CSS Assignment

CSS Selectors & Styling

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

ANSWER:-

- A **CSS selector** is a pattern used to select and style HTML elements. It tells the browser **which elements** the CSS rules should apply to.
- Types of Basic Selectors
- **Element Selector**

- Selects HTML elements by their tag name.

```
p {color:red;}
```

Example in HTML:

```
<p> Hello this is a  paragraph Tag</p>
```

- **Class Selector (.)**

Selects elements that have a specific class attribute.

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

This styles all elements with class="highlight" with a yellow background
Example in HTML:

```
<p class="highlight">This text is highlighted.</p>
```

- **ID Selector (#)**

- Selects a single unique element with a specific id.

```
#main-title {
font-size: 24px;
color: red;}
```

This applies only to the element with id="main-title".

Example in HTML:

```
<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**

CSS specificity is a set of rules that browsers use to determine which style rule should be applied when there are conflicting CSS declarations targeting the same element. Each CSS selector has a "weight" or specificity value, and the browser applies the style with the highest specificity.

- **Specificity Hierarchy**

From lowest to highest priority:

1. Universal selector (*), element/type selectors (p, h1) and pseudo-elements (::before, ::after)
→ Lowest specificity.
2. Class selectors (.class), attribute selectors ([type="text"]), and pseudo-classes (:hover, :first-child)
→ Medium specificity.
3. ID selectors (#id)
→ Higher specificity than classes.
4. Inline styles (style="color: red;")
→ Highest specificity (except !important).
5. !important rule
→ Overrides all other declarations, but should be used sparingly because it breaks normal cascading rules.

How Conflicts are Resolve

When multiple rules apply to the same element, the browser resolves conflicts as follows:

1. Compare Specificity: The rule with higher specificity wins.
   - Example: #box {} overrides .box {}.
2. Source Order (Last Rule Wins): If specificity is the same, the later rule in the CSS file is applied.
   - Example:
   - p { color: blue; }
   - p { color: red; } /* This will apply */
3. Use of !important: A property with !important overrides all others (unless another rule with !important has higher specificity).

Example:

```
p { color: black; }          /* Element selector */
p.intro { color: green; }    /* Class selector */
#special { color: blue; }    /* ID selector */
<p class="intro" id="special">Hello CSS</p>
```

Final color will be blue, because ID selectors have the highest specificity.

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

ANSWER:-

INLINE CSS

- Definition: Styles are applied directly to an element using the style attribute inside the HTML tag.
- Example:**<p style="color: red; font-size: 20px;">This is inline CSS</p>**
- Advantages:
  - Quick and easy to apply for single elements.
  - Useful for testing or overriding small styles.
- Disadvantages:
  - Hard to maintain for large projects.
  - Breaks separation of structure (HTML) and style (CSS).
  - Cannot reuse styles efficiently.

INTERNAL CSS

- Definition: Styles are written inside a `<style>` tag within the `<head>` section of the HTML document.
- Example:
- **<head>**
- **<style>**
- **p {**
- **color: blue;**
- **font-size: 18px;**
- **}**
- **</style>**
- **</head>**
- **<body>**
- **<p>This is internal CSS</p>**
- **</body>**
- Advantages:
  - Easy to style a single HTML page.
  - Keeps styles grouped together in one place.
- Disadvantages:
  - Styles cannot be reused across multiple pages.
  - Increases page size if used excessively.

EXTERNAL CSS

- Definition: Styles are written in a separate .css file and linked to the HTML document using the <link> tag.
- Example:
- **<head>**
- **<link rel="stylesheet" href="styles.css">**
- **</head>**
- **/* styles.css */**
- **p {**
- **color: green;**
- **font-size: 16px;**
- **}**
- Advantages:
  - Styles can be reused across multiple web pages.
  - Makes code cleaner and easier to maintain.
  - Reduces page size (HTML and CSS are separated).
- Disadvantages:
  - Requires an extra HTTP request (may slightly affect loading speed).
  - Styles may not load if the CSS file path is broken or missing.

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 describes how every HTML element is treated as a rectangular box, consisting of content, padding, border, and margin. Understanding the box model is important because it determines how elements are sized and spaced on a webpage.

Components of the Box Model

1. Content
   - The actual text, image, or other content inside the element.
   - Its size can be controlled using properties like width and height.
   - Example:
     ```
     p {
       width: 200px;
       height: 100px;
     }
     ```

2. Padding
   - The space between the content and the border.
   - Adds extra space inside the element but increases the total size.
   - Example:
     ```
     p {
       padding: 20px;
     }
     ```

3. Border
   - A line that wraps around the padding and content.
   - Has thickness (border-width), style (solid, dashed), and color.
   - Example:
     ```
     p {
       border: 5px solid black;
     }
     ```

4. Margin
   - The space outside the border, separating the element from others.
   - Does not affect the content size, only the spacing around the element.

- Example:
  ```
      p {
  margin: 15px;
      }
  ```

<u>How Each Affects the Size of an Element</u>

The total size of an element is calculated as:

- Total Width = content width + padding (left + right) + border (left + right) + margin (left + right)
- Total Height = content height + padding (top + bottom) + border (top + bottom) + margin (top + bottom)

Example:
If an element has:

- width: 200px;
- padding: 10px;
- border: 5px solid;
- margin: 20px;

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

<u>**ANSWER:-**</u>

1. <u>content-box (Default)</u>

- The width and height you set apply only to the content area.
- Padding and border are added outside of the specified width/height, increasing the total element size.

<u>Example:</u>

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

- Content width = 200px
- Total width = 200 + (20+20) + (10+10) = 260px

2. border-box

- The width and height include the content, padding, and border.
- The total size of the element stays fixed.
- Content area shrinks if padding/border is added.

Example:

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

- Total width = exactly 200px (padding + border included).
- Content width = 200 - (20+20) - (10+10) = 140px

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:-

- Flexbox (Flexible Box Layout) is a CSS layout model that makes it easier to design responsive and flexible layouts.
- It allows elements to automatically adjust their size, order, and alignment to fit different screen sizes and available space.
- Unlike traditional block/inline layouts, Flexbox is direction-aware (horizontal or vertical) and helps distribute space between items dynamically.

Why Flexbox is Useful for Layout Design

- Creates responsive layouts without complex float or positioning hacks.
- Provides easy alignment and spacing (both horizontally and vertically).
- Automatically handles resizing of items to fit available space.
- Useful for navigation bars, cards, grids, forms, and responsive designs.

Key Terms in Flexbox

Flex-container

- The parent element that uses display: flex; (or display: inline-flex;).
- It defines the flex context for its children.
- Properties that apply to the container include:
  - flex-direction → row, column, row-reverse, column-reverse
  - justify-content → alignment along the main axis
  - align-items → alignment along the cross axis
  - flex-wrap → whether items wrap to the next line

Example:

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

Flex-items

- The child elements of a flex container.
- Their size and behavior are controlled using flex properties like:
  - flex-grow → how much the item can grow

  - flex-shrink → how much the item can shrink
  - flex-basis → initial size of the item
  - align-self → overrides container alignment for a single item

Example:

```
.item {
  flex-grow: 1;   /* Item expands to fill space */
  flex-basis: 200px; /* Initial size */
}
```

Visual Example

```
<div class="container">
  <div class="item">Box 1</div>
  <div class="item">Box 2</div>
  <div class="item">Box 3</div>
</div>
.container {
  display: flex;
  justify-content: space-around;
  align-items: center;
}
.item {
  background: lightblue;
  padding: 20px;
}
```

.container = flex-container

.item = flex-items

In short:

- Flexbox = powerful layout model for modern responsive design.
- Flex-container = parent element (`display: flex;`).
- Flex-items = children inside the container that adjust based on flex rules.

❖ **Question 2: Describe the properties justify-content, align-items, and flex-direction used in Flexbox.**

ANSWER:-

Flexbox provides powerful properties to control how elements are placed and aligned inside a flex container. Three of the most important ones are:

1. flex-direction

- Defines the main axis (the direction in which flex-items are placed).
- Values:
  - row → items are placed left to right (default).
  - row-reverse → items are placed right to left.
  - column → items are placed top to bottom.
  - column-reverse → items are placed bottom to top.

Example:

```
.container {
  display: flex;
  flex-direction: row; /* horizontal layout */
}
```

2. justify-content

- Aligns items along the main axis (horizontal if row, vertical if column).
- Values:
  - flex-start → items packed at the start.
  - flex-end → items packed at the end.
  - center → items centered.
  - space-between → equal space between items.
  - space-around → equal space around items.
  - 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).
- Values:
  - flex-start → items aligned at start of cross axis.
  - flex-end → items aligned at end of cross axis.
  - center → items centered on cross axis.
  - stretch → items stretched to fill container (default).
  - baseline → items aligned by text baseline.

Example:

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

Visual Summary

Property	Works On	Main Purpose	Common Values
flex-direction	Container	Defines direction of items (row/column)	row, column, row-reverse, column-reverse
justify-content	Container	Aligns items on main axis	flex-start, flex-end, center, space-between, space-around, space-evenly
align-items	Container	Aligns items on cross axis	flex-start, flex-end, center, stretch, baseline

In short:

- `flex-direction` sets layout direction.
- `justify-content` controls main-axis alignment.
- `align-items` controls cross-axis alignment.

CSS Grid

❖ **Question 1: Explain CSS Grid and how it differs from Flexbox. When would you use Grid over Flexbox?**

ANSWER:-

- CSS Grid Layout is a two-dimensional layout system in CSS.
- It allows you to design web layouts by dividing a page into rows and columns.
- With Grid, you can precisely position elements in both horizontal (rows) and vertical (columns) directions at the same time.

Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr; /* 3 equal columns */
  grid-template-rows: auto 200px;
  gap: 10px;
}
```

Feature	Flexbox	Grid
Layout type	One-dimensional (row OR column)	Two-dimensional (row AND column)
Alignment	Focus on aligning items in one direction	Controls layout in both directions
Best for	Distributing space, aligning items	Complex layouts, full page/section designs
Item placement	Based on content flow	Items can be explicitly placed in rows/columns
Use case	Navigation bars, buttons, lists	Web page templates, dashboards, galleries

---

- Use Flexbox when:
  - You need to align elements in a single direction (row or column).
  - Layout is simple and mostly about distributing space.
  - Example: Navbars, toolbars, buttons.
- Use Grid when:
  - You need a two-dimensional layout (rows and columns together).
  - Layout requires precise control over positioning.
  - Example: Page layouts, image galleries, dashboards, forms.

❖ **Question 2: Describe the grid-template-columns, grid-template-rows, and grid-gap properties. Provide examples of how to use them.**

ANSWER:-

`grid-template-columns`

- Defines the number of columns and their widths in a grid.
- You can use units like `px`, `%`, `fr` (fractional unit), or `auto`.

Example:

```
.container {
  display: grid;
  grid-template-columns: 200px 1fr 2fr;
}
```

- 1st column = fixed 200px
- 2nd column = takes 1 fraction of remaining space
- 3rd column = takes 2 fractions of remaining space

grid-template-rows

- Defines the number of rows and their heights in a grid.

Example:

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

- 1st row = fixed 100px
- 2nd row = adjusts to content (auto)
- 3rd row = fixed 50px

grid-gap (or gap)

- Defines the spacing between rows and columns (without affecting outer margins).
- Can set one value (applies to both) or two values (`row-gap column-gap`).

Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-gap: 20px;         /* 20px space between rows and columns */
  /* or */
  gap: 10px 30px;         /* 10px between rows, 30px between columns */
}
```

Complete Example

```
<div class="container">
```

```
  <div class="box">1</div>
  <div class="box">2</div>
  <div class="box">3</div>
  <div class="box">4</div>
  <div class="box">5</div>
  <div class="box">6</div>
</div>
.container {
  display: grid;
  grid-template-columns: 100px 1fr 2fr;
  grid-template-rows: 80px auto;
  grid-gap: 15px;
}
.box {
  background: lightblue;
  padding: 20px;
  text-align: center;
}
```

This creates a 3-column grid with different widths, 2 rows, and 15px spacing between items.

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 are a CSS feature that allows you to apply different styles based on the device characteristics such as screen size, resolution, or orientation.
- They make websites responsive, meaning the layout adapts to desktops, tablets, and mobile devices.

Example:

```
/* For screens up to 600px wide (mobile) */
@media (max-width: 600px) {
  body {
    background-color: lightblue;
    font-size: 14px;
  }
}

/* For screens larger than 600px (desktop/tablet) */
@media (min-width: 601px) {
  body {
    background-color: white;
    font-size: 18px;
  }
}
```

Why are Media Queries Important for Responsive Design?

1. Device Adaptability – Adjusts layout for phones, tablets, laptops, and large screens.
2. Improved User Experience – Ensures text, buttons, and images are readable and well-placed on all devices.
3. Performance Optimization – Allows loading smaller images/styles for mobile devices.
4. Future-Proof Design – Websites stay usable as new devices with varying screen sizes appear.

Common Features Used in Media Queries

- `max-width` → Styles apply up to a certain screen width.
- `min-width` → Styles apply from a certain screen width and above.
- `orientation` → Detects landscape or portrait mode.
- `resolution` → Targets high-resolution (Retina) displays.

Example:

```
/* Portrait mode */
```

```
@media (orientation: portrait) {
  img {
    width: 100%;
  }
}

/* Landscape mode */
@media (orientation: landscape) {
  img {
    width: 50%;
  }
}
```

❖ **Question 2: Write a basic media query that adjusts the font size of a webpage for screens smaller than 600px.**

ANSWER:-

To adjust the font size of a webpage for screens smaller than 600px, you can use the `@media` rule with `max-width`.

Example:

```
/* Default font size for larger screens */
body {
  font-size: 18px;
}

/* For screens smaller than 600px */
@media (max-width: 600px) {
  body {
    font-size: 14px;
  }
}
```

Explanation

- `max-width: 600px` → The styles inside this block will only apply when the screen width is 600px or less (like mobile devices).
- On larger screens (>600px), the default `font-size: 18px;` will be applied.

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:-

Difference Between Web-Safe Fonts and Custom Web Fonts

Web-Safe Fonts

- Definition: Fonts that are commonly pre-installed on most operating systems (Windows, macOS, Linux, Android, iOS).
- They are widely supported and do not require downloading.
- Examples: `Arial`, `Times New Roman`, `Courier New`, `Verdana`, `Georgia`.

Advantages:

- Always available → No need to load from external source.
- Faster loading time (improves performance).
- Consistent rendering across devices.

Disadvantages:

- Limited choices (only a handful of fonts are truly “web-safe”).
- Can look generic or less unique in design.

Custom Web Fonts

- Definition: Fonts that are not pre-installed but are loaded from the web using services like Google Fonts or `@font-face`.
- Examples: `Roboto`, `Open Sans`, `Lato`, `Montserrat`.

Advantages:

- Huge variety → Designers can create unique branding.
- Supports different weights, styles, and icons.
- Makes websites visually more appealing.

Disadvantages:

- Requires downloading → Can slow down page load.
- If not loaded properly, the browser may fall back to a default font (flash of unstyled text, FOUT).
- May not be available offline.

Why Use Web-Safe Fonts Over Custom Fonts?

1. Performance: Faster loading because no extra font files are downloaded.
2. Reliability: Guaranteed to work across all devices and browsers.
3. Fallback Option: Often used as a fallback when a custom font fails to load.

Example:

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

- `"Open Sans"` = custom web font
- `Arial` = web-safe fallback
- `sans-serif` = generic fallback

❖ **Question 2: What is the font-family property in CSS? How do you apply a custom Google Font to a webpage?**

ANSWER:-

What is the font-family Property?

- The `font-family` property in CSS specifies which font should be used for text.
- You can provide a list of fonts in order of priority → if the first font is not available, the browser will use the next one.

Example:

```
p {
  font-family: "Times New Roman", Georgia, serif;
}
```

- The browser will try `"Times New Roman"`.
- If unavailable → uses `Georgia`.
- If both fail → uses the generic `serif` font.

Applying a Custom Google Font

To use a Google Font, follow these steps:

Step 1: Import the font

- Go to Google Fonts, choose a font (e.g., *Roboto*).
- Copy the `<link>` tag provided by Google.

Example (in `<head>` of HTML):

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

Step 2: Apply the font using CSS

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

- `'Roboto'` → Google custom font
- `Arial` → web-safe fallback
- `sans-serif` → generic fallback