CSS Basics-Theory Assignment

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

- A **CSS selector** is the part of a CSS rule that defines *which* HTML element(s) the styles should apply to.
  It "selects" elements in the HTML document based on their tag, class, ID, attributes, or relationships.

Types of Selectors with Examples:

1. **Element Selector**
   - Targets all elements of a specific HTML tag.

```
p {
  color: blue;
  font-size: 16px;
}
```

2. **Class Selector**
   - Targets elements with a specific class attribute.
   - Classes are reusable on multiple elements.

```
 /* This will style all elements with class="highlight" */
.highlight {
  background-color: yellow;
  font-weight: bold;
}

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

3. **ID Selector**
    - Targets an element with a specific id attribute.
    - IDs are unique in a page (should be used once per element).

```
/* This will style the element with id="main-title" */
#main-title {
  color: red;
  text-align: center;
}

<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?

- ➢ Specificity is the way which helps the browsers to decide which property value is most relevant for the element. It determines which style declaration is applied to an element.
- ➢ **Universal selectors (*)** and the inherited values have lower specificity, i.e., 0 specificity.
- ➢ The style property has a greater specificity value compare to the selectors (except the !important in the stylesheet selector).
- ➢ **The !important** alter the selector specificity. When two selectors have equal specificity, then the selector having !important hierarchy Inline styles: It is directly attached to the element which is to be styled. For example:
- ➢ It has the highest priority.
- ➢ **IDs:** It is a unique identifier for the elements of a page that has the second- highest priority. For example: #para. Classes, attributes
- ➢ **pseudo-classes:** It includes classes, attributes, and pseudo-classes (like :focus, :hover, etc.).

- **Elements and pseudo-elements:** It includes the name of elements (div, h1) and pseudo-elements (like :after and :before). They have the lowest priority

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

1. Inline CSS

- Defined *directly inside an HTML element* using the style attribute.

  <p style="color: red; font-size: 18px;">This is inline CSS</p>

  ✅ **Advantages:**

- Quick to apply for a **single element**.
- Useful for **testing or debugging** styles.
- No need to create a separate CSS file.

  ❌ **Disadvantages:**

- Not reusable (must rewrite styles for each element).
- Mixes content (HTML) with design (CSS) → **bad for maintainability**.
- Hard to manage for large projects.
- Lowest scalability.

2. Internal CSS

- Defined *inside the HTML file* within a <style> tag (usually inside <head>).

```
<head>
  <style>
    p {
      color: blue;
      font-size: 16px;
    }
  </style>
</head>
```

✅ **Advantages:**

- Styles are in one place (instead of inline scattered everywhere).
- Useful for **small websites** or **single-page projects**.
- Can override external CSS for that page.

❌ **Disadvantages:**

- Only works for **one page** (not reusable across multiple pages).
- Increases page size if repeated in many HTML files.
- Harder to maintain when website grows.

3. External CSS

- Defined in a **separate .css file** and linked to HTML using <link>.

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

styles.css:

```
p {
  color: green;
  font-size: 14px;
}
```

✅ **Advantages:**

- **Best for large projects** → reusable across multiple pages.
- Keeps HTML clean (separates content from design).
- Easier to maintain and update.
- CSS file can be cached by browser → faster loading.

❌ **Disadvantages:**

- Requires an **extra HTTP request** (one more file to load).
- If CSS file is missing/not linked correctly → page looks unstyled.

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

➢ The **CSS Box Model** is a fundamental concept that describes how every HTML element is structured and how its size and spacing are calculated on a webpage.

➢ Each element in a webpage is considered as a **box** made up of four parts:

👉 **Content**, **Padding**, **Border**, and **Margin**.

1. Content

- **What it is:** The innermost area where the actual text, image, or other content appears.
- **Controlled by:** width, height, font-size, etc.

2. Padding

- **What it is:** The space **between** the content and the border.
- **Controlled by:** padding, padding-top, padding-right, padding-bottom, padding-left.

3. Border

- **What it is:** The line that wraps around the padding and content.
- **Controlled by:** border-width, border-style, border-color.

4. Margin

- **What it is:** The space **outside** the border, separating the element from others.
- **Controlled by:** margin, margin-top, margin-right, margin-bottom, margin-left.

Question 5: What is the difference between border-box and content-box box-sizing in CSS? Which is the default?

1. content-box (default)

- **Description:**
  The width and height properties apply **only to the content area**. Padding and borders are added **outside** of that area.

- **Formula:**
- Total width = content width + padding + border

Total height = content height + padding + border

```
div {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
  box-sizing: content-box; /* default */
}
```

→ Total width = 200 (content) + 40 (padding) + 10 (border) = 250px

2. border-box

- **Description:**
  The width and height include the **content, padding, and border** — all inside the specified size.
  So, padding and border **don't increase** the element's total size.
- **Formula:**
- Total width = specified width
- Total height = specified height
- **Example:**

```
 div {
      width: 200px;
      padding: 20px;
      border: 5px solid black;
      box-sizing: border-box;
     }
```

→ Total width = **200px exactly** (the content shrinks to make room for padding and border)

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

- **Flexbox (Flexible Box Layout)** is a CSS layout module designed to make it **easier to arrange, align, and distribute space** among items in a container — even when their size is unknown or dynamic.

1. **Flex Container**

- The **parent element** that holds flex items.
- Defined by setting:
- display: flex;
- Once you do this, all direct children become **flex items**.
- The container controls **layout direction, wrapping, alignment, and spacing**.

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;          /* Defines a flex container */
  justify-content: center; /* Aligns items horizontally */
  align-items: center;     /* Aligns items vertically */
}
```

2. Flex Items

- The **child elements** inside a flex container.
- They can **grow, shrink, or adjust** automatically to fill available space.
- Each flex item can be individually controlled using properties like:
  - flex-grow
  - flex-shrink
  - flex-basis
  - align-self

Example:

```
.item {
  flex-grow: 1; /* Each item expands equally to fill space */
}
```

Question 7: Describe the properties justify-content, align-items, and flexdirection used in Flexbox.

1. **flex-direction**

   Purpose:
   Defines **the direction in which flex items are placed** inside the flex container.

   Syntax:

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

Value	**Description**	**Main Axis Direction**
row (default)	Items are placed **left to right**	Horizontal
row-reverse	Items are placed **right to left**	Horizontal
column	Items are placed **top to bottom**	Vertical
column-reverse	Items are placed **bottom to top**	Vertical

2. **justify-content**

   Purpose:
   Aligns flex items **along the main axis** (which depends on the flex-direction).

   Syntax:

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

Value	**Description**
flex-start	Items align **at the start** of the main axis (default).
flex-end	Items align **at the end** of the main axis.

Value	Description
center	Items are **centered** along the main axis.
space-between	Equal space **between** items; none at ends.
space-around	Equal space **around** items (half at edges).
space-evenly	Equal space **between and around** all items.

3. align-items

Purpose:
Aligns flex items **along the cross axis** (perpendicular to the main axis).

Syntax:

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

Value	Description
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** vertically (if flex-direction: row).
baseline	Items align according to **text baselines**.
stretch (default)	Items **stretch** to fill the container’s cross axis.

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

- A **grid container** is defined using display: grid;
- Inside it, **grid items** (child elements) are automatically aligned into **rows and columns**.
- You can control the layout using properties like:
  - grid-template-columns
  - grid-template-rows
  - grid-gap (or gap)
  - grid-column and grid-row (to control how items span)

Feature	**CSS Grid**	**Flexbox**
Layout Type	Two-dimensional (rows & columns)	One-dimensional (either a row *or* a column)
Main Purpose	Building full page layouts or complex grid-based designs	Aligning items in a line or small sections
Alignment	Can align both horizontally *and* vertically at the same time	Aligns items along one axis (main or cross)
Item Placement	Items can be placed anywhere in the grid (using row/column lines)	Items are placed in order (cannot easily skip positions)
Syntax	Uses grid-template-rows, grid-template-columns, grid-area, etc.	Uses justify-content, align-items, flex-direction, etc.

Question 9: Describe the grid-template-columns, grid-template-rows, and grid-gap properties. Provide examples of how to use them.

1. grid-template-columns

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

Syntax:

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

Each <track-size> defines the width of a column.
You can use values like px, %, fr, or auto.

Example:

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

2. grid-template-rows

This property defines **the number and height of rows** in a grid layout.

Syntax:

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

Example:

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

3. grid-gap (or gap)

This property adds **spacing between rows and columns** in the grid.

Note: grid-gap is now replaced by the shorter gap property in modern CSS (works for both Grid & Flexbox).

Syntax:

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

You can set both gaps together or individually using:

- row-gap
- column-gap

Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 150px 150px;
  gap: 20px;
}
```

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

Media queries are **CSS rules** that allow you to apply **different styles** to a webpage **based on the device's characteristics**, such as:

- **Screen size (width & height)**
- **Device type (mobile, tablet, desktop, etc.)**
- **Orientation (portrait or landscape)**

- **Resolution**

They make it possible for your website to **adapt its layout and appearance** to fit any screen — large or small.

Syntax of a Media Query

```
@media (condition) {
  /* CSS rules go here */
}
```

How it works:

- When the screen width is **below 768px**, the background becomes **light green**.
- When it's **below 480px**, it becomes **light coral** and the font size reduces.

Question 11 : Write a basic media query that adjusts the font size of a webpage for screens smaller than 600px.

```
/* Default font size for larger screens */
body {
  font-size: 18px;
}
/* Media query for screens smaller than 600px */
@media (max-width: 600px) {
  body {
    font-size: 14px;
  }
}
```

Explanation:

- @media (max-width: 600px) → targets screens **up to 600 pixels wide** (like most phones).
- Inside the braces { }, you place the styles that should apply only when that condition is true.
- In this case, the font size is reduced to make text easier to read on smaller screens.

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

Feature	**Web-Safe Fonts**	**Custom Web Fonts**
Definition	Fonts that are pre-installed on most devices and don't need to be downloaded.	Fonts that are downloaded from the internet (like Google Fonts or Adobe Fonts).
Examples	Arial, Verdana, Times New Roman, Georgia	Poppins, Lato, Roboto, Montserrat
Loading Speed	Very fast (already on the system)	Slightly slower (font files must load)
Design Variety	Limited styles and choices	Wide range of creative styles
Consistency	Looks the same on all devices	May vary if the font fails to load

Feature	**Web-Safe Fonts**	**Custom Web Fonts**
Internet Requirement	No internet needed	Needs internet to load the font file
Usage Best For	Simple, fast-loading websites or emails	Modern, branded websites with custom styles
Example CSS	font-family: Arial, sans-serif;	font-family: 'Poppins', sans-serif;

Why Use a Web-Safe Font Instead of a Custom Font?

You might choose a **web-safe font** when:

- You want your site to **load faster**.
- You need your text to **display perfectly everywhere**, even on old browsers.
- You're creating **HTML emails** (which often don't support custom fonts).

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

The **font-family** property in CSS is used to **specify the typeface (font)** for text on a webpage.
It defines which font or group of fonts should be used for an element.

Syntax:

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

Example:

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

- "Arial" → main font
- "Helvetica" → backup if Arial isn't available
- sans-serif → generic fallback family

How to Apply a Custom Google Font to a Webpage

You can add a Google Font in **two simple steps**:

Step 1: Import the font

Add the Google Font link inside the <head> section of your HTML file.
Example using the **Poppins** font:

```
<link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;
600&display=swap" rel="stylesheet">
```

Step 2: Use it in CSS

Now apply it using the font-family property:

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