgray;
  font-size: 16px;
}

@media screen and (min-width: 600px) {
  body {
    background-color: lightblue;
```

```
    font-size: 18px;
  }
}

@media screen and (min-width: 1024px) {
  body {
    background-color: lightgreen;
    font-size: 20px;
  }
}
```

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

Ans.

Here's a **basic media query** that adjusts the **font size** for screens **smaller than 600px**:

```
body {
  font-size: 18px;
}



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

Explanation:

- `@media screen and (max-width: 600px)` targets screens **up to 600px wide**, like smartphones.
- Inside the media query, the `body` font size is reduced to `14px` for better readability on small screens.

Typography and Web Fonts

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

Ans.

Difference Between Web-Safe Fonts and Custom Web Fonts

1. Web-Safe Fonts

Web-safe fonts are a set of **standard fonts** that are **pre-installed** on most operating systems and devices (Windows, macOS, Linux, etc.). Because they're already available, the browser doesn't need to download them.

Examples:

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

- Helvetica

Characteristics:

- Load instantly (no download needed)
- Very reliable across all platforms
- Limited in style and variety

2. Custom Web Fonts

Custom web fonts are fonts that are **not pre-installed** on the user's device. Instead, they are loaded via the web using services like **Google Fonts**, **Adobe Fonts**, or by embedding font files via `@font-face`.

Example using Google Fonts:

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


<style>

body {

  font-family: 'Roboto', sans-serif;

}

</style>
```

Characteristics:

- Greater design flexibility
- More stylish and modern fonts
- Slight performance impact (fonts must be downloaded)

- May cause a brief “flash of unstyled text” (FOUT)

Why Use a Web-Safe Font Over a Custom Font?

Reason	Explanation
Faster Load Times	Web-safe fonts don’t require downloading.
Better Performance	No extra HTTP requests or rendering delay.
High Compatibility	Works the same across all browsers and devices.
Fallback Readability	Always available as a safe fallback if custom fonts fail.

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

Ans.

What is the `font-family` Property in CSS?

The `font-family` property in CSS is used to **set the typeface** (font style) for text content on a webpage. It defines which fonts the browser should use to display the text.

Syntax:

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

- You can list **multiple fonts** as a fallback chain.
- Font names with spaces must be in **quotes**.
- Always include a **generic family** at the end (like `serif`, `sans-serif`, `monospace`).

Example:

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

If Arial is unavailable, the browser will try Helvetica, then any available sans-serif font.

How to Apply a Custom Google Font to a Webpage

To use a **Google Font**, you follow **two steps**:

Step 1: Import the font

Add the `<link>` tag provided by Google in the `<head>` section of your HTML file:

```
<!-- Example: Importing Roboto from Google Fonts -->
<link
href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">
```

Step 2: Use `font-family` in your CSS

Now apply it in your stylesheet like this:

```
body {
  font-family: 'Roboto', sans-serif;
}
```

Full Example (HTML + CSS):

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Google Font Example</title>
  <!-- Step 1: Link Google Font -->
  <link
href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">

  <style>
        body {
      font-family: 'Roboto', sans-serif;
      font-size: 18px;
    }
  </style>
</head>
```

```
<body>

  <h1>This text uses the Roboto font</h1>

  <p>Google Fonts help make your website more visually
appealing.</p>

</body>

</html>
```