Css que ans ans

**CSS Selectors & Styling**

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

A **CSS selector** is a pattern used to select and style HTML elements on a webpage. It defines which elements the CSS rules should apply to, based on attributes like element type, class, or ID.

### **Types of CSS Selectors:**

1. **Element Selector**: This selects all elements of a specific type.
   * Example: To select all <p> (paragraph) elements:

p {

color: blue;

}

1. **Class Selector**: This selects elements with a specific class. Classes are defined with a period . before the class name.

Example: To select all elements with the class highlight:

.highlight {

background-color: yellow;

}

1. **ID Selector**: This selects a single element with a specific ID. IDs are defined with a hash # before the ID name.
   * Example: To select an element with the ID header:

#header {

font-size: 24px;

}

**Question 2:** Explain the concept of CSS specificity. How do conflicts between multiple

### styles get resolved?

**ANS:** CSS specificity is a mechanism that determines which CSS rule is applied to an element when multiple rules could apply. It's used to resolve conflicts between styles by assigning a hierarchy to selectors. When multiple rules target the same element, the rule with the highest specificity wins.

### **How Specificity is Calculated:**

Specificity is calculated based on a four-part value, often represented as:

(a, b, c, d)

Where:

a is the number of inline styles applied to the element (highest specificity).

b is the number of ID selectors in the rule.

c is the number of class selectors, attributes selectors, and pseudo-classes.

d is the number of element selectors and pseudo-elements.

### **The Rules for Specificity:**

1. Inline styles have the highest specificity (represented as a).
   * Example: <div style="color: red;"> will override any external or internal CSS.
   * Specificity: (1, 0, 0, 0)
2. ID selectors are more specific than classes and element selectors.
   * Example: #header { color: blue; } has a specificity of (0, 1, 0, 0).
3. Class selectors, attribute selectors, and pseudo-classes have the same specificity level.
   * Example: .container { color: green; } or input[type="text"] { color: green; } has a specificity of (0, 0, 1, 0).
4. Element selectors and pseudo-elements have the lowest specificity.
   * Example: p { color: black; } has a specificity of (0, 0, 0, 1).

### **How Conflicts Between Multiple Styles Are Resolved**:

When multiple CSS rules target the same element, the rule with the higher specificity is applied. If two rules have the same specificity, the one that appears last in the CSS code (or the later in the stylesheet, if there are multiple stylesheets) takes precedence.

### **Question 3:** What is the difference between internal, external, and inline CSS? Discuss

### the advantages and disadvantages of each approach.

**ANS:**CSS (Cascading Style Sheets) can be applied to HTML in three primary ways: internal, external, and inline. Each approach has its own advantages and disadvantages, depending on the context in which it is used.

### **1. External CSS**

External CSS involves linking to a separate CSS file from an HTML document. The styles are written in a separate .css file and are linked to the HTML file using the <link> tag in the <head> section of the HTML document.

#### Example:

html

<head>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

CSS (styles.css):

body {

background-color: #f0f0f0;

}

h1 {

color: blue;

}

#### **Advantages of External CSS:**

1. Separation of Concerns: HTML structure is separate from styling, making the code more organized and easier to maintain.
2. Reusability: The same CSS file can be linked to multiple HTML files, making it easy to apply the same styles across a whole website.
3. Performance: External CSS files are cached by the browser, reducing load times for subsequent page visits.
4. Cleaner HTML: HTML code remains uncluttered with CSS rules, making it more readable and maintainable.

#### **Disadvantages of External CSS:**

1. Multiple HTTP Requests: An additional HTTP request is needed to load the external CSS file, which could increase page load times, especially for larger files or slow network connections.
2. Dependency on File: If the external CSS file is missing or corrupted, the webpage may not display properly, as the styling would be absent.

### **2. Internal CSS**

Internal CSS is placed directly within the <style> tag in the <head> section of the HTML document. This allows the styles to apply to that specific HTML document only.

#### Example:

html

<head>

<style>

body {

background-color: #f0f0f0;

}

h1 {

color: blue;

}

</style>

</head>

#### **Advantages of Internal CSS**:

1. Convenient for Single Page: Ideal for styling a single HTML document without needing an external CSS file.
2. No Extra HTTP Requests: Since the styles are embedded in the HTML file, there is no need for an additional request to load a CSS file, which can slightly improve load times.
3. Easier for Small Projects: Great for small, simple projects where you don't need to worry about the overhead of managing external files.

#### **Disadvantages of Internal CSS:**

1. Lacks Reusability: Styles are limited to the specific HTML document, so the same styles cannot be reused across multiple pages without duplicating the CSS.
2. Makes HTML Less Organized: The HTML file becomes less readable as it mixes content and presentation. For larger projects, this can lead to code bloat.
3. Not Cached: The styles are not cached like external CSS, so every time the page is loaded, the styles are reloaded, which can affect performance in larger projects.

### 3. **Inline CSS**

Inline CSS is applied directly to individual HTML elements using the style attribute within the HTML tags.

#### Example:

<h1 style="color: blue; background-color: #f0f0f0;">Hello World</h1>

#### **Advantages of Inline CSS:**

1. Quick and Specific: Ideal for quick, one-off styling that only needs to apply to a single element.
2. No Need for External or Internal Files: All styles are directly within the HTML tag, so you don't need an additional CSS file or <style> section.
3. Overrides Other Styles: Inline styles have a high specificity and will override any conflicting external or internal styles unless !important is used in the other CSS rules.

#### **Disadvantages of Inline CSS:**

1. No Reusability: Styles cannot be reused across multiple elements or pages, leading to redundant code if the same styles are needed elsewhere.
2. Clutters HTML: Inline styles mix presentation with content, making the HTML code less readable and harder to maintain, especially in large documents.
3. Difficult to Maintain: Changes to the design require modifying individual elements, which is time-consuming and error-prone for larger websites.
4. Performance Issues: Since the styles are applied directly to each element, the HTML file can become bulky, affecting page load times and making it difficult to cache.

**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?

**ANS:**The CSS box model is a fundamental concept in web design and layout, determining how elements are displayed and sized in a webpage. It describes the rectangular boxes that wrap around HTML elements and includes the following components:

### **1. Content**

Definition: This is the innermost part of the box, where the actual content (text, images, etc.) of the element resides.

Effect on Size: The content area defines the base size of the element, as it is where the width and height are applied (using the width and height properties). It does not include padding, border, or margin.

Example: If you set width: 200px; height: 100px;, the content area will be 200px by 100px.

### **2. Padding**

Definition: Padding is the space between the content and the border of the element. It is an area of transparency around the content.

Effect on Size: Padding increases the total size of the element. If padding is applied on all sides (top, right, bottom, left), it will add to the width and height. So, the total width and height will increase by the padding values on each side.

**Example**: If an element has padding: 20px; and the content area is 200px by 100px, the total width will become 240px (200px + 20px on the left and 20px on the right), and the total height will be 140px (100px + 20px on the top and 20px on the bottom).

### **3. Border**

Definition: The border wraps around the padding (if present) and content. You can customize the border’s width, style (solid, dotted, dashed, etc.), and color.

Effect on Size: Like padding, the border increases the overall size of the element. If you add a border, it adds to the width and height beyond the content and padding areas.

**Example**: If you have border: 5px solid black; in addition to the previous padding, the total width will increase by 10px (5px left and 5px right), and the total height will increase by 10px (5px top and 5px bottom).

### **4. Margin**

**Definition**: The margin is the space outside the border. It separates the element from other surrounding elements. Margins are always transparent.

Effect on Size: Unlike padding and border, margin does not affect the size of the element itself but affects the positioning and spacing between elements. The total space between elements will increase as the margin increases.

Example: If you add margin: 30px;, it will push the element away from neighboring elements, but the total size of the element (content + padding + border) will remain unchanged.

### **Calculating Total Size of an Element**

The total size of an element can be calculated by adding the dimensions of all the components (content, padding, border) along with any margin (although margin does not affect the box size, it affects spacing):

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

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

Note: If the CSS property box-sizing is set to border-box, then the padding and border are included in the width and height. This can help avoid the issue of the element growing unexpectedly when padding and borders are added.

### Example of box-sizing: border-box:

div {

width: 200px;

height: 100px;

padding: 20px;

border: 5px solid black;

box-sizing: border-box; /\* Includes padding and border within the specified width/height \*/

}

Without box-sizing: border-box, the total size would be 240px wide (200px + 20px padding + 5px border) and 140px high (100px + 20px padding + 5px border). With box-sizing: border-box, the total size remains 200px by 100px, as the padding and border are included within the specified dimensions.

**Question 2:** What is the difference between border-box and content-box box-sizing inCSS?

Which is the default?

**ANS:**

**1. Content-box:**

This is the default value for the box-sizing property. When the box-sizing is set as content-box, width and height are measured from the content only. Any padding and borders add to the outside of the width and height.

**Calculation:**

width = content width + padding + border

height = content height + padding + border

**2. border-box:**

Definition: When box-sizing is set to border-box, the width and height of the element include the content, padding, and border. That is, the specified width and height will be the total size of the element.

**Calculation:**

Width = content width (including padding and border)

Height = content height (including padding and border)

Default Value

The default value for box-sizing is content-box.

**Example**

/\* content-box (default) \*/

.box1

width: 200px; /\* total width = 200px + padding + border \*/

padding: 20px;

border: 5px solid black;

}

/\* border-box \*/

.box2 {

box-sizing: border-box;

width: 200px; /\* total width = 200px (includes padding and border) \*/

padding: 20px;

border: 5px solid black;

}

**CSS Flexbox**

### **Question 1:** What is CSS Flexbox, and how is it useful for layout design? Explain the terms

### flex-container and flex-item.

### **ANS:**CSS Flexbox is short for Flexible Box Layout. It is a layout model in CSS that makes it possible to design complex layouts with ease. Flexbox is designed to distribute space along a single axis, either horizontal or vertical, and align items within a container, making it much easier to create responsive, flexible, and dynamic layouts.

### 

### With Flexbox, the size and position of elements are determined by the container, and the layout adjusts dynamically according to the space available. Flexbox eliminates many of the traditional layout techniques, such as floats or positioning, and provides a much more efficient method for aligning items within a container.

### 

### **How Is Flexbox Helpful in Layout Design?**

### Flexbox offers the following advantages for layout design:

### 

### **Alignment and Centering:** Flexbox makes centering content both horizontally and vertically much easier than previous techniques.

### **Responsive Layout:** Flexbox is very flexible, responsive to size changes in the screen, which is just ideal when building responsive web designs.

### **Ordering:** Flexbox can rearrange elements in a different visual order without altering the underlying HTML structure, making it perfect for reordering items in a container.

### **Distribution of Space:** Flexbox can evenly distribute space between items, ensuring that items scale and adapt to different container sizes.

### **Dynamic Sizing:** It provides options to create layouts where items can grow, shrink, or stay fixed based on the available space.

### Key Concepts in Flexbox:

### To understand how Flexbox works, you need to understand the two primary components involved in the Flexbox layout: flex-container and flex-item.

### 

### **1. Flex-container**

### **Definition:** The flex-container is the parent element that contains the flex items (children). It is the element that is made into a flexbox by applying display: flex or display: inline-flex.

### 

### **Role:** The flex-container determines how the flex-items are arranged, aligned, and spaced. It sets up the flex context for its child elements.

### 

### **Key properties of flex-container:**

### 

### **display:** flex or display: inline-flex: Makes the container a flexbox.

### **flex-direction:** It gives the direction of the flex items, which may be horizontal in nature, i.e., row, row-reverse, or vertical, i.e., column, column-reverse.

### **justify-content:** Aligns flex items along the main axis, i.e., either horizontally or vertically.

### **align-items:** Aligns the flex items along the cross-axis, which is perpendicular to the main axis.

### **align-content:** Aligns the multiple rows or columns in case of extra space.

### Example:

### 

### 

### 

### 

### 

### .container {

### display: flex; /\* Creates a flex container \*/

### justify-content: space-between; /\* Distributes space between items \*/

### }

### 2. Flex-item

### Definition: Flex-items are child elements of a flex-container. They are elements whose arrangement and alignment are determined according to the flex-container rules.

### 

### Role: The role of a flex-item is to be placed and sized as the flex-items within the container by the applied properties of flexbox on the container.

### 

### **Main Properties of Flex-Items**

### 

### **flex-grow :** Determines the amount that a flex-item can expand to fill out extra space left between flex items in case space exists.

### **flex-shrink:** Specifies that a flex-item can shrink if the size of the container is reduced.

### **flex-basis:** Specifies a flex-item's initial size prior to any shrink or grow occuring.

### **order:** Determines how flex-items display visually, possibly in a different order than the original HTML.

### **align-self:** lets each item display differently along its cross-axis if it wants, overriding align-items on the container.

### Example

### 

### 

### 

### .item

### flex-grow: 1; /\* The item will take up all available space \*/

### flex-basis: 100px; /\* The item will start with a width of 100px \*/

### }

### **How Flexbox Works:**

### **Flex-container:** Once you set an element to display: flex, it becomes a flex-container. The direct children of this container become flex-items.

### **Flex items:** The flex items are distributed along the axis specified by the flex-direction property, which also defaults to row, meaning their distribution is horizontal.

### **Alignment and Distribution:** You can distribute the space between and around flex items with the help of properties like justify-content, align-items, and align-self.

### **Flexbox layout example:**

### html

### <div class="flex-container">

### <div class="flex-item">Item 1</div>

### <div class="flex-item">Item 2</div>

### <div class="flex-item">Item 3</div>

### </div>

### 

### .flex-container {

### display: flex;

### justify-content: space-between; /\* Space items evenly \*/

### }

### 

### .flex-item {

### background-color: light gray;

### padding: 10px;

### }

### 

**Question 2**: Describe the properties justify-content, align-items, and flex-direction

used in Flexbox.

**ANS:1. flex-direction**

Description: The flex-direction property defines the main axis of the flex-container, along which the flex-items are laid out. It determines whether the flex items are arranged in a row (horizontal) or column (vertical) direction, and if the items should be reversed or not.

**Values:**

**row:** The default value. The flex-items are laid out horizontally, from left to right (in left-to-right languages).

**row-reverse:** The flex-items are laid out horizontally, but the order is reversed (from right to left).

**column:** The flex-items are laid out vertically, from top to bottom.

**column-reverse:** The flex-items are laid out vertically, but the order is reversed (from bottom to top).

**Effect:** The flex-direction property dictates the direction along which the items will be aligned and placed in the flex-container. It defines the "main axis" of the container, and the cross-axis (perpendicular to the main axis) will be used for alignment.

Example:

.container {

display: flex;

flex-direction: row;

}

### **2. justify-content**

**Description:** The justify-content property controls the alignment of the flex-items along the main axis (which is determined by flex-direction). It is used to distribute any extra space between the flex-items or adjust their positioning within the container.

**Values:**

**flex-start:** Aligns the flex-items to the start of the container along the main axis (default).

**flex-end:** Aligns the flex-items to the end of the container along the main axis.

**center:** Centers the flex-items along the main axis.

**space-between:** Distributes the flex-items evenly along the main axis with the first item at the start and the last item at the end, leaving equal space between items.

**space-around:** Distributes the flex-items evenly with equal space around each item (space between items is the same, and the space before the first and after the last item is half of the space between items).

**space-evenly:** Distributes the flex-items evenly with equal space between the items and the container edges.

Effect: This property helps control how flex-items are distributed along the main axis of the flex-container, allowing you to control their positioning and spacing.

Example:

.container {

display: flex;

justify-content: space-between; /\* Items are spaced evenly between the start and end of the container \*/

}

### **3. align-items**

**Description:** The align-items property controls the alignment of the flex-items along the cross-axis, which is perpendicular to the main axis (determined by flex-direction). It helps position the items vertically (if flex-direction is row) or horizontally (if flex-direction is column).

Values:

**stretch:** The default value. Stretches the flex-items to fill the container along the cross-axis.

**flex-start:** Aligns the flex-items to the start of the cross-axis (top for rows, left for columns).

**flex-end:** Aligns the flex-items to the end of the cross-axis (bottom for rows, right for columns).

**center:** Centers the flex-items along the cross-axis.

**baseline:** Aligns the flex-items along their baseline, which is the line where text characters sit. This can be useful if the flex-items contain text.

Effect: This property is particularly useful for vertical alignment of items when flex-direction is row, or horizontal alignment when flex-direction is column.

**CSS Grid**

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

overFlexbox?

**ANS:**CSS Grid Layout is a two-dimensional layout system, allowing you to create complex layouts using rows and columns. It allows you to design web pages by defining a grid structure where you can place items in specific grid cells or areas. Grid is particularly useful for creating layouts that require precise control over both dimensions, that is, horizontal and vertical.

**Key Features of CSS Grid:**

**Two-Dimensional Layout:** Grid can set the rows and columns at the same time, which is always handy for complex layouts.

**Grid Lines and Areas:** You can define grid lines and areas and thus have more control on items placement.

**Responsive Design:** The grid can easily fit into a responsive design with media queries and grid template areas.

**Alignment and Spacing:** Grid offers very powerful alignment and spacing features, such as align-items, justify-items, grid-gap, etc.

**Flexbox:**

Flexbox, or Flexible Box Layout, is a one-dimensional layout model designed to lay out items in a single direction- either as a row or a column. It is useful when the distributing of space along a single axis is much required, and also for aligning items within a container.

**Key Features of Flexbox**

**One-Dimensional Layout:** The focus of Flexbox is on either rows or columns but not both simultaneously.

**Flexible Item Sizes:** Flexbox makes items expand or shrink to fill the available space. This is a perfect feature for responsive designs.

**Alignment and Distribution**: Flexbox also provides properties that align items and distribute the available space between them, like justify-content, align-items, and flex-grow.

**When to Use Grid Over Flexbox**

You'll typically decide between CSS Grid vs. Flexbox in one of the following scenarios:

**Complex Layouts**: When you need a layout that requires both rows and columns, for example, full web page layout with sidebars, footers, headers, and main content areas.

**Precise Control:** When you want to have substantial control over placing elements in a grid, such as overlapping elements or specific grid areas.

**Responsive Design:** You want to create a responsive layout that responds to different screen sizes and orientations, especially when the layout changes significantly between breakpoints.

**Grid Areas:** You want to define specific areas of the layout (for example, header, sidebar, main content) and control how items span across those areas.

**Question 2**: Describe the grid-template-columns, grid-template-rows, and grid-gas properties.

Provide examples of how to use them.

**ANS:**CSS Grid Layout, the properties grid-template-columns, grid-template-rows, and grid-gap (or gap) are the building blocks of the structure and spacing of the grid. Here is a detailed description of each property along with examples of how to use them.

**1. grid-template-columns**

The grid-template-columns property declares the number and size of columns in a grid container. You can specify the size of each column using various units (like pixels, percentages, fr units, etc.).

Syntax:

grid-template-columns: <track-size> [<track-size>.];

**Example**:

.container {

display: grid;

grid-template-columns: 100px 200px 1fr; /\* 3 columns: 100px, 200px, and 1 fraction of the remaining space \*/

}

**2. grid-template-rows**

The grid-template-rows property defines the number and size of the rows in a grid container, similar to grid-template-columns.

**Syntax:**

grid-template-rows: <track-size> [<track-size>.];

**Example:**

.container {

display: grid;

grid-template-rows: 150px auto 2fr; /\* 3 rows: 150px, auto height, and 2 fractions of the remaining space \*/

}

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

The grid-gap property, in recent times popularly known as gap, helps you set row and column space for a grid. You might give it one value to spread them evenly on rows and columns, or use two values so you can differentiate which are the gap on rows or columns.

**Syntax:**

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

**Example**:

.container {

display: grid;

grid-template-columns: repeat(3, 1fr); /\* 3 equal columns

grid-template-rows: repeat(2, 100px); /\* 2 rows of 100px each \*/

gap: 10px; /\* 10px gap between rows and columns \*/

}

**Combined Example**

Below is the full example using all three:

<div class="container">

<div class="item1">Item 1</div>

<div class="item2">Item 2</div>

<div class="item3">Item 3</div>

<div class="item4">Item 4</div>

</div>

.container {

display: grid;

grid-template-columns: 100px 200px 1fr; /\* 3 columns \*/

grid-template-rows: 150px auto; /\* 2 rows \*/

gap: 15px; /\* 15px gap between rows and columns \*/

}

.item1 { background-color: lightblue; }

.item2 { background-color: light green; }

.item3 { background-color: light coral; }

.item4 { background-color: lightgoldenrodyellow; }

**Responsive Web Design with Media Queries**

**Question 1:** What are media queries in CSS, and why are they important for

responsive design?

**ANS:**Media queries are a feature of CSS that allow you to apply styles based on the characteristics of the device or viewport displaying the content. They enable you to create responsive designs that adapt to different screen sizes, orientations, resolutions, and other conditions. Media queries can be used to apply specific styles when certain conditions are met, such as the width of the viewport or the resolution of the device.

### **Why are Media Queries Important for Responsive Design?**

Media queries are crucial for responsive design for several reasons:

1. **Adaptability:** They allow web pages to adapt to various screen sizes and orientations, ensuring that content is accessible and visually appealing on devices ranging from mobile phones to large desktop monitors.
2. **Improved User Experience**: By tailoring styles to different devices, media queries help enhance the user experience. For example, larger touch targets can be provided for mobile users, and layouts can be adjusted to prevent horizontal scrolling.
3. **Performance Optimization:** Media queries can help optimize performance by loading only the necessary styles for a specific device. This can reduce the amount of CSS that needs to be processed, leading to faster load times.
4. **Maintainability:** Using media queries allows developers to maintain a single stylesheet that adapts to different devices rather than creating separate stylesheets for each device type. This makes the codebase easier to manage and reduces redundancy.
5. **Future-Proofing:** As new devices with varying screen sizes and resolutions continue to emerge, media queries provide a flexible way to ensure that web designs remain functional and attractive across all platforms.

**Question 2**: Write a basic media query that adjusts the font size of a webpage for

screens smaller than 600px

**ANS:**

**Example:**

/\* Default styles for larger screens \*/

body {

font-size: 16px; /\* Default font size \*/

}

/\* Media query for screens smaller than 600px \*/

@media screen and (max-width: 600px) {

body {

font-size: 14px; /\* Smaller font size for smaller screens \*/

}

**}**

**Explanation:**

**Default Styles:** The element body has a default font size of 16 pixels, applying to all screens larger than 600 pixels.

**Media Query:** The @media rule checks whether it is true to have a screen width of less than 600 pixels. And if that's the case, the styles between the curly braces will apply a font size of 14 pixels.

**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?

**ANS:**

**Web-Safe Fonts:**

These are fonts that are pre-installed across most operating systems (Windows, macOS, Linux, etc.). They are widely supported and available on virtually every device, making them reliable choices for web design.

Examples include Arial, Times New Roman, Courier New, and Verdana.

Because they are universally accessible, web-safe fonts ensure consistency in appearance across various platforms and browsers without additional loading time or issues.

Web-safe fonts are usually plain, system fonts that are optimized for legibility and compatibility.

**Custom Web Fonts:**

Custom web fonts are fonts not installed on a device but instead loaded from a web server. These are mostly done through services like Google Fonts, Adobe Fonts, or by including font files such as TTF or OTF.

These can be more stylized and unique and provide more leeway in branding and design.

Custom web fonts allow web designers to include special typography that is not available as a web-safe font, promoting a more unique, personal appearance for the website.

They download upon the user's browser at the time the website loads, which means this will also increase page load time by a little amount

**Why Might You Use a Web-Safe Font Over a Custom Font?**

**Compatibility:** Web-safe fonts ensure that the typography will look the same on all devices and browsers without relying on external resources. This avoids potential issues with font rendering across different platforms or the font failing to load if there's a problem with the custom font file or its service.

**Performance:** Web-safe fonts are usually pre-installed on a device, meaning one doesn't need to download additional files, and consequently, this means the page takes less time to load. For optimization in both user experience and SEO, web pages prefer quick loading.

Simplicity-Web-safe fonts are often appropriate for simple and clean, even functional designs and are widely familiar and highly readable. They will be preferred more in applications, or websites focusing on content clearness and the minimalism used.

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

Font to a webpage?

**ANS:The font-family Property in CSS:**

The font-family property in CSS is used to define the font style for text in a webpage. It specifies the font or a list of fonts that should be applied to an element or group of elements.

**How it works**:

**Font Names:** You can specify either a specific font (e.g., "Arial", "Times New Roman") or a generic font family (e.g., serif, sans-serif, monospace).

**Font Stacks:** You may specify more than one font in a comma-separated list. If the browser does not support the first, it will try the next on in the list and so on.

Generic Font Families: Broad categories of fonts representing many fonts within (serif, sans-serif, monospace).

**Example**:

body {

font-family: "Arial", sans-serif;

}

This CSS rule will use "Arial" if it's available, and if not, it will fall back to any available sans-serif font.

**How to Apply a Custom Google Font to a Webpage:**

To use a custom Google Font on your webpage, you need to include the font from Google Fonts in your HTML file and then reference it in your CSS.

**Steps to Apply a Google Font:**

Go to Google Fonts: Head over to Google Fonts and pick your desired font.

**Choose the Font:** Select your chosen font by clicking on it. You can also select a different font weight or style if you want to.

**Get the Link:** After selecting your chosen font, Google Fonts will give you a <link> tag that you can place inside the <head> section of your HTML file.

**example:**

<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">

Insert the Link in HTML: Copy and paste this <link> element into the <head> section of your HTML document.

<head>

<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">

</head>

**Use the Font in CSS:** After linking the font, use the font-family property in your CSS to apply it.

**example:**

body {

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

}