To ensure that your website displays correctly across different devices like Android, Mac, PC, HP PC, Dell PC, and laptops, you need to make it **responsive**. A responsive website adjusts its layout and design based on the screen size of the device being used. This is achieved using HTML and CSS techniques.

Here’s how you can approach building a responsive website:

**1. Use a Viewport Meta Tag (HTML)**

Add the following meta tag inside the <head> section of your HTML to ensure proper scaling on mobile devices:

html

Copy code

<meta name="viewport" content="width=device-width, initial-scale=1.0">

This ensures that the browser scales the webpage to fit the screen width of the device, and prevents zooming by default.

**2. CSS Media Queries**

Use **media queries** in CSS to apply different styles depending on the screen size. This is key to making the site responsive across all devices.

css

Copy code

/\* Base styles for desktop and larger screens \*/

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

background-color: #f0f0f0;

}

.container {

width: 80%;

margin: 0 auto;

padding: 20px;

}

/\* Styles for tablets and smaller screens (screen width <= 768px) \*/

@media (max-width: 768px) {

.container {

width: 90%;

}

}

/\* Styles for smartphones (screen width <= 480px) \*/

@media (max-width: 480px) {

.container {

width: 100%;

padding: 10px;

}

body {

font-size: 14px;

}

}

**3. Fluid Layouts (Percentages or Viewport Units)**

Instead of fixed pixel sizes for widths, use percentage-based widths, or use vw (viewport width) and vh (viewport height) units.

css

Copy code

.container {

width: 100%;

max-width: 1200px;

margin: 0 auto;

padding: 20px;

}

img {

width: 100%;

height: auto;

}

**4. Flexbox or CSS Grid for Layouts**

You can use **Flexbox** or **CSS Grid** to create flexible and responsive layouts.

**Example using Flexbox:**

css

Copy code

/\* A flexible grid layout \*/

.container {

display: flex;

flex-wrap: wrap;

justify-content: space-between;

}

.card {

flex: 1 1 300px; /\* Flex item will take up at least 300px width \*/

margin: 10px;

}

**Example using CSS Grid:**

css

Copy code

/\* Grid layout with responsive breakpoints \*/

.container {

display: grid;

grid-template-columns: repeat(4, 1fr);

gap: 20px;

}

@media (max-width: 768px) {

.container {

grid-template-columns: repeat(2, 1fr);

}

}

@media (max-width: 480px) {

.container {

grid-template-columns: 1fr;

}

}

**5. Responsive Images**

Use the srcset attribute in the <img> tag for responsive images. This allows the browser to choose the appropriate image size based on the screen resolution.

html

Copy code

<img src="image.jpg"

srcset="image-small.jpg 480w, image-medium.jpg 768w, image-large.jpg 1200w"

sizes="(max-width: 480px) 480px, (max-width: 768px) 768px, 1200px"

alt="Description">

**6. Test on Multiple Devices and Screen Sizes**

To ensure your site is responsive:

* Use **browser developer tools** to simulate different devices.
* Test on **real devices** as much as possible.
* Consider using a **responsive design framework** (like **Bootstrap** or **Tailwind CSS**) that already provides predefined responsive classes for layouts, grids, and components.

**Example with Bootstrap (optional):**

html

Copy code

<!-- Link Bootstrap CSS -->

<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">

<div class="container">

<div class="row">

<div class="col-sm-12 col-md-6 col-lg-4">

<div class="card">

<img src="image.jpg" class="card-img-top" alt="...">

<div class="card-body">

<h5 class="card-title">Card title</h5>

<p class="card-text">Some quick example text.</p>

</div>

</div>

</div>

</div>

</div>

In this example, Bootstrap’s grid system (col-sm-12, col-md-6, col-lg-4) automatically adjusts the layout on small, medium, and large screens.

**7. Typography Adjustments**

Use relative units (like em, rem, or %) for font sizes, so text scales well on different screen sizes.

css

Copy code

body {

font-size: 16px;

}

h1 {

font-size: 2rem;

}

p {

font-size: 1rem;

}

**8. Test Cross-Platform Compatibility**

Since you want the website to display well on different devices (Android, iOS, Mac, Windows PCs), test across:

* **Android devices**: Google Chrome's developer tools allow testing on mobile views.
* **Mac/Windows PCs**: Ensure compatibility with both Chrome and Safari.
* **Multiple screen sizes**: Ensure everything looks good on large monitors (like desktops and laptops) and small screens (like smartphones).

By combining these practices (viewport meta tag, CSS media queries, flexible layouts, etc.), you can build a website that adapts well to any device.

Text animation:-

To create a text animation where letters appear and disappear one by one using CSS and HTML, you can use **keyframes** in CSS to animate the letters. This effect will involve animating the opacity and position of each letter, making them appear and disappear sequentially.

Here’s how you can create such an animation:

**HTML Structure:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text Animation</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="text-container">

<span class="letter">H</span>

<span class="letter">e</span>

<span class="letter">l</span>

<span class="letter">l</span>

<span class="letter">o</span>

<span class="letter">,</span>

<span class="letter"> </span>

<span class="letter">W</span>

<span class="letter">o</span>

<span class="letter">r</span>

<span class="letter">l</span>

<span class="letter">d</span>

</div>

</body>

</html>

**CSS Animation (styles.css):**

css

Copy code

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

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

background-color: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.text-container {

display: flex;

}

.letter {

display: inline-block;

opacity: 0;

transform: translateY(20px);

animation: fadeInOut 2s ease-in-out infinite;

}

/\* Delay each letter's animation \*/

.letter:nth-child(1) {

animation-delay: 0s;

}

.letter:nth-child(2) {

animation-delay: 0.2s;

}

.letter:nth-child(3) {

animation-delay: 0.4s;

}

.letter:nth-child(4) {

animation-delay: 0.6s;

}

.letter:nth-child(5) {

animation-delay: 0.8s;

}

.letter:nth-child(6) {

animation-delay: 1s;

}

.letter:nth-child(7) {

animation-delay: 1.2s;

}

.letter:nth-child(8) {

animation-delay: 1.4s;

}

.letter:nth-child(9) {

animation-delay: 1.6s;

}

.letter:nth-child(10) {

animation-delay: 1.8s;

}

.letter:nth-child(11) {

animation-delay: 2s;

}

.letter:nth-child(12) {

animation-delay: 2.2s;

}

/\* Keyframes for the typing and removing animation \*/

@keyframes fadeInOut {

0% {

opacity: 0;

transform: translateY(20px);

}

50% {

opacity: 1;

transform: translateY(0);

}

100% {

opacity: 0;

transform: translateY(-20px);

}

}

**Explanation:**

* **HTML**: We wrap each letter in a <span> tag, which allows us to animate them individually.
* **CSS**:
  + The .letter class is used to apply the animation to each letter.
  + We use @keyframes to define the animation. The animation transitions the letter’s opacity from 0 to 1 (for typing effect) and back to 0 (for removal effect), along with a small vertical movement (translateY) to create a smoother effect.
  + The animation-delay is used to stagger the animation of each letter so that they animate one by one.
  + The infinite keyword ensures the animation keeps repeating.

**Result:**

When you run this code, you’ll see the text "Hello, World" where each letter appears one by one and then disappears in sequence, repeating indefinitely.

Zoom:-

To create a **3vh zoom-in effect** on a circular image using CSS, you can apply a hover effect that will scale the image. The zoom effect will be achieved by using the transform property with scale() for zooming and transition for smooth easing.

Here's how to create this effect:

**HTML Structure:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Zoom Effect on Circular Image</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="image-container">

<img src="your-image.jpg" alt="Circular Image" class="zoom-image">

</div>

</body>

</html>

**CSS (styles.css):**

css

Copy code

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.image-container {

overflow: hidden; /\* Hides the part of the image that goes outside the circle \*/

width: 200px; /\* Adjust based on your preferred size \*/

height: 200px; /\* Adjust based on your preferred size \*/

border-radius: 50%; /\* Creates a circular shape \*/

position: relative;

}

.zoom-image {

width: 100%;

height: 100%;

object-fit: cover; /\* Ensures the image covers the container without distorting \*/

transition: transform 0.3s ease; /\* Smooth transition for zoom effect \*/

}

.image-container:hover .zoom-image {

transform: scale(1.03); /\* Zooms in by 3vh (approx 3%) \*/

}

**Explanation:**

1. **HTML**:
   * The image is wrapped inside a div container, .image-container, which will be responsible for keeping the circular shape.
2. **CSS**:
   * .image-container:
     + overflow: hidden ensures that any part of the image outside the circular container is not visible.
     + width and height: These are set to define the size of the circular image (you can adjust these values as needed).
     + border-radius: 50%: This makes the image container circular.
   * .zoom-image:
     + width: 100% and height: 100% make the image fill the entire container.
     + object-fit: cover: Ensures that the image covers the container without stretching or distorting, even if it needs to crop some of the image.
     + transition: transform 0.3s ease: This makes the zoom effect smooth when hovering over the image.
   * .image-container:hover .zoom-image:
     + The transform: scale(1.03) zooms in the image by 3% when you hover over the .image-container. This will create a zoom effect that enlarges the image slightly. The value 1.03 represents a 3% increase (you can adjust the value to make the zoom more or less intense).

**Result:**

When you hover over the circular image, it will smoothly zoom in by 3vh (approximately 3%), and when you remove the hover, it will return to its original size.

All side shadow:-

To create a **shadow on all sides of a circle** using CSS and HTML, you can use the box-shadow property in CSS. This property allows you to add shadows to elements, and you can adjust the spread, blur radius, color, and offset of the shadow to get the desired effect.

Here's how to create a circular image with a shadow on all sides:

**HTML Structure:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Circle Shadow Effect</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="circle">

<img src="your-image.jpg" alt="Circular Image" class="circle-image">

</div>

</body>

</html>

**CSS (styles.css):**

css

Copy code

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.circle {

width: 200px; /\* Adjust based on your preferred size \*/

height: 200px; /\* Adjust based on your preferred size \*/

border-radius: 50%; /\* Makes the element circular \*/

overflow: hidden; /\* Ensures the image doesn't spill out of the circle \*/

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.4), 0 6px 20px rgba(0, 0, 0, 0.3); /\* Shadow on all sides \*/

}

.circle-image {

width: 100%;

height: 100%;

object-fit: cover; /\* Ensures the image fits within the circle without stretching \*/

}

**Explanation:**

1. **HTML**:
   * The image is wrapped inside a div container with the class .circle. This div will have a circular shape, and we’ll apply the shadow to it.
2. **CSS**:
   * .circle:
     + width and height: These define the size of the circle (you can adjust them based on your preferred size).
     + border-radius: 50%: This makes the div element circular.
     + overflow: hidden: Ensures that any part of the image extending beyond the circle will be clipped and not visible.
     + box-shadow: This creates the shadow around the circle. The property box-shadow takes multiple values:
       - The first 0 is the horizontal offset (shadow is centered).
       - The second 4px is the vertical offset (you can adjust this for the desired shadow position).
       - The third 8px is the blur radius (controls how soft or hard the shadow is).
       - The fourth rgba(0, 0, 0, 0.4) is the color of the shadow with alpha transparency.
       - The second shadow value 0 6px 20px rgba(0, 0, 0, 0.3) adds a second shadow to make it more pronounced on all sides of the circle.
   * .circle-image:
     + width: 100% and height: 100%: These ensure the image fills the entire circle.
     + object-fit: cover: Ensures the image fits properly inside the circular shape without distortion or stretching, though parts of it may be cropped.

**Result:**

This will create a circular image with a shadow effect on all sides, making it look like it's slightly elevated from the background. You can adjust the shadow’s size, blur, and opacity by modifying the values of box-shadow to achieve the effect you desire.

Shapes :-

create diffrent shapes using html css

https://t3.gstatic.com/faviconV2?client=SOCIAL&type=FAVICON&fallback_opts=TYPE,SIZE,URL&url=https://www.w3schools.com/howto/howto_css_shapes.asp

## How To Create Different Shapes with CSS - W3Schools

https://www.w3schools.com/howto/howto\_css\_shapes.a...

## How To Create Different Shapes with CSS - W3Schools

Well organized and easy to understand Web building tutorials with lots of examples of how to use HTML, CSS, JavaScript, SQL, Python, PHP, Bootstrap, Java, ...

* Rectangle CSS
* Trapezoid CSS
* Parallelogram CSS
* Triangle Right CSS

**BLACKBOXAI**

**How To Create Different Shapes with CSS - W3Schools**

https://t3.gstatic.com/faviconV2?client=SOCIAL&type=FAVICON&fallback_opts=TYPE,SIZE,URL&url=https://www.w3schools.com/howto/howto_css_shapes.asp

w3schools

**The Shapes Of CSS**

https://t3.gstatic.com/faviconV2?client=SOCIAL&type=FAVICON&fallback_opts=TYPE,SIZE,URL&url=https://css-tricks.com/the-shapes-of-css/

css-tricks

**Creating basic and more advanced shapes in CSS - Albert Walicki**

https://t3.gstatic.com/faviconV2?client=SOCIAL&type=FAVICON&fallback_opts=TYPE,SIZE,URL&url=https://albertwalicki.com/blog/creating-shapes-in-css

albertwalicki

**Shapes in CSS - Alvaro Montoro**

https://t3.gstatic.com/faviconV2?client=SOCIAL&type=FAVICON&fallback_opts=TYPE,SIZE,URL&url=https://alvaromontoro.com/blog/68042/shapes-in-css

alvaromontoro

Creating different shapes using HTML and CSS can be achieved through various techniques. You can start with basic shapes like squares and rectangles, which are straightforward since they are the default shapes in HTML. For circles, you can use border-radius to round the corners of a square.

To create more complex shapes like triangles , polygons, and more, you can utilize CSS properties such as **clip-path** or manipulate borders. Here’s a breakdown of how to create some common shapes:

**1. Square and Rectangle**

* **HTML**: Use a **<div>** element.
* **CSS**: Set the width and height.

html

VerifyCopy code

1<div class="square"></div>

2<div class="rectangle"></div>

css

VerifyCopy code

1.square {

2 width: 100px;

3 height: 100px;

4 background-color: blue;

5}

6

7.rectangle {

8 width: 200px;

9 height: 100px;

10 background-color: red;

11}

**2. Circle**

* **HTML**: Use a **<div>** element.
* **CSS**: Set the width and height equal and apply **border-radius**.

html

VerifyCopy code

1<div class="circle"></div>

css

VerifyCopy code

1.circle {

2 width: 100px;

3 height: 100px;

4 background-color: green;

5 border-radius: 50%;

6}

**3. Triangle**

* **HTML**: Use a **<div>** element.
* **CSS**: Use borders to create a triangle effect.

html

VerifyCopy code

1<div class="triangle"></div>

css

VerifyCopy code

1.triangle {

2 width: 0;

3 height: 0;

4 border-left: 50px solid transparent;

5 border-right: 50px solid transparent;

6 border-bottom: 100px solid orange;

7}

**4. Ellipse**

* **HTML**: Use a **<div>** element.
* **CSS**: Set width and height with **border-radius**.

html

VerifyCopy code

1<div class="ellipse"></div>

css

VerifyCopy code

1.ellipse {

2 width: 200px;

3 height: 100px;

4 background-color: purple;

5 border-radius: 100px / 50px;

6}

**5. Polygon**

* **HTML**: Use a **<div>** element.
* **CSS**: Use **clip-path** to create a polygon shape.

html

VerifyCopy code

1<div class="polygon"></div>

css

VerifyCopy code

1.polygon {

2 width: 100px;

3 height: 100px;

4 background-color: yellow;

5 clip-path: polygon(50% 0%, 100% 100%, 0% 100%);

6}

These examples provide a foundation for creating various shapes using HTML and CSS. You can further customize the colors, sizes, and other properties to fit your design needs.

Animations :-

To create beautiful animations using CSS @keyframes, you can make use of various effects like fades, slides, rotations, and scaling. Below are a few examples of animations you can create with keyframes that can add visual interest and smooth transitions to elements on your web page.

**1. Fading and Scaling Animation**

This animation fades in the element while scaling it up from 0 to 1.

**HTML:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Fading and Scaling Animation</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="fade-scale-box">Welcome!</div>

</body>

</html>

**CSS (styles.css):**

css

Copy code

body {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

background-color: #f4f4f4;

}

.fade-scale-box {

width: 200px;

height: 200px;

background-color: #3498db;

color: white;

display: flex;

justify-content: center;

align-items: center;

font-size: 24px;

font-weight: bold;

border-radius: 10px;

animation: fadeScale 3s ease-out;

}

@keyframes fadeScale {

0% {

opacity: 0;

transform: scale(0.5);

}

100% {

opacity: 1;

transform: scale(1);

}

}

**Explanation:**

* The @keyframes fadeScale animation gradually changes the opacity from 0 to 1 while scaling the element from 0.5 to 1, giving the effect of fading and zooming in.
* The animation property is applied to the .fade-scale-box element, which causes the animation to run on load.

**2. Sliding In from the Left**

This animation slides an element in from the left side of the screen.

**HTML:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Slide In Animation</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="slide-box">Slide In!</div>

</body>

</html>

**CSS (styles.css):**

css

Copy code

body {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

background-color: #f0f0f0;

}

.slide-box {

width: 250px;

height: 150px;

background-color: #e74c3c;

color: white;

display: flex;

justify-content: center;

align-items: center;

font-size: 20px;

font-weight: bold;

border-radius: 10px;

position: relative;

animation: slideIn 2s ease-out;

}

@keyframes slideIn {

0% {

transform: translateX(-100%);

}

100% {

transform: translateX(0);

}

}

**Explanation:**

* The @keyframes slideIn animation moves the element from -100% on the X-axis (left off the screen) to 0 (its original position).
* The animation property is applied to the .slide-box class, so the animation occurs on page load.

**3. Bouncing Animation**

This animation causes an element to bounce up and down, creating a fun and dynamic effect.

**HTML:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Bouncing Animation</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="bounce-box">Bounce!</div>

</body>

</html>

**CSS (styles.css):**

css

Copy code

body {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

background-color: #ecf0f1;

}

.bounce-box {

width: 150px;

height: 150px;

background-color: #2ecc71;

color: white;

display: flex;

justify-content: center;

align-items: center;

font-size: 24px;

font-weight: bold;

border-radius: 50%;

animation: bounce 2s infinite ease-in-out;

}

@keyframes bounce {

0%, 20%, 50%, 80%, 100% {

transform: translateY(0);

}

40% {

transform: translateY(-30px);

}

60% {

transform: translateY(-15px);

}

}

**Explanation:**

* The @keyframes bounce animation makes the element "bounce" up and down by altering its translateY value.
* It starts by moving the element up by -30px and then settles back to its original position, creating the bounce effect.
* The animation is set to repeat infinitely (infinite) with smooth timing.

**4. Color Changing Background Animation**

This animation changes the background color of an element continuously, creating a glowing effect.

**HTML:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Color Changing Animation</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="color-box">Color Change!</div>

</body>

</html>

**CSS (styles.css):**

css

Copy code

body {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

background-color: #34495e;

}

.color-box {

width: 200px;

height: 200px;

background-color: #3498db;

color: white;

display: flex;

justify-content: center;

align-items: center;

font-size: 20px;

font-weight: bold;

border-radius: 10px;

animation: colorChange 5s infinite;

}

@keyframes colorChange {

0% {

background-color: #3498db;

}

25% {

background-color: #9b59b6;

}

50% {

background-color: #2ecc71;

}

75% {

background-color: #f39c12;

}

100% {

background-color: #3498db;

}

}

**Explanation:**

* The @keyframes colorChange animation smoothly transitions the background color of the .color-box between multiple colors over 5 seconds.
* The animation property is set to run infinitely, creating a continuous color-changing effect.

**Conclusion:**

These are just a few examples of animations using CSS @keyframes. You can create a wide variety of animations, such as fading, sliding, bouncing, rotating, or transforming an element, to add visual interest to your web page. Experiment with different properties and timings to achieve the desired effect!

Top of Form



Bottom of Form

Data storage:-

To store data in a table and display it later, you can use **HTML**, **CSS**, and **JavaScript**. However, since HTML alone cannot save data persistently (across page reloads), you can use **JavaScript** to store data in **localStorage** or **sessionStorage** to keep the data even after the page reloads.

Here’s a simple example where:

1. You can input data (e.g., Name, Age, etc.).
2. The data gets saved in a table.
3. The saved data is retrieved and displayed even after the page reloads using **localStorage**.

**HTML Structure:**

* A form to input the data.
* A table to display the saved data.
* Buttons to save and show the data.

**CSS (For Styling):**

A simple table and form design.

**JavaScript:**

For saving the data into localStorage and retrieving it later.

**Full Code Example:**

**HTML:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Data Storage and Display</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2>Store and Display Data</h2>

<!-- Form to input data -->

<form id="dataForm">

<input type="text" id="name" placeholder="Enter Name" required>

<input type="number" id="age" placeholder="Enter Age" required>

<button type="submit">Save Data</button>

</form>

<h3>Saved Data:</h3>

<!-- Table to display saved data -->

<table id="dataTable">

<thead>

<tr>

<th>Name</th>

<th>Age</th>

</tr>

</thead>

<tbody>

<!-- Rows will be dynamically added here -->

</tbody>

</table>

<button id="loadData">Load Saved Data</button>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

css

Copy code

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

width: 80%;

max-width: 600px;

background-color: #fff;

padding: 20px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

border-radius: 8px;

}

h2 {

text-align: center;

}

form {

display: flex;

flex-direction: column;

gap: 10px;

margin-bottom: 20px;

}

input {

padding: 10px;

font-size: 16px;

border-radius: 4px;

border: 1px solid #ccc;

}

button {

padding: 10px;

background-color: #3498db;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #2980b9;

}

table {

width: 100%;

border-collapse: collapse;

margin-bottom: 20px;

}

th, td {

padding: 10px;

text-align: center;

border: 1px solid #ccc;

}

th {

background-color: #3498db;

color: white;

}

**JavaScript (script.js):**

javascript

Copy code

// Wait for the document to be fully loaded

document.addEventListener('DOMContentLoaded', () => {

const form = document.getElementById('dataForm');

const nameInput = document.getElementById('name');

const ageInput = document.getElementById('age');

const dataTable = document.getElementById('dataTable').getElementsByTagName('tbody')[0];

const loadDataButton = document.getElementById('loadData');

// Function to load data from localStorage and display it in the table

const loadData = () => {

// Clear any existing rows in the table

dataTable.innerHTML = '';

// Get the saved data from localStorage (if any)

const savedData = JSON.parse(localStorage.getItem('data')) || [];

// Populate the table with saved data

savedData.forEach((data) => {

const row = dataTable.insertRow();

const nameCell = row.insertCell(0);

const ageCell = row.insertCell(1);

nameCell.textContent = data.name;

ageCell.textContent = data.age;

});

};

// Event listener to handle the form submission

form.addEventListener('submit', (event) => {

event.preventDefault(); // Prevent the form from reloading the page

// Get the entered values

const name = nameInput.value.trim();

const age = ageInput.value.trim();

// Check if the values are valid

if (name && age) {

// Get the existing data from localStorage (or an empty array if none exists)

const savedData = JSON.parse(localStorage.getItem('data')) || [];

// Add the new entry to the array

savedData.push({ name, age });

// Save the updated data back to localStorage

localStorage.setItem('data', JSON.stringify(savedData));

// Clear the form inputs

nameInput.value = '';

ageInput.value = '';

// Reload the data into the table

loadData();

} else {

alert('Please enter both name and age!');

}

});

// Load data when the page is loaded

loadData();

// Event listener for loading data button

loadDataButton.addEventListener('click', loadData);

});

**Explanation:**

1. **HTML**:
   * The form allows the user to input data (Name and Age).
   * A table is provided to display the saved data.
   * The "Load Saved Data" button can be clicked to manually load data from localStorage.
2. **CSS**:
   * Basic styling is applied to the form, buttons, and table for better presentation.
3. **JavaScript**:
   * The localStorage API is used to store data in the browser. localStorage allows data to persist across page reloads.
   * The form's submit event is handled to prevent the page from reloading and to save the data entered in the form.
   * The loadData() function retrieves the saved data from localStorage, parses it into an array, and dynamically updates the table.

**Key JavaScript Methods:**

* **localStorage.setItem('key', value)**: Stores data in localStorage.
* **localStorage.getItem('key')**: Retrieves data from localStorage.
* **JSON.stringify()**: Converts a JavaScript object into a string to store in localStorage.
* **JSON.parse()**: Converts a string back into a JavaScript object.

**How It Works:**

1. When the user enters data in the form and clicks "Save Data", the data is stored in the browser’s localStorage.
2. The data is saved in a key called 'data' and is stored as a JSON array of objects.
3. The loadData() function reads this data from localStorage and populates the table.
4. If the page is reloaded or the user clicks "Load Saved Data", the table is populated with the saved data.

**Conclusion:**

This simple app allows you to input, save, and display data in a table, all using **HTML**, **CSS**, and **JavaScript**. The use of localStorage makes sure the data persists across page reloads.

Storage:-

To store data in a table and display it later, you can use **HTML**, **CSS**, and **JavaScript**. However, since HTML alone cannot save data persistently (across page reloads), you can use **JavaScript** to store data in **localStorage** or **sessionStorage** to keep the data even after the page reloads.

Here’s a simple example where:

1. You can input data (e.g., Name, Age, etc.).
2. The data gets saved in a table.
3. The saved data is retrieved and displayed even after the page reloads using **localStorage**.

**HTML Structure:**

* A form to input the data.
* A table to display the saved data.
* Buttons to save and show the data.

**CSS (For Styling):**

A simple table and form design.

**JavaScript:**

For saving the data into localStorage and retrieving it later.

**Full Code Example:**

**HTML:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Data Storage and Display</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2>Store and Display Data</h2>

<!-- Form to input data -->

<form id="dataForm">

<input type="text" id="name" placeholder="Enter Name" required>

<input type="number" id="age" placeholder="Enter Age" required>

<button type="submit">Save Data</button>

</form>

<h3>Saved Data:</h3>

<!-- Table to display saved data -->

<table id="dataTable">

<thead>

<tr>

<th>Name</th>

<th>Age</th>

</tr>

</thead>

<tbody>

<!-- Rows will be dynamically added here -->

</tbody>

</table>

<button id="loadData">Load Saved Data</button>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

css

Copy code

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

width: 80%;

max-width: 600px;

background-color: #fff;

padding: 20px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

border-radius: 8px;

}

h2 {

text-align: center;

}

form {

display: flex;

flex-direction: column;

gap: 10px;

margin-bottom: 20px;

}

input {

padding: 10px;

font-size: 16px;

border-radius: 4px;

border: 1px solid #ccc;

}

button {

padding: 10px;

background-color: #3498db;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #2980b9;

}

table {

width: 100%;

border-collapse: collapse;

margin-bottom: 20px;

}

th, td {

padding: 10px;

text-align: center;

border: 1px solid #ccc;

}

th {

background-color: #3498db;

color: white;

}

**JavaScript (script.js):**

javascript

Copy code

// Wait for the document to be fully loaded

document.addEventListener('DOMContentLoaded', () => {

const form = document.getElementById('dataForm');

const nameInput = document.getElementById('name');

const ageInput = document.getElementById('age');

const dataTable = document.getElementById('dataTable').getElementsByTagName('tbody')[0];

const loadDataButton = document.getElementById('loadData');

// Function to load data from localStorage and display it in the table

const loadData = () => {

// Clear any existing rows in the table

dataTable.innerHTML = '';

// Get the saved data from localStorage (if any)

const savedData = JSON.parse(localStorage.getItem('data')) || [];

// Populate the table with saved data

savedData.forEach((data) => {

const row = dataTable.insertRow();

const nameCell = row.insertCell(0);

const ageCell = row.insertCell(1);

nameCell.textContent = data.name;

ageCell.textContent = data.age;

});

};

// Event listener to handle the form submission

form.addEventListener('submit', (event) => {

event.preventDefault(); // Prevent the form from reloading the page

// Get the entered values

const name = nameInput.value.trim();

const age = ageInput.value.trim();

// Check if the values are valid

if (name && age) {

// Get the existing data from localStorage (or an empty array if none exists)

const savedData = JSON.parse(localStorage.getItem('data')) || [];

// Add the new entry to the array

savedData.push({ name, age });

// Save the updated data back to localStorage

localStorage.setItem('data', JSON.stringify(savedData));

// Clear the form inputs

nameInput.value = '';

ageInput.value = '';

// Reload the data into the table

loadData();

} else {

alert('Please enter both name and age!');

}

});

// Load data when the page is loaded

loadData();

// Event listener for loading data button

loadDataButton.addEventListener('click', loadData);

});

**Explanation:**

1. **HTML**:
   * The form allows the user to input data (Name and Age).
   * A table is provided to display the saved data.
   * The "Load Saved Data" button can be clicked to manually load data from localStorage.
2. **CSS**:
   * Basic styling is applied to the form, buttons, and table for better presentation.
3. **JavaScript**:
   * The localStorage API is used to store data in the browser. localStorage allows data to persist across page reloads.
   * The form's submit event is handled to prevent the page from reloading and to save the data entered in the form.
   * The loadData() function retrieves the saved data from localStorage, parses it into an array, and dynamically updates the table.

**Key JavaScript Methods:**

* **localStorage.setItem('key', value)**: Stores data in localStorage.
* **localStorage.getItem('key')**: Retrieves data from localStorage.
* **JSON.stringify()**: Converts a JavaScript object into a string to store in localStorage.
* **JSON.parse()**: Converts a string back into a JavaScript object.

**How It Works:**

1. When the user enters data in the form and clicks "Save Data", the data is stored in the browser’s localStorage.
2. The data is saved in a key called 'data' and is stored as a JSON array of objects.
3. The loadData() function reads this data from localStorage and populates the table.
4. If the page is reloaded or the user clicks "Load Saved Data", the table is populated with the saved data.

**Conclusion:**

This simple app allows you to input, save, and display data in a table, all using **HTML**, **CSS**, and **JavaScript**. The use of localStorage makes sure the data persists across page reloads.

Text shadows:-

Text shadow is a CSS property used to add shadow effects to text. It allows you to give text a subtle or dramatic 3D appearance or emphasize it with different colors and blurring effects. The text-shadow property in CSS accepts multiple values that define the shadow's position, blur radius, and color.

**Syntax of text-shadow:**

css

Copy code

text-shadow: horizontal-offset vertical-offset blur-radius color;

* **horizontal-offset**: The horizontal distance of the shadow (positive for shadow to the right, negative for shadow to the left).
* **vertical-offset**: The vertical distance of the shadow (positive for shadow below the text, negative for shadow above).
* **blur-radius** (optional): Defines how blurred the shadow is. The higher the value, the more blurred the shadow.
* **color** (optional): The color of the shadow. You can use any valid color format (e.g., rgb(), rgba(), hex, hsl()).

**Examples of Text Shadow:**

**1. Simple Text Shadow**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text Shadow Example</title>

<style>

.simple-shadow {

font-size: 48px;

color: #3498db;

text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.5); /\* Horizontal, Vertical, Blur, Color \*/

}

</style>

</head>

<body>

<h1 class="simple-shadow">Simple Text Shadow</h1>

</body>

</html>

* In this example, the shadow is moved 2px to the right and 2px down from the text with a blur radius of 5px. The color of the shadow is a semi-transparent black (rgba(0, 0, 0, 0.5)).

**2. Multiple Shadows**

You can add multiple shadows to the text by separating them with commas.

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Multiple Text Shadows</title>

<style>

.multiple-shadows {

font-size: 60px;

color: #e74c3c;

text-shadow: 3px 3px 2px rgba(0, 0, 0, 0.7), -3px -3px 2px rgba(0, 0, 0, 0.4);

}

</style>

</head>

<body>

<h1 class="multiple-shadows">Multiple Shadows</h1>

</body>

</html>

* In this example, there are two shadows: one positioned 3px to the right and 3px down, with a dark shadow color, and another shadow positioned in the opposite direction (3px left, 3px up) with a slightly lighter shadow color.

**3. Text Shadow with No Blur (Sharp Shadow)**

If you want a sharp shadow without any blur, set the blur radius to 0.

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sharp Text Shadow</title>

<style>

.sharp-shadow {

font-size: 60px;

color: #9b59b6;

text-shadow: 4px 4px 0 rgba(0, 0, 0, 0.8); /\* No blur (0) \*/

}

</style>

</head>

<body>

<h1 class="sharp-shadow">Sharp Text Shadow</h1>

</body>

</html>

* In this case, the shadow is not blurred at all (0 blur radius), creating a sharp and clear shadow effect.

**4. Glow Effect on Text**

You can create a glowing effect by using a bright color and a larger blur radius.

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Glowing Text Shadow</title>

<style>

.glow-shadow {

font-size: 60px;

color: #f39c12;

text-shadow: 0 0 10px rgba(255, 165, 0, 0.7), 0 0 20px rgba(255, 165, 0, 0.5);

}

</style>

</head>

<body>

<h1 class="glow-shadow">Glowing Text</h1>

</body>

</html>

* This example uses an orange color with a glowing effect, where the shadow is spread over a larger area with increased blur, creating a soft, glowing look.

**5. Text Shadow with Inset Effect**

If you want an inset shadow (a shadow that appears inside the text), you can simulate it by using a negative text-shadow:

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Inset Text Shadow</title>

<style>

.inset-shadow {

font-size: 60px;

color: #ecf0f1;

text-shadow: inset 1px 1px 2px rgba(0, 0, 0, 0.8);

}

</style>

</head>

<body>

<h1 class="inset-shadow">Inset Text Shadow</h1>

</body>

</html>

* The inset keyword creates an inward shadow effect, making the text look like it's carved out from a darker background.

**Summary of text-shadow Properties:**

1. **Horizontal Offset**: Controls the horizontal position of the shadow.
2. **Vertical Offset**: Controls the vertical position of the shadow.
3. **Blur Radius**: Controls how blurry the shadow will be. If set to 0, the shadow will be sharp.
4. **Color**: The color of the shadow, which can be any valid CSS color format (hex, rgb, rgba, etc.).

With text-shadow, you can create various effects such as glowing text, 3D effects, and even text with multiple shadows for more visual depth!

Box shadow:-

**Box shadow** is a CSS property used to add shadow effects around an element's box. It helps to create depth and dimension on your web page by making elements appear raised or floating above the background.

**Syntax of box-shadow:**

css

Copy code

box-shadow: horizontal-offset vertical-offset blur-radius spread-radius color inset;

* **horizontal-offset**: Required. The horizontal position of the shadow (positive values move it right, negative values move it left).
* **vertical-offset**: Required. The vertical position of the shadow (positive values move it down, negative values move it up).
* **blur-radius** (optional): Defines how blurry the shadow is. The higher the value, the more blurred the shadow.
* **spread-radius** (optional): Defines the size of the shadow. Positive values make the shadow bigger, negative values make it smaller.
* **color** (optional): Specifies the color of the shadow.
* **inset** (optional): If specified, the shadow will be inside the element (creating an inset effect).

**Examples of Box Shadow Effects:**

**1. Basic Box Shadow**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Basic Box Shadow</title>

<style>

.basic-shadow {

width: 200px;

height: 200px;

background-color: #3498db;

box-shadow: 10px 10px 15px rgba(0, 0, 0, 0.3);

}

</style>

</head>

<body>

<div class="basic-shadow"></div>

</body>

</html>

* **Explanation**: The shadow is 10px to the right and 10px down from the box, with a blur radius of 15px. The color of the shadow is a semi-transparent black (rgba(0, 0, 0, 0.3)).

**2. Multiple Shadows**

You can apply multiple shadows by separating them with commas.

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Multiple Box Shadows</title>

<style>

.multiple-shadows {

width: 250px;

height: 250px;

background-color: #2ecc71;

box-shadow: 10px 10px 20px rgba(0, 0, 0, 0.5), -10px -10px 20px rgba(0, 0, 0, 0.2);

}

</style>

</head>

<body>

<div class="multiple-shadows"></div>

</body>

</html>

* **Explanation**: There are two shadows applied: one shadow to the bottom-right and another to the top-left.

**3. Inset Box Shadow**

By using the inset keyword, the shadow appears inside the element instead of outside.

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Inset Box Shadow</title>

<style>

.inset-shadow {

width: 200px;

height: 200px;

background-color: #f39c12;

box-shadow: inset 5px 5px 15px rgba(0, 0, 0, 0.4);

}

</style>

</head>

<body>

<div class="inset-shadow"></div>

</body>

</html>

* **Explanation**: The inset keyword causes the shadow to appear inside the element, giving it a "pressed" effect.

**4. Box Shadow with Spread Radius**

The spread radius controls how much larger or smaller the shadow will be relative to the element.

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Box Shadow with Spread Radius</title>

<style>

.spread-radius {

width: 300px;

height: 300px;

background-color: #e74c3c;

box-shadow: 10px 10px 20px 10px rgba(0, 0, 0, 0.6);

}

</style>

</head>

<body>

<div class="spread-radius"></div>

</body>

</html>

* **Explanation**: The spread radius (10px) expands the shadow outward from the element, making the shadow appear larger.

**5. Soft Glow Effect**

A soft glow effect can be achieved by using a large blur radius and a bright color.

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Glow Box Shadow</title>

<style>

.glow-shadow {

width: 250px;

height: 250px;

background-color: #9b59b6;

box-shadow: 0 0 15px 5px rgba(255, 255, 255, 0.8);

}

</style>

</head>

<body>

<div class="glow-shadow"></div>

</body>

</html>

* **Explanation**: A soft, white glow effect is created by using a blur radius of 15px and a spread radius of 5px.

**6. Box Shadow with Color and Transparency**

Using RGBA colors can give you more control over the shadow’s transparency.

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Transparent Box Shadow</title>

<style>

.transparent-shadow {

width: 250px;

height: 250px;

background-color: #34495e;

box-shadow: 10px 10px 20px rgba(0, 0, 0, 0.4), 0 0 15px rgba(255, 255, 255, 0.3);

}

</style>

</head>

<body>

<div class="transparent-shadow"></div>

</body>

</html>

* **Explanation**: A shadow with a semi-transparent black color is applied, along with a subtle white glow to enhance the appearance.

**Summary of Box Shadow Properties:**

1. **horizontal-offset**: Defines how far to the left or right the shadow should be. Positive values move the shadow right; negative values move it left.
2. **vertical-offset**: Defines how far above or below the box the shadow should be. Positive values move the shadow down; negative values move it up.
3. **blur-radius**: Defines how blurry the shadow should be. A higher value results in a softer, more diffused shadow.
4. **spread-radius**: Controls the size of the shadow. Positive values expand the shadow, and negative values shrink it.
5. **color**: The color of the shadow, which can be any valid CSS color format (hex, rgba, etc.).
6. **inset** (optional): If used, it places the shadow inside the box, giving the element a pressed or inset look.

**Conclusion:**

With box-shadow, you can easily create a variety of effects such as depth, glowing, inset, and multiple shadows. By experimenting with the different properties (horizontal offset, vertical offset, blur radius, spread radius, color, and inset), you can create many creative and visually interesting designs for your web elements.

Image stylishing:-

To create an image that displays as a circle and shows the full-size image when clicked, you can achieve this with HTML and CSS. Here's how you can do it:

**Steps:**

1. **Create a circular image**: This is done using border-radius: 50% in CSS to make the image round.
2. **Show full-size image on click**: We'll use the :checked pseudo-class and a hidden checkbox to toggle between the small circular image and the full-size image.

**Example Code:**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Circle Image Click to View Full</title>

<style>

/\* Style for the full-size image \*/

.image-container {

position: relative;

width: 300px;

height: 300px;

}

.full-image {

display: none;

width: 100%;

height: auto;

position: absolute;

top: 0;

left: 0;

z-index: 10;

cursor: pointer;

}

/\* Style for the circle image \*/

.circle-image {

width: 100%;

height: 100%;

border-radius: 50%;

object-fit: cover;

cursor: pointer;

}

/\* Checkbox to toggle the image visibility \*/

input[type="checkbox"] {

display: none;

}

/\* When checkbox is checked, show full-size image \*/

input[type="checkbox"]:checked + .image-container .full-image {

display: block;

}

/\* When checkbox is checked, hide the circle image \*/

input[type="checkbox"]:checked + .image-container .circle-image {

display: none;

}

</style>

</head>

<body>

<!-- Checkbox to toggle the full-size image display -->

<input type="checkbox" id="toggle-image">

<div class="image-container">

<!-- Full-size image -->

<img src="https://via.placeholder.com/600" alt="Full Image" class="full-image" id="fullImage">

<!-- Circular image (thumbnail) -->

<label for="toggle-image">

<img src="https://via.placeholder.com/150" alt="Circle Image" class="circle-image" id="circleImage">

</label>

</div>

</body>

</html>

**Explanation:**

1. **HTML Structure**:
   * We have an input checkbox (<input type="checkbox" id="toggle-image">) that will be used to toggle between the circular and full-size image.
   * The <div class="image-container"> holds both the full-size image (.full-image) and the circular image (.circle-image).
2. **CSS Styling**:
   * The .circle-image class applies border-radius: 50% to make the image appear circular. The image is also set to object-fit: cover to ensure it fits well inside the circle.
   * The .full-image class ensures the full-size image is hidden by default (display: none).
   * When the checkbox is checked (input[type="checkbox"]:checked), we display the full-size image and hide the circular image using display: block and display: none.
3. **Interaction**:
   * When the user clicks on the circular image, the checkbox is checked, which triggers the full-size image to appear.
   * Clicking the full-size image will hide it and show the circular image again.

**How It Works:**

* Initially, only the circular image is displayed.
* When the user clicks on the circular image (which is wrapped inside a <label> element that targets the checkbox), the checkbox is checked, and the full-size image becomes visible while the circular image disappears.
* This interaction is handled purely with CSS and HTML without the need for JavaScript.

**Result:**

You will see a small circular image, and when you click it, the full-size version of the image will appear, replacing the circular image. Clicking again will toggle back to the circular image.

Top of Form



Bottom of Form