



Interview Guide

FRONTEND CRACKED

HTML | CSS | JS | REACT JS | GITHUB

SYED SHAAZ AKHTAR



Starting with HTML

"Aaj hum ek nayi duniya mein entry kar rahe hain – HTML ki duniya! Yeh ek foundation hai jo websites ki duniya ko banata hai aur chalata hai. Ready ho sab? Let's start this exciting journey!" 🚀

What is HTML?

HTML (HyperText Markup Language) is the backbone of all websites.

- **HyperText:** Socho, ek magical book jisme har page ek doosre se connected hai. Links ki madad se ek page se doosre page par jaane ko kehte hain HyperText.
- **Markup Language:** Matlab, content ko structure dena – jaise ek shaadi ke pandal mein tables aur decorations set karte ho.

Analogy:

"HTML ek film script ki tarah hai jo director (browser) ko batata hai ki kya dikhana hai aur kaise dikhana hai."

Without HTML, the browser is clueless. Jaise bina script ke actors stage par confuse ho jayein. 😊

Setting Up HTML in VS Code

Step 1: Install VS Code

Sabse pehle, ek achha code editor chahiye. Download Visual Studio Code – it's like your notebook for coding.

Step 2: Create Your First File

- File > New File, save it as index.html.
- **Why index.html?** "Index" ek default naam hai jo browser sabse pehle search karta hai.

Step 3: Write Your First Code

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome to HTML</title>
</head>
<body>
  <h1>Hello World!</h1>
  <p>This is my first HTML page.</p>
</body>
</html>
```

- Yeh code likhne ke baad "Live Server" extension se run karo aur magic dekho – ek web page aapke saamne! ✨

Understanding the Structure of HTML

HTML ek blueprint hai jo har element ka role define karta hai.

Let's break it down:

1. **DOCTYPE:**

- Yeh ek declaration hai jo browser ko batata hai ki kaunsa version use ho raha hai. For example: `<!DOCTYPE html>`.
- Without this, browser thoda confuse ho sakta hai – jaise bina rules ke cricket match.

1. **HTML Tag (<html>):**

- Pura document isi ke andar likha jaata hai.

1. **Head Tag (<head>):**

- Meta-information rakhta hai jo browser ke kaam ka hai, but users ko nahi dikhta.
- Examples: Title, CSS, Fonts, etc.

1. **Body Tag (<body>):**

- Jo kuch user ko web page par dikhega, sab yaha likha jaata hai.

1. **Title Tag (<title>):**

- Browser tab ka naam.
- Example: `<title>Welcome</title>`

Basic HTML Tags and Elements

Let's explore some important tags:

Headings (h1 to h6):

<h1>Main Heading**</h1>**

<h2>Subheading**</h2>**

<h3>Another Subheading**</h3>**

- Tip: Headings ka use karne se content easy-to-read ban jaata hai.

Paragraph (p):

<p>This is a paragraph of text.**</p>**

- Pro Tip: Paragraphs text ko divide karte hain, jaise ek long essay ko sections mein todna.

Line Break (br):

<p>This is line one.**
This is line two.</p>**

- Nayi line start karne ke liye perfect.

Anchor Tag (a):

****Click here to visit Google****

Links banane ke liye use hota hai.

Fun Fact: Target="_blank" ka matlab hai "Open in a new tab."

Working with Lists

Lists content ko systematic way mein dikhate hain.

Ordered List (ol):

```
<ol>  
  <li>Step 1: Open VS Code</li>  
  <li>Step 2: Write HTML</li>  
  <li>Step 3: Run Live Server</li>  
</ol>
```

Unordered List (ul):

```
<ul>  
  <li>Fruits</li>  
  <li>Vegetables</li>  
</ul>
```

Analogy: Ordered list numbers use karta hai – jaise recipe steps.

Aur unordered list bullets use karta hai – jaise grocery list.

Working with Media

Adding images, videos, and audio makes your webpage interesting.

Images (img):

```

```

- **alt:** Jab image load nahi hoti, alt-text display hota hai. 6

Videos (video):

<video controls>

<source src="video.mp4" type="video/mp4">

</video>

- Controls add karne se user video ko play/pause kar sakta hai.

Audio (audio):

<audio controls>

<source src="audio.mp3" type="audio/mpeg">

</audio>

- Audio files ke liye perfect!

HTML Attributes

Attributes element ko extra information dete hain:

- href: Links ke liye.
- src: Media sources ke liye.
- alt: Images ke description ke liye.

<a href="https://youtube.com" target="_blank">Visit
YouTube****

<img src="logo.png" alt="Company Logo">

Commenting in HTML

Code mein comments likhne ke liye:

<!-- This is a comment -->

- Comments kisi bhi user ko nahi dikhenge – sirf coder ke liye helpful hote hain.



More on HTML

Welcome back, rockstar developers! Ab tak aap sab HTML ki basics samajh gaye ho. Aaj hum apne weapons aur sharpen karenge – advanced HTML ke saath! Ready? Chalo shuru karein!

Using the <div> Tag

The <div> tag is like the most versatile container in HTML. It's a block-level element used to group content for layout or styling.

Why <div> is special?

- Socho ek ghar ka hall. Usme chhoti-chhoti diwaarein hain jo different areas define karti hain – dining area, TV area, etc. div yeh hi kaam karta hai – content ko logically group karta hai.

Example:

```
<div style="border: 1px solid black; padding: 10px;">
```

```
<h2>Group 1</h2>
```

```
<p>This is a paragraph inside a div.</p>
```

```
</div>
```

Pro Tip: Though <div> is awesome, don't overuse it. Semantic HTML tags (discussed next) are more meaningful.

Semantic HTML Tags

Semantic tags are like labels that explain the content's purpose. They make your page more understandable for developers and search engines.

Common Semantic Tags:

1. <article>

Represents independent, reusable content like blog posts or news articles.

example:

```
<article><h2>Breaking News</h2>
```

```
<p>The sun rises in the east. Scientists confirm!</p></article>
```

Analogy: "Socho ek magazine ka ek article – standalone, reusable, aur informative."

2.<section>

Groups related content together.

```
<section>
```

```
<h1>About Us</h1>
```

```
<p>We are a team of passionate developers.</p></section>
```

3. <main>

Defines the main content of the document – jo sabse important hai.

```
<main>
```

```
<h1>Welcome to Our Website</h1>
```

```
<p>Explore our amazing features.</p>
```

```
</main>
```


4.<aside>

Side content like ads, sidebars, or related links.

<aside>

<h3>Related Articles</h3>

<p>Check out our other blogs!</p>

</aside>

5. <header>

Contains introductory or navigational content, like a page title or menu.

<header>

<h1>My Website</h1>

<nav>

Home | Contact

</nav>

</header>

6. <footer>

Represents the footer of the document or section.

<footer>

<p>© 2025 My Website. All rights reserved.</p>

</footer>

7. <details>

Creates a collapsible content block.

<details>

<summary>Click to learn more</summary>

<p>This is additional information.</p>

</details>

Fun: It's like an expandable FAQ section!

8. <figure>

Groups media content like images with captions.

```
<figure>
```

```

```

```
<figcaption>A breathtaking view of nature.</figcaption>
```

```
</figure>
```

Block vs. Inline Elements

Block Elements:

- Take up the full width.
- Always start on a new line.
- Examples: <div>, <p>, <h1>, <section>

Inline Elements:

- Only take up as much space as needed.
- Don't start on a new line.
- Examples: , <a>,

Analogy: "Block elements are like bricks (stand-alone and take up space), while inline elements are like decorations (fitting into existing spaces)."

Text Formatting Tags

1. – Bold Text

<p>This is bold text.</p>

2. <i> – Italic Text

<p>This is <i>italicized</i> text.</p>

3. <small> – Smaller Text

<p>This is <small>smaller</small> text.</p>

4. <ins> – Inserted (Underlined) Text

<p>This is <ins>inserted</ins> text.</p>

5. <sub> – Subscript

<p>Water formula is H₂O.</p>

6. <sup> – Superscript

<p>E = mc²</p>

7. – Strikethrough

<p>This is deleted text.</p>

8. <mark> – Highlighted Text

<p>This is <mark>highlighted</mark> text.</p>

Working with HTML Symbols

HTML provides predefined codes for special characters:

- **©;** © (Copyright)
- **←;** ← (Left Arrow)
- **♥;** ♥ (Heart Symbol)

Example:

```
<p>&copy; 2025 My Website</p><p>I <span style="color:red;">&hearts;</span> HTML!</p>
```

HTML Tables

Tables are used to organize data in rows and columns.

Structure:

- **<table>**: The container for the table.
- **<tr>**: Table rows.
- **<th>**: Table header cells.
- **<td>**: Table data cells.

```
<table border="1">
  <tr>
    <th>Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>John</td>
    <td>30</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>25</td>
  </tr>
</table>
```

Fun Fact: Tables are perfect for presenting structured data, but avoid using them for layouts – use CSS instead.



HTML Forms and Inputs

Aaj hum forms ke jaadu ke bare mein baat karenge. Yeh wahi forms hain jo tumhe login karne, sign up karne, ya feedback dene ke liye har website par milte hain. Forms ka kaam hai tumhare aur server ke beech dosti karwana. Chalo start karein!

What is a Form and Why is it Important?

Form ek digital zariya hai jahan user apni baatein, data ya feedback bhar ke server ko bhejta hai. Yeh data phir backend process karta hai aur action karta hai.

Why Important?

- **Data Collection:** Email, passwords, aur feedback jaise cheezein collect karna.
- **Interactivity:** Users ko website ke saath interact karne ka option dena.
- **Dynamic Websites:** Forms bina kaam nahi chalega – signup, login, aur even shopping cart bhi isi ke through chalta hai!

Creating a Simple Form

Ek simple form ka basic structure kuch aisa hota hai:

```
<form action="/submit" method="POST">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username"
placeholder="Enter your username" required>

  <label for="password">Password:</label>
  <input type="password" id="password" name="password"
placeholder="Enter your password" required>

  <button type="submit">Submit</button>
</form>
```

Breakdown:

1. **<form>**: Yeh sab elements ko group karta hai aur data ko server tak le jaata hai.
 - **action="/submit"**: Form data kahan bhejna hai (server ka URL ya endpoint).
 - **method="POST"**: Data bhejne ka tareeka (POST private hota hai, GET nahi).
2. **<label>**: Input field ka description aur accessibility ke liye.
 - **for="username"**: Yeh bata raha hai ki label kis input field ke liye hai.
3. **<input>**: User ka input lene ke liye.
 - **type="text"**: Single-line text input.
 - **placeholder="..."**: Input field ke andar hint dene ke liye.
 - **required**: Input lena compulsory hai.
4. **<button>**: Form submit karne ka button.

Types of Input Fields

1. Text Input (type="text")

Single-line text input ke liye.

```
<input type="text" name="fullname" placeholder="Enter your full name">
```

- **Example:** Name, username.
- **Use Case:** Signup forms, profile details.

2. Password Input (type="password")

Confidential data (like password) lene ke liye. Yeh input ko asterisks (****) mein dikhata hai.

```
<input type="password" name="userpassword" placeholder="Enter your password">
```

3. Checkbox (type="checkbox")

Multiple options select karne ke liye.

```
<label><input type="checkbox" name="hobbies" value="cricket"> Cricket</label>
```

```
<label><input type="checkbox" name="hobbies" value="music"> Music</label>
```

4. Radio Button (type="radio")

Single option select karne ke liye.

```
<label><input type="radio" name="gender" value="male"> Male</label>
```

```
<label>
```

```
<input type="radio" name="gender" value="female">
```

```
Female</label>
```


5. File Upload (type="file")

Files upload karne ke liye. Example: Resume ya profile picture upload karna.

```
<input type="file" name="resume">
```

6. Color Picker (type="color")

Color choose karne ke liye color palette kholta hai.

```
<label for="color">Favorite Color:</label>
```

```
<input type="color" id="color" name="favcolor">
```

7. Date Input (type="date")

Date choose karne ke liye calendar open karta hai.

```
<input type="date" name="dob">
```

8. Range Slider (type="range")

Values select karne ke liye slider.

```
<input type="range" name="volume" min="0" max="100">
```

9. Submit Button (type="submit")

Form submit karne ke liye button.

```
<button type="submit">Submit</button>
```


Multi-line Text Input (<textarea>)

Jahan users ko zyada text likhne ki zarurat ho (e.g., comments, feedback).

```
<textarea name="feedback" rows="4" cols="50"
placeholder="Write your feedback here...">

</textarea>
```

Attributes:

- **rows:** Kitni lines visible hain.
- **cols:** Kitni width hogi textarea ki.

Dropdown Menu (<select> and <option>)

Users ko ek list se option choose karne ke liye.

```
<label for="cars">Choose a car:</label>

<select id="cars" name="car">

<option value="volvo">Volvo</option>

<option value="bmw">BMW</option>

</select>
```

- **<select>**: Dropdown ka container.
- **<option>**: Har ek option define karta hai.

Attributes of Form Elements

1. method:

- **GET:** Data URL ke sath append hota hai (visible to user).
- **POST:** Data securely bheja jata hai (better for sensitive data).

2. action:

- Yeh specify karta hai ki data kahan bhejna hai (server endpoint).

<form action="/submit-data" method="POST">

3. required:

Kisi field ko mandatory banane ke liye.

<input type="text" name="name" required>

4. placeholder:

Input field ke andar temporary text.

<input type="email" placeholder="Enter your email">

5. enctype:

File uploads ke liye data encoding type define karta hai.

<form enctype="multipart/form-data">

6. target:

Response kahan dikhana hai, define karta hai.

- **_blank:** Naye tab mein response khulega.
- **_self:** Same tab mein khulega (default).



Media Tags in HTML

Dosto, ek plain text-based website kabhi kisi ka attention nahi kheechti. Media hi woh tadka hai jo tumhare web page ko zyada interesting banata hai. Toh chalo, aaj seekhte hain audio, video, aur iframe ka jaadu!

Audio Tag (<audio>)

Purpose: Webpage par audio files embed karne ke liye.

Syntax:

```
<audio controls><source src="audio-file.mp3" type="audio/mpeg">
```

Your browser does not support the audio tag.

```
</audio>
```

Explanation:

- **<audio>**: Audio content embed karne ke liye container.
- **<source>**: File ka source define karta hai.
- **controls attribute**: Play, pause, aur volume adjust karne ke buttons dikhata hai.
- **Fallback text**: "Your browser does not support the audio tag." agar browser audio support nahi karta.

Video Tag (<video>)

Purpose: Webpage par video files embed karne ke liye.

Syntax:

<video width="**640**" height="**360**" **controls**>

<source src="**video-file.mp4**" type="**video/mp4**">

Your browser does not support the video tag.

</video>

Explanation:

- **<video>**: Video embed karne ke liye container.
- **width & height**: Video player ka size set karta hai.
- **controls**: Play, pause, volume aur fullscreen buttons ke liye.
- **<source>**: Video file ka source specify karta hai.
- **Fallback text**: Agar browser video support nahi kare toh yeh message show hoga.

Common Attributes for Media Tags

1. src:

Media file ka location specify karta hai.

```
<audio src="song.mp3" controls></audio>
```

2. width and height:

Media element ka size define karta hai.

```
<video src="movie.mp4" width="800" height="450" controls>
</video>
```

3. muted:

Media ko by default mute karta hai.

```
<video src="movie.mp4" muted controls></video>
```

4. loop:

Media ko repeat karta hai jab tak user stop na kare.

```
<audio src="song.mp3" loop controls></audio>
```

5. autoplay:

Page load hone par media automatically play hota hai.

```
<video src="movie.mp4" autoplay muted></video>
```

6. controls:

Media ke liye buttons add karta hai (play, pause, volume).

Using the <iframe> Tag

Purpose: Kisi doosri website ya resource ko apne web page par embed karne ke liye.

Syntax:

<iframe

src="**https://www.youtube.com/embed/dQw4w9WgXcQ**"
width="**560**" height="**315**" frameborder="**0**" **allowfullscreen**>

</iframe>

Explanation:

- **<iframe>:** External content (like **YouTube** video ya **Google Map**) embed karta hai.
- **src:** Embedded resource ka URL specify karta hai.
- **width & height:** Iframe ka size define karta hai.
- **frameborder:** Iframe ke border ko set karta hai (0 means no border).
- **allowfullscreen:** Fullscreen mode enable karta hai.



CSS – The Magic Wand of Web Design

Class, aaj hum web pages ko sundar banayenge! HTML ka kaam hai structure dena, lekin CSS ka kaam hai usko style karna, jaise bina makeup ke kisi shaadi mein jaana thoda ajeeb lagta hai, waise hi bina CSS ke ek web page adhura lagta hai. Chalo shuru karte hain!

Introduction to CSS and Why It's Important

CSS (Cascading Style Sheets) is the language used to style HTML elements.

- HTML se page banta hai, lekin woh boring sa dikhta hai. CSS us page ka makeover expert hai!
- With CSS, hum color, font, spacing, alignment aur bahut kuch control karte hain.

Example Without CSS (Dukh Bhari Kahani):

```
<!DOCTYPE html>

<html>

<head>

  <title>Plain Page</title>

</head>

<body>

  <h1>Welcome to My Page</h1>

  <p>This page has no style. 😞</p>

</body>

</html>
```


With CSS (Khushi Bhari Kahani):

```
<!DOCTYPE html>

<html>

<head>

  <title>Styled Page</title>

  <style>

    h1 {

      color: blue;

      text-align: center;

    }

    p {

      color: gray;

      font-size: 18px;

    }

  </style>

</head>

<body>

  <h1>Welcome to My Page</h1>

  <p>Look at this beautiful style! 😊</p>

</body>

</html>
```


CSS Syntax, Selectors, and Comments

CSS ka syntax simple aur organized hota hai:

```
selector {  
    property: value;  
}
```

Example:

```
h1 {  
    color: red; /* Yeh red color set karega */  
    font-size: 24px; /* Font size badha dega */  
}
```

- **Selector:** Kya style karna hai (e.g., h1, p, .className, #idName).
- **Property:** Kaunsa feature style karna hai (e.g., color, font-size).
- **Value:** Property ka kya value hoga (e.g., red, 20px).
- **Comment:** Notes likhne ke liye, browser ignore karega (/* This is a comment */).

Adding CSS to an HTML Page

CSS ko 3 tariko se HTML mein add kar sakte hain:

Inline CSS: Directly element ke andar likho.

<h1 style="color: green;">Inline CSS Example</h1>

Internal CSS: <style> tag ka use karo in the <head>.

<style>

h1 {
color: blue;
}

</style>

External CSS: Ek alag .css file banao aur usko <link> karo.

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

CSS Selectors – Class, ID, and Element

CSS selectors web elements ko target karte hain, aur unka style gang war kaafi interesting hota hai. 😊

Element Selector: Direct tag ko target karega.

```
h1 {  
    color: purple;  
}
```

Class Selector: Multiple elements ko style karne ke liye.

```
<p class="highlight">This is important!</p>  
  
.highlight {  
    background-color: yellow;  
}
```

ID Selector: Ek unique element ke liye.

```
<p id="uniquePara">Style me alone!</p>  
  
#uniquePara {  
    color: orange;  
}
```

Difference Between Class and ID:

Class: Reusable (jaise ek gang ke sare members ek jaisa kapda pehne).

ID: Unique (jaise ek king ka crown, sirf ek ke liye hota hai).

Precedence of Selectors

Jab multiple selectors ek hi element ko target karein, toh precedence hoti hai:

1. **Inline CSS** > 2. **ID** > 3. **Class** > 4. **Element**.

Example:

<p id="para" class="text" style="color: red;">

Precedence Example

</p>

- **Final color:** red (kyunki inline CSS sabse zyada priority rakhta hai).

Styling Text Using CSS

Ab baat karte hain text styling ki:

Font Family: Choose a font.

```
p {  
    font-family: Arial, sans-serif;  
}
```

Font Style: Italic ya normal.

```
p {  
    font-style: italic;  
}
```

Font Weight: Text bold kare.

```
p {  
    font-weight: bold;  
}
```

Line Height: Line spacing control kare.

```
p {  
    line-height: 1.5;  
}
```

Text Decoration: Underline ya strikethrough.

```
p {  
    text-decoration: underline;  
}
```

Text Align: Center, left, ya right alignment.

```
p {  
    text-align: center;  
}
```


Text Transform: Uppercase, lowercase, ya capitalize.

```
p {  
  text-transform: uppercase;  
}
```

Letter Spacing: Letters ke beech space badhao.

```
p {  
  letter-spacing: 2px;  
}
```

Word Spacing: Words ke beech space adjust karo.

```
p {  
  word-spacing: 5px;  
}
```

Text Shadow: Cool shadow effects add karo.

```
h1 {  
  text-shadow: 2px 2px 5px gray;  
}
```




CSS – Colors, Units, Borders, and Box Properties

Bachcho, aaj hum CSS ke colors, units, borders aur box properties ke magic ka jaadu seekhenge. Samajh lo, web page kaise 'wow' dikh sakta hai. Tumhara design ka swag aaj se shuru hoga!" 😎

Working with Colors in CSS

CSS mein colors lagana ek art hai, aur options kaafi hain:

Color Names:

CSS me predefined color names use kar sakte ho (e.g., red, blue, green).

```
h1 {  
    color: red;  
}
```

RGB (Red, Green, Blue):

RGB me colors ko mix karke create karte hain.

```
h1 {  
    color: rgb(255, 0, 0); /* Pure red */  
}
```

HEX Codes:

Ek unique 6-character code, hexadecimal values ke saath.

```
h1 {  
    color: #ff0000; /* Red */  
}
```


HSL (Hue, Saturation, Lightness):

Hue: Color (angle in a circle), Saturation: Intensity, Lightness: Brightness.

```
h1 {  
  color: hsl(0, 100%, 50%); /* Red */  
}
```

RGBA and HSLA:

RGB aur HSL ke saath transparency add karne ka option (last value 0-1 hoti hai).

```
h1 {  
  color: rgba(255, 0, 0, 0.5); /* Half-transparent red */  
}
```

Working with CSS Units

Web elements ka size define karne ke liye units kaafi zaruri hain:

Pixels (px): Fixed size. Ekdum accurate, lekin responsive nahi.

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

Percentage (%): Parent element ke size ka percentage.

```
div {  
  width: 50%;  
}
```


Rem: Root element ke font size ka multiple.

```
p {  
    font-size: 2rem; /* 2x root font size */  
}
```

Em: Current element ke font size ka multiple.

```
p {  
    font-size: 1.5em; /* 1.5x current font size */  
}
```

View Width (vw) and View Height (vh): Screen ke viewport ka percentage.

```
div {  
    width: 100vw; /* Full viewport width */  
}
```

Min and Max: Minimum aur maximum size ke liye.

```
div {  
    min-width: 300px;  
    max-width: 600px;  
}
```


Working with Borders and Border Styling

Border elements ko highlight karne ke kaam aata hai:

Basic Border:

```
div {  
    border: 2px solid black;  
}
```

Border Radius: Corners ko round banata hai.

```
div {  
    border: 2px solid black;  
    border-radius: 10px; /* Rounded corners */  
}
```

Dashed and Dotted Borders:

```
div {  
    border: 2px dashed red;  
}
```

Shorthand Syntax: Ek line me sab kuch define karo.

```
div {  
    border: 2px solid blue;  
}
```


Working with Box Properties

Box properties tumhare web element ka area and spacing define karte hain:

Margin: Element ke bahar ka space.

```
div {  
    margin: 20px;  
}
```

Padding: Content aur border ke beech ka space.

```
div {  
    padding: 10px;  
}
```

Box Sizing: Padding aur border ko size calculation me include karta hai.

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

Width and Height: Dimensions set karta hai.

```
div {  
    width: 300px;  
    height: 200px;  
}
```


Understanding Background Properties

Background ke effects se page ka style level upar le jao:

Background Color:

```
div {  
    background-color: lightblue;  
}
```

Background Image:

```
div {  
    background-image: url('background.jpg');  
}
```

Background Size:

```
div {  
    background-size: cover;  
}
```

Background Attachment:

Fixed: Scroll karne par bhi background rukta hai.

```
div {  
    background-attachment: fixed;  
}
```

Background Repeat:

```
div {  
    background-repeat: no-repeat;  
}
```

Linear Gradient: Smooth color transitions.

```
div {  
    background: linear-gradient(to right, red, yellow);  
}
```




Display

Chalo bachho, aaj hum CSS ke advanced concepts explore karenge jo tumhare web pages ko ekdum stylish aur organized banayenge! Aaj ka episode hai: "CSS ka deeper dive with Flex, Grid, Positioning aur selectors ka magic!" 🎨✨

Display Properties ka Jaadu

Display property decide karti hai ki ek element kaise behave karega aur page par kaisa dikhai dega. Let's explore! 🚀

Inline:

Kya hota hai?

Ye elements ek hi line me dikhte hain, aur sirf utna space lete hain jitna content ka size hota hai.

Example: ``, `<a>`

```
span {  
    display: inline;  
}
```

Dekhne me kaisa?

Text ke beech me small tags laga kar kaam hota hai, jaise ek line me highlighted word.

Block:

Ye elements hamesha ek nayi line se shuru hote hain aur full width occupy karte hain.

Example: <div>, <p>

```
div {  
    display: block;  
}
```

Kab use karein?

Jab alag sections create karne ho, jaise headings ya paragraphs.

Inline-block:

Inline jaisa behave karta hai, lekin block ki tarah height aur width control karne deta hai.

```
span {  
    display: inline-block;  
    width: 100px;  
    height: 50px;  
}
```

Special feature:

Ek line me elements dikhte hain, lekin unka size adjust kar sakte ho!

CSS Positioning ?

Positioning CSS ka ek feature hai jisse tum kisi element ko page ke andar kahin bhi adjust kar sakte ho. Yeh decide karta hai ki ek element page me ya doosre elements ke sath kaise behave karega.

Types of Positioning

1. Static (Default Positioning)

- Default hota hai har element ke liye.
- Element normal document flow ke according place hota hai.

Example:

`<div class="static-box">Main ek static element hoon!</div>`

```
.static-box {  
    position: static; /* Default hai, likhne ki zarurat nahi */  
}
```

Mazedar baat: Static elements scroll ke sath move karte hain.

2. Relative Positioning

- Element apne original position ke relative move karta hai.
- Use karo top, right, bottom, left properties ke saath.

Example:

```
<div class="relative-box">Main relative hoon!</div>
```

```
.relative-box {  
  position: relative;  
  top: 20px;  
}
```

Output: Element apni jagah se shift hota hai lekin uska space document flow me reserve rehta hai.

3.Absolute Positioning

- Element apne nearest positioned ancestor ke relative place hota hai. Agar ancestor positioned nahi hai, to yeh viewport ke relative hota hai.
- top, left, right, bottom properties yahan important hain.

Example:

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

```
<div class="absolute-box">Main absolute hoon!</div>  
</div>
```

```
.absolute-container {  
  position: relative; /* Ancestor ko positioned banao */  
}
```

```
.absolute-box {  
  position: absolute;  
  top: 10px;  
  left: 20px;  
}
```

Important: Absolute positioning document flow se bahar chala jata hai, iska space nahi banta.

4. Fixed Positioning

- Yeh element viewport ke relative fix hota hai, chahe tum scroll kar lo, yeh wahi rahega.
- Iska common use sticky navbar ya footer banane me hota hai.

Example:

```
<div class="fixed-box">Main fixed hoon!</div>
```

```
.fixed-box {  
  position: fixed;  
  bottom: 0;  
  right: 0;  
  background-color: lightblue;  
  padding: 10px;  
}
```

Use Case: Chat boxes, sticky ads, ya "Back to Top" buttons.

5. Sticky Positioning

- Hybrid hota hai relative aur fixed ka.
- Jab tak tum scroll nahi karte, yeh relative behave karta hai. Scroll karte hi yeh fix ho jata hai ek position par.

Example:

```
<div class="sticky-box">Main sticky hoon!</div>
```

```
.sticky-box {  
  position: sticky;  
  top: 50px;  
  padding: 10px;  
}
```


Positioning Properties: top, right, bottom, left

- Ye properties define karti hain ki element kitna move karega.
- Inka kaam tab hota hai jab tum relative, absolute, fixed, ya sticky position use kar rahe ho.

Example:

```
.position-box {  
    position: absolute;  
    top: 20px;  
}
```

z-index: Kaun Sa Element Upar Aayega?

- z-index decide karta hai ki kaunsa element doosre ke upar ya neeche dikhega.

Example:

```
<div class="box box1">Box 1</div>  
<div class="box box2">Box 2</div>  
  
.box {  
    position: absolute;  
}  
  
.box1 {  
    z-index: 1; /* Neeche rahega */  
}  
  
.box2 {  
    z-index: 2; /* Upar aayega */  
}
```

Note: z-index tabhi kaam karta hai jab element positioned ho.

Flexbox

Flexbox ka kaam hai elements ko dynamic aur responsive alignment dena. Iska matlab hai ki tumhare elements apne aap adjust ho jayenge screen size ke hisaab se!

Socho tumhare paas ek dabba hai (div), aur tumhe uske andar 3 chhote dabbe (items) align karne hain. Flexbox is like that one cool kid jo sabko line me khada karwa deta hai—center me, corner me, ya evenly spaced.

Step 1: Flexbox Ko Activate Karo

Flexbox lagane ke liye parent container par display: flex use karna hota hai. Example lete hain:

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

.container {
  display: flex; /* Flexbox activate */
}
```


Flexbox Properties Ko Decode Karte Hain

1. flex-direction

- Yeh decide karta hai ki tumhare items ka layout kaise hoga —horizontal (row) ya vertical (column).
- Default value: row

.container {

display: flex;

flex-direction: row; /* Horizontal (default) */

}

Options:

- row (left to right)
- row-reverse (right to left)
- column (top to bottom)
- column-reverse (bottom to top)

Example: Socho tumhe ek vertical navbar banana hai. Bas flex-direction: column lagao!

2. justify-content

- Yeh property tumhare items ko horizontally align karti hai.

```
.container {
```

```
display: flex;
```

```
justify-content: center;
```

```
}
```

Options:

- **flex-start:** Items left align honge.
- **flex-end:** Items right align honge.
- **center:** Items center me honge.
- **space-between:** First aur last item edges pe honge, baaki evenly spaced.
- **space-around:** Har item ke aas-paas equal space.

Example:

Socho tum ek button row bana rahe ho, aur chahte ho ki buttons evenly spaced ho. Use space-between.

3. align-items

- Yeh property tumhare items ko vertically align karti hai.

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

Options:

- **flex-start:** Top pe align.
- **flex-end:** Bottom pe align.
- **center:** Center me align.
- **baseline:** Text ke baseline ke hisaab se align.
- **stretch:** By default, items ko stretch kar deta hai.

Example:

Socho tumhe vertically centered buttons chahiye—align-items: center lagao aur ho gaya!

4. flex-wrap

- By default, Flexbox items ek hi row/column me rehte hain, chahe space kam ho jaye. flex-wrap lagao aur items wrap hone lagte hain.

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

Options:

- **nowrap (default):** Items wrap nahi honge.
- **wrap:** Items next line pe chale jayenge agar space kam ho.
- **wrap-reverse:** Wrap karega lekin reverse direction me.

5. align-content

- Jab multiple rows/columns hote hain, tab unko align karne ke liye use hoti hai.

.container {

display: flex;

flex-wrap: wrap;

align-content: space-between;

}

Options:

- **flex-start:** Top pe sab rows align.
- **flex-end:** Bottom pe sab rows align.
- **center:** Center me sab rows align.
- **space-between:** Rows ke beech me equal space.
- **space-around:** Rows ke aas-paas equal space.
- **stretch:** Rows ko stretch kar deta hai.

6. flex-grow, flex-shrink, flex-basis

- Ye properties items ke size aur behavior ko control karti hain.

flex-grow

- Decide karta hai ki item kitna space le sakta hai agar extra space available ho.

.item {

flex-grow: 1; /* Sab items equal space lenge */

}

flex-shrink

- Decide karta hai ki item kitna shrink karega agar space kam ho.

.item {

flex-shrink: 0; /* Shrink mat karo */

}

flex-basis

- Item ka initial size set karta hai.

.item {

flex-basis: 100px; /* Har item 100px wide hoga */

}

align-self

- Individual item ke vertical alignment ko override karta hai.

.item {

align-self: flex-end; /* Sirf yeh item bottom pe jayega */

}

Ek Mazedaar Example:

Chalo ek responsive navbar banate hain jo center me aligned ho.

```
<div class="navbar">  
  
<div class="logo">LOGO</div>  
  
<div class="links">  
  
  <div>Home</div>  
  
  <div>About</div>  
  
  <div>Contact</div>  
  
</div>  
</div>
```

```
.navbar {  
  
  display: flex;  
  
  justify-content: space-between;  
  
  align-items: center;  
  
  background-color: #f0f0f0;  
  
  padding: 10px;  
  
}
```

```
.links {  
  
  display: flex;  
  
  gap: 20px;  
  
}
```


Grid

Flexbox ki tarah Grid bhi ek layout system hai, lekin yeh 2D layout banane ke liye hota hai. Matlab, tum rows aur columns dono ko ek saath control kar sakte ho. Flexbox ka focus zyada tar ek hi direction (row ya column) par hota hai, lekin Grid dono directions me kamaal karta hai!

Step 1: Grid Layout Ko Activate Karo

Sabse pehle parent container me display: grid lagao, aur tumhare elements ek grid system ka part ban jayenge.

Example:

```
<div class="grid-container">  
  <div class="item">1</div>  
  <div class="item">2</div>  
  <div class="item">3</div>  
  <div class="item">4</div>  
</div>
```

```
.grid-container {  
  display: grid;  
}
```


Step 2: Columns aur Rows Set Karo

Grid ko structure dene ke liye tumhe columns aur rows define karni hoti hain.

1. grid-template-columns / grid-template-rows

- Columns aur rows ke size define karte hain.

```
.grid-container {  
display: grid;  
grid-template-columns: 100px 100px 100px;  
/* Teen columns, har ek 100px wide */  
grid-template-rows: 50px 50px;  
/* Do rows, har ek 50px tall */  
}
```

Result: Ek 3x2 grid banega—3 columns aur 2 rows.

Shortcut:repeat()

Agar same size ke multiple columns/rows chahiye, to repeat() use karo.

```
.grid-container {  
grid-template-columns: repeat(3, 100px); /* 3 columns of  
100px each */  
}
```

2. gap

- Rows aur columns ke beech ka space define karta hai.

```
.grid-container {  
display: grid;  
grid-template-columns: repeat(3, 100px);  
gap: 10px; /* Rows aur columns ke beech 10px ka gap */  
}
```


How to Place Grid Items ?

3. grid-column / grid-row

- Tum individual items ko kisi specific column ya row me rakh sakte ho.

.item {

grid-column: 2; /* 2nd column me rakho */

grid-row: 1; /* 1st row me rakho */

}

4. grid-column-start / grid-column-end

- Ek item ko multiple columns/rows me stretch karne ke liye use hota hai.

.item {

grid-column-start: 1; /* Start from 1st column */

grid-column-end: 3; /* End at 3rd column */

}

Shortcut:

.item {

grid-column: 1 / 3; /* Start from 1st, end at 3rd column */

}

5. grid-template-areas

- Tum apne layout ka naam dekar ek readable aur structured grid bana sakte ho.

Example: Ek 3x3 grid jisme header, sidebar, aur content ho.

```
.grid-container {  
  display: grid;  
  grid-template-areas:  
    "header header header"  
    "sidebar content content"  
    "footer footer footer";  
  grid-template-rows: 50px 1fr 30px;  
  grid-template-columns: 100px 1fr;  
}  
  
.header {  
  grid-area: header;  
}  
  
.sidebar {  
  grid-area: sidebar;  
}  
  
.content {  
  grid-area: content;  
}  
  
.footer {  
  grid-area: footer;  
}
```


Responsive Layouts with Grid

6. auto-fit aur auto-fill

- Ye tumhare grid ko automatically adjust kar dete hain screen size ke hisaab se.

```
.grid-container {  
display: grid;  
grid-template-columns: repeat(auto-fit, minmax(150px,  
1fr));  
gap: 10px;  
}
```

- **auto-fit:** Items ko compact karta hai.
- **auto-fill:** Items ko fill karta hai pura space cover karne ke liye.

7. justify-content / align-content

- Grid ka alignment control karte hain (similar to Flexbox).

```
.grid-container {  
display: grid;  
grid-template-columns: repeat(3, 100px);  
justify-content: center; /* Horizontal alignment */  
align-content: center; /* Vertical alignment */  
}
```


Ek Mazedaar Example:

Chalo ek photo gallery banate hain jo responsive ho. 📱

```
<div class="gallery">  
  <div class="photo">1</div>  
  <div class="photo">2</div>  
  <div class="photo">3</div>  
  <div class="photo">4</div>  
  <div class="photo">5</div>  
  <div class="photo">6</div>  
</div>
```

```
.gallery {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(150px,  
1fr));  
  gap: 10px;  
}
```

```
.photo {  
  background-color: #f0f0f0;  
  padding: 20px;  
  text-align: center;  
  border: 1px solid #ccc;  
}
```


Pseudo Classes and Pseudo Elements

Pseudo classes aur pseudo elements ka matlab hai ki tum CSS ke through kisi element ke special state ya uske specific part ko target kar sakte ho bina uska structure badle.

Pseudo Classes: Special Effects for States

1. **:hover** – Mouse laane ka jaadu!

Jab tum kisi element par mouse le jaate ho, to uska style change karne ke liye hum :hover use karte hain.

Example:

```
<button class="hover-btn">Hover Karo!</button>
```

```
.hover-btn {  
    background-color: lightblue;  
    color: black;  
    padding: 10px;  
    border: none;  
    border-radius: 5px;  
}
```

```
.hover-btn:hover {  
    background-color: orange;  
    color: white;  
    cursor: pointer; /* Dikhaane ke liye ki yeh clickable hai */  
}
```

Samajhlo: Jab bhi tum button pe mouse loge, iska background lightblue se orange ho jayega. Cursor bhi pointer banega, kaafi professional lagta hai na?

2. :focus – Input box ka highlight!

Jab tum kisi input field pe click karte ho, tab uska styling kaise change hoga yeh :focus decide karta hai.

Example:

```
<input type="text" class="focus-input" placeholder="Focus  
Mujhpar!">
```

```
.focus-input {  
    border: 2px solid gray;  
    padding: 8px;  
}
```

```
.focus-input:focus {  
    border-color: green; /* Border ka color green ho jayega  
    outline: none; /* Default outline ko hata diya */  
}
```

Logic: Jab user type kar raha hoga, toh green border milega.
Yeh visual feedback user ko achha experience deta hai.

3. :active – Button dabane ka effect!

Yeh tab trigger hota hai jab tum kisi button ko press karte ho.

Example:

```
<button class="active-btn">Click Here!</button>
```

```
.active-btn {
```

```
background-color: lightcoral;
```

```
padding: 10px;
```

```
border: none;
```

```
}
```

```
.active-btn:active {
```

```
background-color: darkred;
```

```
color: white;
```

```
}
```

Logic: Jab user button dabayega, iska color darkred ho jayega.

Ek instant feedback!

Pseudo Elements: Adding Content Without HTML

1. ::before – Pehle ka style!

Kisi bhi element ke content ke pehle kuch add karna ho toh use karo.

Example:

```
<p class="before-example">Hello World!</p>
```

```
.before-example::before {
```

```
content: "👉 "; /* Emoji add kiya */
```

```
}
```

Output: 👉 Hello World!

2. **::after** – Baad ka style!

Content ke baad kuch add karna ho toh.

Example:

```
<p class="after-example">Welcome</p>
```

```
.after-example::after {  
  content: "👏";  
}
```

Output: Welcome 👏

Magic: Tum emojis, icons ya even additional text ko bina HTML me likhe add kar sakte ho.

CSS Transitions: Smooth Effects ka King!

Transitions tumhare design me smoothness le kar aata hai.
Matlab ek state se doosri state me dheere se jaane ka effect.

Syntax:

transition: property duration timing-function delay;

Example:

```
<div class="transition-box"></div>
```

```
.transition-box {
```

```
  width: 100px;
```

```
  height: 100px;
```

```
  background-color: lightblue;
```

```
  transition: background-color 0.5s ease-in-out;
```

```
/* Smooth effect */
```

```
}
```

```
.transition-box:hover {
```

```
  background-color: orange; /* Hover karte hi orange ho jayega
```

```
*/
```

```
}
```

Logic: Mouse le jaane pe background dheere-dheere orange ho jayega. Yeh animation users ko smooth aur interactive feel deta hai.

Transform: Ghoomna, Bada-Chhota Karna aur Twist!

Transform tumhare elements ko visually modify karne ke liye kaam aata hai.

Example: Rotate aur Scale

```
<div class="transform-box"></div>
```

```
.transform-box {
```

```
width: 100px;
```

```
height: 100px;
```

```
background-color: lightgreen;
```

```
transform: rotate(45deg) scale(1.5);
```

```
/* 45-degree ghooma aur 1.5 times bada */
```

```
}
```

3D Transform: WOW Effects!

Jab tumhe depth ka effect banana ho, tab use karo.

Example:

```
<div class="box-3d">Rotate Me</div>
```

```
.box-3d {
```

```
width: 100px;
```

```
height: 100px;
```

```
background-color: coral;
```

```
transform: rotateX(30deg) rotateY(45deg);
```

```
/* X aur Y axis me rotate kiya */
```

```
}
```


CSS Animation: Full-on Drama!

Animation tumhare elements ko dynamic movement aur style deta hai.

Syntax:

```
@keyframes animation-name {  
  0% { property: value; }  
  100% { property: value; }  
}
```

Example:

```
<div class="animation-box"></div>
```

```
@keyframes move {  
  0% {  
    transform: translateX(0);  
  }  
  100% {  
    transform: translateX(200px);  
  }  
}
```

```
.animation-box {  
  width: 100px;  
  height: 100px;  
  background-color: pink;  
  animation: move 2s infinite; /* Har 2 second me repeat hoga */  
}
```


Mobile-First vs Desktop-First Approach

Is approach me tum pehle mobile ke liye design banate ho, fir usko bade devices ke liye adjust karte ho (jaise tablet aur desktop).

```
/* Basic mobile styling */
```

```
body {  
    font-size: 16px;  
}
```

```
@media (min-width: 768px) {
```

```
    /* Tablet styling */  
    body {  
        font-size: 18px;  
    }  
}
```

```
@media (min-width: 1024px) {
```

```
    /* Desktop styling */  
    body {  
        font-size: 20px;  
    }  
}
```


2. Desktop-First Approach

Yaha tum pehle desktop ke liye design karte ho aur usko chhoti screens ke liye adjust karte ho.

```
/* Basic desktop styling */
```

```
body {  
  
  font-size: 20px;  
  
}
```

```
@media (max-width: 1024px) {
```

```
  /* Tablet styling */
```

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

```
@media (max-width: 768px) {
```

```
  /* Mobile styling */
```

```
body {  
  
  font-size: 16px;  
  
}  
}
```

Samajhlo:

- Agar tumhe mobile audience zyada milti hai, toh mobile-first approach lo.
- Agar tum desktop users ko target kar rahe ho, toh desktop-first approach better hai.

Responsive Measurement Units

Responsive banane ke liye humein different units samajhni padengi. Let's decode them!

1. px (Pixel)

Sabse chhota unit hota hai jo ek screen ke ek dot ko represent karta hai. Fixed size ke liye use hota hai.

2. % (Percentage)

Relative unit hai jo parent element ka percentage leta hai. Har device ke hisaab se adjust hota hai.

```
div {  
  width: 50%; /* Parent width ka aadha lega */  
}
```

3. rem (Root em)

Yeh root element ke font-size ke basis par size define karta hai. Default root font-size 16px hoti hai.

```
h1 {  
  font-size: 2rem; /* 2 x 16px = 32px */  
}
```

4. vh / vw (Viewport Height/Width)

Yeh poore screen ke height ya width ka percentage leta hai.

```
div {  
  width: 50vw; /* Screen width ka aadha */  
  height: 50vh; /* Screen height ka aadha */  
}
```


Viewport Meta Tag

Responsive design ka hero tag hai! Yeh browser ko batata hai ki page ka size aur scale kaise handle karna hai.

Add this in <head>:

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

Kya karega?

- Page ko screen width ke hisaab se adjust karega.
- Zoom karne ka default behavior set karega.

Responsive Images and Typography

1. Responsive Images

```

```

Logic: Image ki width container ke 100% ke barabar hogi aur height automatically adjust hogi. Isse image stretch nahi hogi.

2. Responsive Typography

```
h1 {  
    font-size: calc(1rem + 1vw); /* Dynamic font-size */  
}
```

Logic: Font size screen width ke saath dynamically change karega.

Media Queries

Media queries CSS ka sabse powerful tool hai jo tumhe screen size ke hisaab se styles apply karne deta hai.

Syntax:

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

/* Mobile screens ke liye */

```
@media (max-width: 768px) {  
    body {  
        background-color: lightblue;  
    }  
}
```

/* Tablet screens ke liye */

```
@media (min-width: 769px) and (max-width: 1024px) {  
    body {  
        background-color: lightgreen;  
    }  
}
```

/* Desktop screens ke liye */

```
@media (min-width: 1025px) {  
    body {  
        background-color: lightcoral;  
    }  
}
```

Samajhlo:Alag-alag screen sizes ke liye alag-alag colors apply ho rahe hain. Tum ise layout aur fonts ke liye bhi use kar sakte ho.

CSS Functions for Responsiveness

1. clamp()

Dynamic value set karta hai jisme min, preferred aur max value hoti hai.

```
h1 {  
  
  font-size: clamp(16px, 5vw, 32px);  
  
  /* 16px se chhota nahi, 32px se bada nahi */  
  
}
```

2. max()

Sabse badi value choose karta hai.

```
div {  
  
  width: max(300px, 50%);  
  
}
```

3. min()

Sabse chhoti value choose karta hai.

```
div {  
  
  width: min(500px, 80%);  
  
}
```


HTML Structure for Responsive Design

Responsive design ke liye HTML ka structure simple aur logical hona chahiye.

```
<!DOCTYPE HTML>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Responsive Design</title>
```

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

```
</head>
```

```
<body>
```

```
<header>
```

```
<h1>Responsive Website</h1>
```

```
<nav>
```

```
<ul>
```

```
<li>Home</li>
```

```
<li>About</li>
```

```
<li>Contact</li>
```

```
</ul>
```

```
</nav>
```

```
</header>
```

```
<main>
```

```
<section>
```

```
<h2>Welcome!</h2>
```

```
<p>This is a responsive website demo.</p>
```

```
</section>
```

```
</main>
```

```
<footer>
```

```
<p>© 2025 Responsive Design</p>
```

```
</footer>
```

```
</body>
```

```
</html>
```


JavaScript

JavaScript ek programming language hai jo tumhari website ko interactive banata hai. Matlab, agar HTML ek skeleton hai aur CSS uska design hai, toh JavaScript us skeleton ko jaan deta hai.

Jaise ki tumhe button click karna ho aur kuch actions dikhana ho, ya kisi input field mein type karte hi suggestions aayein—ye sab JavaScript ka kamaal hai!

Weakly Typed Language

JavaScript ko weakly typed language kehte hain. Matlab, tum ek hi variable mein alag-alag type ke data store kar sakte ho.

j

```
let name = 'Shaaz';
```

```
// Yeh ek string hai.
```

```
name = 5;
```

```
// Ab yeh ek number ban gaya.
```

```
name = { age: 23 };
```

```
// Ab yeh ek object ban gaya.
```

JavaScript tumse strict rules follow karne ko nahi bolta, isliye beginner-friendly hai.

Console ka Magic

JavaScript ka output screen par dekhne ke liye hum **console.log()** use karte hain. Yeh tumhare browser ke "console" section mein result dikhata hai.

example:

```
let name = "Shaaz";  
console.log(name);  
  
// Output: Shaaz
```

Console kaise open karein?

1. Browser kholo (Chrome recommended).
2. Right-click karo -> "Inspect" par click karo -> "Console" tab pe jao.

Variables - Data Store Karne Ka Dibba

Variable ek container hai jisme tum data store kar sakte ho. JavaScript mein 3 tareeke hain variables declare karne ke liye:

1. **var**: Purana tareeka, overwrite kar sakte ho.
2. **let**: Naya tareeka, block ke andar limited rehta hai.
3. **const**: Permanent value, badal nahi sakte.

Example:

```
var score = 10;  
let name = "Shaaz";  
const pi = 3.14;
```


Data Types - Kya-Kya Cheezein Store Hoti Hain?

JavaScript ke paas alag-alag type ka data store karne ke options hain:

Numbers: Jaise 1, 2, 3.14.

```
let age = 25;
```

Strings: Text ke liye double ya single quotes.

```
let name = "Shaaz";
```

Booleans: True ya False.

```
let isLoggedIn = true;
```

null: Jab tum intentionally kuch khaali store karna chaho.

```
let emptyValue = null;
```

undefined: Jab tumne variable declare kiya hai, par value assign nahi ki.

```
let noValue;
```

```
console.log(noValue); // undefined
```

Objects: Key-value pairs ke liye.

```
let person = { name: "Shaaz", age: 25 };
```

Arrays: List of items.

```
let fruits = ["Apple", "Banana", "Mango"];
```


Equality - Compare

JavaScript mein tum values ko compare kar sakte ho:

== (Loose Equality): Bas values check karta hai, types nahi.

```
console.log(5 == "5"); // true
```

=== (Strict Equality): Values aur types dono check karta hai.

```
console.log(5 === "5"); // false
```

Conditional Statements - Agar Ye, Toh Wo

Kuch conditions check karke, alag-alag code chalana ho, toh if-else ka use karte hain.

Example:

```
let age = 18;
```

```
if (age >= 18) {  
    console.log("You can vote!");  
} else {  
    console.log("Sorry, you're too young to vote.");  
}
```


Logical Operators - Conditions Ko Jodo

Agar tumhe ek condition se zyada check karni hai, toh logical operators ka use hota hai:

&& (AND): Dono condition true honi chahiye.

```
if (age > 12 && age < 20) {  
    console.log("You're a teenager!");  
}
```

|| (OR): Koi ek condition true ho toh chalega.

```
if (age === 18 || age === 19) {  
    console.log("You're just starting adulthood!");  
}
```

Loops - Repeat Karna

Kabhi-kabhi tumhe ek kaam baar-baar karna hota hai, tab loops kaam aate hain.

Example:

```
for (let i = 1; i <= 5; i++) {  
    console.log("JavaScript rocks!");  
}
```


String Concatenation - Strings Ko Jodo

Strings aur values ko jodne ke liye + operator ka use hota hai.

Example:

```
let name = "Shaaz";
```

```
console.log("Hello, " + name + "!");
```

```
// Output: Hello, Shaaz!
```


Strings

Strings ko Samajhna aur Declare karna

String is basically ek sequence of characters.

Isse hum single quotes ' ', double quotes " ", ya backticks ke beech likhte hain.

```
let firstName = 'Shaaz'; // Single quotes
```

```
let greeting = "Hello"; // Double quotes
```

```
let intro = `My name is ${firstName}`;
```

```
// Template Literal (advanced)
```

```
console.log(intro);
```

```
// Output: My name is Shaaz
```

Strings Print Karna

```
let name = "Shaaz";
```

```
console.log("Hello " + name); // Concatenation with +
```

```
console.log(`Hello ${name}`); // Concatenation with Template  
Literal
```

💡 Pro Tip: Template literals use backticks aur expressions ko `${ }` mein likhte hain. Yeh simple aur readable hota hai.

String Concatenation (Joining Strings)

Method 1: + Operator

```
let firstName = "Hello";  
let lastName = "World";  
console.log(firstName + " " + lastName); // Output: Hello  
World
```

Method 2: Template Literals

```
let name = "Shaaz";  
console.log(`Welcome, ${name}!`); // Output: Welcome,  
Shaaz!
```

Characters ko Access Karna

Strings are like arrays of characters, toh index use karke hum specific character ko access karte hain:

```
let word = "JavaScript";  
console.log(word[0]); // Output: J (index starts at 0)  
console.log(word.charAt(4));  
// Output: S (character at index 4)
```


Case Conversion (Uppercase/Lowercase)

Kabhi kisi input ka case change karna ho, toh yeh methods kaam aate hain:

```
let sentence = "Coding is FUN!";  
  
console.log(sentence.toLowerCase()); // Output: coding is  
fun!  
  
console.log(sentence.toUpperCase()); // Output: CODING  
IS FUN!
```

Strings Trim Karna (Unnecessary Spaces Hataana)

Kabhi string ke starting ya ending spaces ko hatana ho, toh `trim()` use karte hain:

```
let messyString = "  Hello World!  ";  
  
console.log(messyString.trim()); // Output: Hello World!
```

Substring Slicing (Ek Part Extract Karna)

Hum kisi string ka specific part extract kar sakte hain:

```
let text = "JavaScript";  
  
console.log(text.slice(0, 4)); // Output: Java (index 0 se 3  
take)
```

Splitting Strings (Parts Mein Todna)

String ko parts mein todne ke liye `split()` use karte hain:

```
let colors = "red,green,blue";  
  
console.log(colors.split(',')); // Output: ["red", "green", "blue"]
```


String Search (Finding Characters)

indexOf(): Pehli baar kisi character ya word ka index batata hai.

```
let word = "programming";
```

```
console.log(word.indexOf('g')); // Output: 3
```

lastIndexOf(): Last occurrence ka index batata hai.

```
console.log(word.lastIndexOf('g')); // Output: 10
```

includes(): Check karta hai ki string mein koi word ya character hai ya nahi.

```
console.log(word.includes('gram')); // Output: true
```

Repeating Strings

```
console.log("ha".repeat(3)); // Output: hahaha
```

String Mutability

Strings immutable hoti hain, iska matlab unhe directly modify nahi kar sakte. But variable ko nayi value assign kar sakte ho:

```
let str = "Hello";
```

```
str = "Shaaz";
```

```
console.log(str); // Output: Shaaz
```


Advanced Methods

startsWith() and endsWith()

Yeh check karte hain ki string kisi specific character ya word se shuru/end hoti hai ya nahi.

```
let word = "JavaScript";  
console.log(word.startsWith("Java")); // Output: true  
console.log(word.endsWith("Script")); // Output: true  
concat()
```

Strings ko join karne ka ek aur tareeka:

```
let part1 = "Hello";  
let part2 = "World";  
console.log(part1.concat(" ", part2)); // Output: Hello World
```


JavaScript Functions

Functions Kya Hote Hain?

Ek function ek block of code hota hai jo ek specific kaam karne ke liye likha jaata hai. Functions code ko modular aur reusable banate hain.

Function Declaration

```
function greet() {  
    console.log("Hello, Shaaz!"); // Task 1  
    console.log("Kaise ho?");    // Task 2  
}
```

Function Call `greet();`

// Output: Hello, Shaaz!

Kaise ho?

Explanation:

1. **function greet()** → Yeh function define kar raha hai.
2. **greet();** → Yeh function ko call kar raha hai, yani code ko execute kar raha hai.

Function Parameters aur Arguments

Parameters wo placeholders hote hain jo function definition mein use hote hain, aur arguments wo actual values hain jo function ko call karte waqt pass karte hain.

Example:

```
function greet(name) {  
  // 'name' is the parameter  
  console.log(`Hello, ${name}!`);  
}  
  
greet("Shaaz"); // "Shaaz" is the argument  
// Output: Hello, Shaaz!  
  
greet("Rahul");  
// Output: Hello, Rahul!
```

Function with Return Statement

Functions me hum return ka use karte hain output ko function ke bahar le jaane ke liye.

```
function square(x) {  
  return x * x;  
  // x ko square karke return karega  
}  
  
let result = square(5); // Result store hogaya  
console.log(result); // Output: 25
```

Tip: Agar return nahi likha, toh function automatically undefined return karega.

Function Expressions

Functions ko variables mein store karke bhi use kar sakte hain.

```
let greet = function(name) {  
    return `Hello, ${name}!`;  
};  
  
console.log(greet("Shaaz"));  
  
// Output: Hello, Shaaz!
```

Default Parameter Values

Agar koi argument pass na kare, toh hum default values set kar sakte hain.

```
function greet(name = "Guest") {  
    console.log(`Hello, ${name}!`);  
}  
  
greet(); // Output: Hello, Guest!  
greet("Shaaz"); // Output: Hello, Shaaz!
```

Function Without Parameters

Agar function me koi parameter na ho, toh bhi hum tasks perform kar sakte hain.

```
function sayHi() {  
    console.log("Hello, World!");  
}  
  
sayHi(); // Output: Hello, World!
```


Passing Arrays as Arguments

Arrays ko arguments ke roop me pass kar sakte hain:

```
function sumOfTwoNumbers(numbers) {  
    return numbers[0] + numbers[1];  
}  
  
let myNumbers = [10, 20];  
console.log(sumOfTwoNumbers(myNumbers));  
  
// Output: 30
```

Functions with Unlimited Parameters

Agar hume unknown number of arguments pass karne hain, toh hum arguments object use karte hain.

```
function sumAll() {  
    let sum = 0;  
    for (let i = 0; i < arguments.length; i++) {  
        sum += arguments[i];  
    }  
    return sum;  
}  
  
console.log(sumAll(1, 2, 3, 4, 5));  
  
// Output: 15
```


Arrow Functions

Arrow functions short syntax provide karte hain:

Example 1: Single Statement

```
const square = (x) => x * x;
```

```
console.log(square(5));
```

```
// Output: 25
```

Example 2: Multiple Statements

```
const greet = (name) => {  
    console.log(`Hello, ${name}!`);  
    return `How are you, ${name}?`;  
};
```

```
console.log(greet("Shaaz"));
```

```
// Output:
```

```
Hello, Shaaz!
```

```
How are you, Shaaz?
```

Example 3: Object Return

```
const calculate = (x, y) => ({ sum: x + y, diff: x - y });
```

```
console.log(calculate(7, 3));
```

```
// Output: { sum: 10, diff: 4 }
```


JavaScript Arrays

Ek array ek box jaisa hai, jisme hum multiple cheezein (values) ek saath store kar sakte hain. Values alag-alag types ki ho sakti hain—numbers, strings, booleans, objects, aur even doosre arrays!

Why do we need Array ?

1. **Group Similar Data:** Ek saath similar cheezon ko store karna easy ho jata hai.
2. **Handle Data:** Bahut saara data ek variable mein store kar sakte ho.
3. **Enhance Performance:** JavaScript arrays ko handle karne ke liye optimize hai.
4. **Convenient methods:** Array ke saath kaam karne ke liye bahut saare built-in methods milte hain.
5. **Readable code:** Code dekhne aur samajhne mein asaan lagta hai.

How to make an Array ?

Square Brackets Method

```
let arr = [1, 2, 3, "Shaaz", false];  
  
console.log(arr);
```

// Output: [1, 2, 3, "Shaaz", false]

Simple! Square brackets ke andar values likho, bas array ban gaya.

Array Constructor Method

```
let arr2 = new Array(1, 2, 3, "Shaaz", false);  
  
console.log(arr2);
```

// Output: [1, 2, 3, "Shaaz", false]

Par, zyada log square brackets wala method use karte hain kyunki yeh short aur clean hota hai.

How to Access Array Element?

Array ke har element ka ek index hota hai, jo 0 se start hota hai.

```
let arr3 = ["Shaaz", "Laptop", 23, "Mobile"];
```

// Ek specific index ka value lena

```
console.log(arr3[2]);
```

// Output: 23 (index 2 ka element)

```
arr3[3] = 78;
```

// Ek element ko change kar diya

```
console.log(arr3);
```

// Output: ["Shaaz", "Laptop", 23, 78]// index doesn't exist

```
console.log(arr3[8]);
```

// Output: undefined

Common Array Methods

1. push() aur pop()

push() se array ke end mein element add hota hai.

pop() se array ka last element remove ho jata hai.

```
let newArr = [1, 2, 3, 4, 5];
```

```
newArr.push(29); // End mein add kiya
```

```
console.log(newArr);
```

```
// Output: [1, 2, 3, 4, 5, 29]
```

```
newArr.pop(); // Last element hata diya
```

```
console.log(newArr);
```

```
// Output: [1, 2, 3, 4, 5]
```

2. shift() aur unshift()

shift() se first element remove hota hai.

unshift() se array ke start mein element add karte hain.

```
newArr.shift(); // Pehla element hata diya
```

```
console.log(newArr);
```

```
// Output: [2, 3, 4, 5]
```

```
newArr.unshift(99, 100); // Starting mein elements add kiye
```

```
console.log(newArr);
```

```
// Output: [99, 100, 2, 3, 4, 5]
```


3. concat()

Do ya zyada arrays ko merge karne ke liye.

```
let a1 = [2, 3, 4, 5];
```

```
let a2 = [9, 12, 15];
```

```
let a3 = a1.concat(a2);
```

```
console.log(a3);
```

```
// Output: [2, 3, 4, 5, 9, 12, 15]
```

4. join()

Array ke elements ko ek string mein convert karta hai, specified separator ke saath.

```
let a4 = a3.join("");
```

```
console.log(a4);
```

```
// Output: "234591215"
```

```
let a5 = a3.join("@");
```

```
console.log(a5);
```

```
// Output: "2@3@4@5@9@12@15"
```

5. reverse()

Array ke elements ka order ulta kar deta hai.

```
let arr6 = [88, 22, 34, 12, 45];
```

```
arr6.reverse();
```

```
console.log(arr6);
```

```
// Output: [45, 12, 34, 22, 88]
```


6. indexOf()

Kisi element ka first index find karta hai.

```
let arr7 = [90, 225, 134, 172, 405];
```

```
console.log(arr7.indexOf(172));
```

```
// Output: 3
```

7. slice()

Array ke ek portion ko naya array banakar deta hai.

```
let arr8 = [45, 23, 85, 7, 6, 98];
```

```
console.log(arr8.slice(2, 5));
```

```
// Output: [85, 7, 6]
```

8. splice()

Array mein add ya remove karne ke liye use hota hai.

```
let arr9 = [89, 65, 23, 78, 98];
```

```
// Add karna
```

```
arr9.splice(2, 0, 11);
```

```
console.log(arr9);
```

```
// Output: [89, 65, 11, 23, 78, 98]// Replace karna
```

```
let arr10 = [23, 76, 45, 32, 98];
```

```
arr10.splice(3, 1, 888);
```

```
console.log(arr10);
```

```
// Output: [23, 76, 45, 888, 98]
```


map(), filter(), and reduce()

map() Method

map() ka kaam hai har element ko ek nayi shape dena aur ek naya array banakar wapas dena. Yeh original array ko change nahi karta, bas ek naye array mein transformed values de deta hai.

Example:

```
let numbers = [1, 2, 3, 4, 5];
```

```
let doubledNumbers = numbers.map((element) => {  
  return element * 2;  
});
```

```
console.log(doubledNumbers);
```

```
// Output: [2, 4, 6, 8, 10]
```

Explanation?

1. Ek array hai: [1, 2, 3, 4, 5].
2. map() method har element pe ek function apply karta hai.
 - Yahaan har element ko ***2** kar diya.
3. Result ek naya array ban gaya: [2, 4, 6, 8, 10].

filter() Method

filter() ek array ke andar ghuskar sirf un elements ko chhanta hai jo condition ko pass karte hain. Jo fail karein, unhe ignore karta hai.

Example:

```
let numbers = [1, 2, 3, 4, 5];
```

```
let filteredArray = numbers.filter((element) => {  
  if (element % 2 === 0) {  
    return element;  
  }  
});  
console.log(filteredArray);
```

// Output: [2, 4]

Explanation ?

1. Ek array hai: [1, 2, 3, 4, 5].
2. filter() method har element ko check kar raha hai:
 - Agar element even hai (% 2 === 0), toh nayi array mein add ho jayega.
3. Result nayi array ban gayi: [2, 4] (odd numbers ko ignore kar diya).

reduce() Method

reduce() ek recipe ki tarah kaam karta hai, jo har element ko process karta hai aur ek single output deta hai. Iska use tab hota hai jab sab elements ko combine karna ho.

Example:

```
let numbers = [1, 2, 3, 4, 5];
```

```
let sum = numbers.reduce((accumulator, currentValue) => {  
    return accumulator + currentValue;  
}, 0);
```

```
console.log(sum);
```

```
// Output: 15
```

Explanation ?

1. Ek array hai: [1, 2, 3, 4, 5].

2. reduce() method ke andar ek function hai:

- accumulator ka kaam hai result ko store karna.
- currentValue har baar array ka current element hota hai.

3. Shuru se lekar end tak:

- $0 + 1 = 1$
- $1 + 2 = 3$
- $3 + 3 = 6$
- $6 + 4 = 10$
- $10 + 5 = 15$

4. Final result: 15.

forEach() Method

forEach() ka kaam hai ek array ke har element ko individually process karna.

Bas ye ek loop ki tarah kaam karta hai, lekin zyada readable aur short hota hai.

Dhyan do:

- forEach() kuch return nahi karta, iska kaam sirf process karna hai.

Syntax:

```
array.forEach((element, index, array) => {
```

```
// Code to execute on each element
```

```
});
```

- **element**: Current element jo process ho raha hai.
- **index**: (Optional) Current element ka index (position in array).
- **array**: (Optional) Pura array jisme se elements aa rahe hain.

Example 1: Simple Use Case

```
let numbers = [1, 2, 3, 4, 5];
```

```
numbers.forEach((element) => {
```

```
  console.log(element * 2);
```

```
});
```

```
// Output:// 2// 4// 6// 8// 10
```

Explanation ?

1. Array [1, 2, 3, 4, 5] ke har element ko forEach() method process kar raha hai.
2. Har element *2 ho raha hai aur result console pe print ho raha hai.
3. Bas, iska kaam khatam! Return kuch nahi karta.

Example 2: Index ka Use

```
let fruits = ["Apple", "Banana", "Cherry"];
```

```
fruits.forEach((fruit, index) => {  
    console.log(`Index ${index}: ${fruit}`);  
});
```

// Output:

// Index 0: Apple

// Index 1: Banana

// Index 2: Cherry

Kya Seekha?

- Har element ke saath uska index bhi print kar sakte ho.
- Useful jab tumhe position track karni ho

JavaScript Object

JavaScript ka object ek real-world cheez ki tarah hota hai jisme properties (state) aur methods (behavior) hote hain.

- **Example:** Ek car ka socho:
 - **Properties:** Color, brand, speed.
 - **Methods:** Start karna, stop karna, accelerate karna.

Key Points about Objects

1. **Multiple Values Store Karna:** Objects ek naam ke andar multiple values store karne ka option dete hain (key-value pairs ke form mein).
2. **Example:** Ek person ke naam, age aur city ko ek saath store karna.
3. **Readable Aur Organized Code:** Variables se better hai kyunki objects ka structure clean aur samajhne layak hota hai.

How to make an Object ?

Object Literal (Sabse Easy Way)

Sabse simple aur zyada use hone wala tarika.

```
let person = {
```

```
  id: 1,
```

```
  name: "Shaaz",
```

```
  age: 23
```

```
};
```

// Access karna:

```
console.log(person.name); // Output: Shaaz
```

```
console.log(person["age"]); // Output: 23
```

Note:

- Dot notation zyada common aur clean lagta hai.
- Bracket notation tab use hota hai jab key dynamic ho ya ek string ho.

Object Constructor (Thoda Detailed Way)

Ek khaali object banao aur properties ko baad mein add karo.

```
let car = new Object();
```

```
car.brand = "Toyota";
```

```
car.color = "Red";
```

```
car.speed = 120;
```

```
console.log(car);
```

// Output: { brand: 'Toyota', color: 'Red', speed: 120 }

Constructor Function (Reusability ke Liye Best)

Agar multiple objects create karne hain, toh constructor function ka use karte hain.

```
function Person(name, age, city) {  
  this.name = name;  
  this.age = age;  
  this.city = city;  
}  
  
let person1 = new Person("Shaaz", 23, "Delhi");  
let person2 = new Person("Aman", 25, "Mumbai");  
console.log(person1);  
  
// Output: { name: 'Shaaz', age: 23, city: 'Delhi' }  
  
console.log(person2);  
  
// Output: { name: 'Aman', age: 25, city: 'Mumbai' }
```

Object Properties ke Saath Kya-Kya Kar Sakte Hain?

Add/Update Properties

```
let laptop = { brand: "HP", price: 50000 };  
  
laptop.ram = "16GB";    // Add new property  
  
laptop.price = 45000;    // Update existing property  
  
console.log(laptop);  
  
// Output: { brand: 'HP', price: 45000, ram: '16GB' }
```

Delete Property

```
delete laptop.ram;  
  
console.log(laptop);  
  
// Output: { brand: 'HP', price: 45000 }
```


Object Methods: Important Cheezein

Object.keys(): Saari keys (property names) ka array laata hai.

```
let car = { brand: "Honda", color: "Blue", speed: 150 };
```

```
console.log(Object.keys(car));
```

```
// Output: [ 'brand', 'color', 'speed' ]
```

Object.values(): Saari values ka array laata hai.

```
console.log(Object.values(car));
```

```
// Output: [ 'Honda', 'Blue', 150 ]
```

Object.entries(): Key-value pairs ko ek nested array ke form mein laata hai.

```
console.log(Object.entries(car));
```

```
// Output: [ ['brand', 'Honda'], ['color', 'Blue'], ['speed', 150] ]
```

Object.assign(): Ek object ko dusre object ke saath merge karna ya copy karna.

```
let obj1 = { name: "Shaaz", age: 23 };
```

```
let obj2 = { city: "Delhi", profession: "Developer" };
```

```
let mergedObj = Object.assign({}, obj1, obj2);
```

```
console.log(mergedObj);
```

```
// Output: { name: 'Shaaz', age: 23, city: 'Delhi', profession: 'Developer' }
```


Object Immutability: Object.freeze() vs Object.seal()

Object.freeze():

Object ko completely unchangeable bana deta hai.

```
let obj = { name: "Shaaz", age: 23 };
```

```
Object.freeze(obj);
```

```
obj.age = 25; // Error (strict mode) or ignored
```

```
obj.city = "Delhi"; // Error (strict mode) or ignored
```

```
console.log(obj);
```

```
// Output: { name: 'Shaaz', age: 23 }
```

Object.seal():

Kewal properties ko update kar sakte ho, naye properties add/delete nahi kar sakte.

```
let obj = { name: "Shaaz", age: 23 };
```

```
Object.seal(obj);
```

```
obj.age = 25; // Allowed
```

```
obj.city = "Delhi"; // Ignored console.log(obj);
```

```
// Output: { name: 'Shaaz', age: 25 }
```


Storing Data in Browser

Local Storage – Permanent Memory for Browser

Samjho ki tumhare paas ek digital locker hai jo browser ke andar hai. Yahan pe tum data save kar sakte ho aur tab tak rak sakte ho jab tak tum manually delete na karo ya browser clear na kare. Ye server se baar-baar data fetch karne ki zaroorat ko kam karta hai, jo performance aur speed dono badhata hai.

Example: Tumhari ek shopping website hai, aur tum chahte ho ki users ke favorite items hamesha store rahein, toh tum Local Storage ka use karoge.

How to Use Local Storage?

1. Storing & Retrieving Simple Values

```
localStorage.setItem("game", "cricket");  
console.log(localStorage.getItem("game"));  
localStorage.setItem("username", "ShaazAhmad");  
localStorage.setItem("username", "ShaazAkhtar");  
karnallocalStorage.removeItem("username");  
localStorage.clear();
```

Key Point: Local Storage me string format me data store hota hai. Isme sirf same-origin wale pages access kar sakte hain.

2. Storing & Retrieving Objects (Updated Method)

Agar tumhe objects ya arrays store karne hain, toh pehle usko **JSON.stringify()** se string me convert karna padega, aur jab retrieve karoge tab **JSON.parse()** ka use karna padega.

```
let user = {  
  name: "Shaaz",  
  age: 23,  
  city: "Delhi",  
};
```

```
localStorage.setItem("user", JSON.stringify(user));
```

```
let storedUser = JSON.parse(localStorage.getItem("user"));  
console.log(storedUser.name); // Output: "Shaaz"
```

Pro Tip: Hamesha JSON.parse() ka use karo jab Local Storage se object ya array retrieve karna ho.

Cookies – Small but Powerful

Cookies ekdum chhoti memory chip ki tarah hoti hain jo browser aur server ke beech ka bridge hoti hain. Ye har request ke saath server ko send hoti hain aur mostly login ya user preferences store karne ke kaam aati hain.

Example: Tumne kisi shopping website pe "Remember Me" option dekha hoga? Ye cookies ka use karke implement hota hai.

How to Use Cookies?

1. Set a Cookie

```
document.cookie = "username=Shaaz; expires=Fri, 31 Dec 2024 23:59:59 GMT";
```

Explanation:

- "username=Shaaz" → Key-Value Pair me store hota hai.
- "expires=..." → Expiry date set ki jati hai, taki data tab tak save rahe.

2. Get a Cookie

Cookies ko directly JavaScript me access kar sakte ho:

```
console.log(document.cookie); // Output: "username=Shaaz"
```

3. Delete a Cookie

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC";
```

Explanation: Expiry date ko past me set kar do, toh browser automatically delete kar dega.

Session Storage – Browser Close Karte Hi Delete Ho Jaata Hai

Ye Local Storage ki tarah hi hai, lekin temporary hai. Jab tak browser tab open hai tab tak ye data store rahta hai, jaise hi close karoge, saara data udd jayega!

Example: Tum ek online exam de rahe ho, aur tumhara current question number store karna hai jab tak exam chal raha hai. Jaise hi tab close karoge, data delete ho jayega.

How to Use Session Storage?

1. Storing & Retrieving Data

```
sessionStorage.setItem("sessionKey", "sessionValue");  
console.log(sessionStorage.getItem("sessionKey"));  
// Output: "sessionValue"  
sessionStorage.clear();
```

Use Case: Session Storage tab use hota hai jab tumhe temporary data store karna ho jo browser band hote hi delete ho jaye.

Session Storage – Browser Close Karte Hi Delete Ho Jaata Hai

Ye Local Storage ki tarah hi hai, lekin temporary hai. Jab tak browser tab open hai tab tak ye data store rahta hai, jaise hi close karoge, saara data udd jayega!

Example: Tum ek online exam de rahe ho, aur tumhara current question number store karna hai jab tak exam chal raha hai. Jaise hi tab close karoge, data delete ho jayega.

How to Use Session Storage?

1. Storing & Retrieving Data

```
sessionStorage.setItem("sessionKey", "sessionValue");  
console.log(sessionStorage.getItem("sessionKey"));  
// Output: "sessionValue"  
sessionStorage.clear();
```

Use Case: Session Storage tab use hota hai jab tumhe temporary data store karna ho jo browser band hote hi delete ho jaye.

Understanding the DOM

DOM Kya Hai?

Maan lo ek shandaar ghar bana rahe ho jisme alag-alag rooms hain (HTML elements).

- HTML pura ghar hai
- Rooms, windows, aur doors uske andar ke elements hain
- JavaScript tumhara haath hai jo in rooms ko modify kar sakta hai

Ek aur example lo, DOM ek tree structure hai, jisme:

- Root node hota hai `<html>`
- Iske andar branches hoti hain (`<head>`, `<body>`, etc.)
- Aur phir chhoti-chhoti leaves hoti hain (`<h1>`, `<p>`, `<button>` etc.)

JavaScript ka kaam hai is tree ke kisi bhi part ko change karna ya access karna!

Example: Basic HTML Structure

Maan lo tumhare paas ek simple web page hai:

```
<!DOCTYPE html>

<HTML>

<head>

<title>DOM Example</title>

</head>

<body>

<h1 id="main-heading">Hello, DOM!</h1>

<p class="text">This is a paragraph.</p>

<button id="myButton">Click Me</button>

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

</body>

</html>
```

Ye ek basic structure hai, ab JavaScript ke through isko manipulate karna seekhenge!

1. DOM Elements Ko Select Karna

Bhai, sabse pehle kisi bhi element ko pakadna zaroori hai, tabhi toh usme changes kar paoge! Tum different methods ka use kar sakte ho:

Query Selectors

// ID se select karna

```
let heading = document.getElementById("main-heading");
```

```
console.log(heading.innerText);
```

// Output: "Hello, DOM!"

// Class se select karna

```
let para = document.querySelector(".text");
```

// Sirf pehli matching class milegi

```
console.log(para.innerText);
```

// Multiple elements select karna

```
let allParas = document.querySelectorAll(".text");
```

// Sabhi matching elements milenge

```
console.log(allParas);
```

Things to remember:

- `getElementById()` sirf ek element return karta hai.
- `querySelector()` bhi ek hi element dega, but CSS selectors ka use karta hai.
- `querySelectorAll()` sabhi matching elements ka NodeList return karega.

2. Event Listeners (User Actions Ko Handle Karna)

Maan lo tum chahte ho ki jab koi button click kare, toh kuch ho. Toh Event Listeners ka use hota hai!

```
// Button select karna
```

```
let button = document.getElementById("myButton");
```

```
// Event listener lagana
```

```
button.addEventListener("click", function () {  
    alert("Button was clicked!");  
});
```

Explained:

- `addEventListener()` ek function hai jo user ke actions ko sunta hai (like click, keypress etc.).
- Jab button pe click hoga, tab alert box dikhayega!

3. DOM Elements Ko Style Karna

Tum JavaScript se directly CSS properties change kar sakte ho!

```
heading.style.color = "blue"; // Text blue ho jayega
```

```
heading.style.fontSize = "30px"; // Font size badh jayega
```

```
heading.style.backgroundColor = "yellow";
```

```
// Yellow background aayega
```

Yeh CSS ka alternative hai, but hamesha classes use karna best practice hota hai!

4.CSS Classes Ko Add, Remove & Toggle Karna

Maan lo tumhe kisi element ka style dynamically change karna hai toh `classList` use kar sakte ho.

```
// Class add karna
```

```
heading.classList.add("highlight");
```

```
// Class remove karna
```

```
heading.classList.remove("highlight");
```

```
// Class toggle karna (Agar hai toh hata dega, agar nahi hai toh  
add karega)
```

```
heading.classList.toggle("highlight");
```

Example Use Case:

Maan lo tum dark mode toggle kar rahe ho, toh tum `classList.toggle()` use kar sakte ho!

5. Naya Element Banana & Insert Karna

Maan lo tum chahte ho ki JavaScript se ek naya paragraph add ho page pe.

```
// Naya element banana
```

```
let newPara = document.createElement("p");
```

```
// Usme text dalna
```

```
newPara.innerHTML = "This is a new paragraph added with  
JavaScript!";
```

```
// Isko page pe insert
```

```
karnadocument.body.appendChild(newPara);
```

Explained:

- `createElement()` naya HTML element banata hai.
- `innerHTML` se uske andar text add kiya.
- `appendChild()` se HTML ke andar add kar diya!

6. DOM Se Elements Ko Hatana

Agar tum kisi element ko DOM se delete karna chahte ho, toh `remove()` method ka use karo.

```
// Paragraph select karo
```

```
let paraToRemove = document.querySelector(".text");
```

```
// Usko hatao
```

```
paraToRemove.remove();
```

Explained:

- `.remove()` directly element hata deta hai page se!

Real-World Example: Dark Mode Toggle

Agar tum dark mode ka button banana chahte ho jo light aur dark mode toggle kare, toh ye code likho:

```
let btn = document.getElementById("myButton");
```

```
let body = document.body;
```

```
btn.addEventListener("click", function () {  
    body.classList.toggle("dark-mode");  
});
```

Aur CSS me ek dark mode class bana lo:

```
.dark-mode {  
    background-color: black;  
    color: white;  
}
```

Bas! Ek button dabao aur light/dark mode switch ho jayega!

Asynchronous JavaScript

Synchronous vs Asynchronous Code

Synchronous (Blocking)

Ek kaam jab tak complete nahi hota, tab tak doosra nahi chalega. Sab kuch line-by-line execute hoga.

Example: Maan lo tum ek restaurant me ho aur ek waiter ek time pe sirf ek order lega.

```
console.log("A: Pizza order kiya...");  
console.log("B: Waiter pizza bana raha hai...");  
console.log("C: Pizza ready!");
```

Problem? Agar ek kaam time leta hai, toh pure script ka execution ruk jata hai.

Asynchronous (Non-Blocking)

Ek task background me chalta rahega, aur baaki kaam rukenge nahi.

Example: Tum buffet system me ho, jisme sab apna khana khud le rahe hain.

```
console.log("A: Pizza order kiya...");  
setTimeout(() => {  
    console.log("B: Pizza ready ho gaya!");  
}, 2000);  
console.log("C: Burger order kiya...");
```


Yahan kya ho raha hai?

1. "A: Pizza order kiya.." turant execute ho gaya.
2. `setTimeout()` ke andar ka code 2 sec ke liye wait karega, lekin tab tak "C: Burger order kiya.." turant execute ho jayega.
3. 2 sec baad "B: Pizza ready ho gaya!" print hoga.

Key Takeaway: JavaScript single-threaded hoti hai, but asynchronous features ka use karke non-blocking kaam kar sakti hai!

How JavaScript Handles Asynchronous Code?

JavaScript me asynchronous kaam karne ke 3 main tarike hain:

Callbacks (Purana tareeka)

Promises (Better way)

Async/Await (Sabse best aur modern)

Callbacks – Old Way (Messy)

Callback ek function ke andar ek aur function pass karta hai jo baad me execute hoga.

Example: Callback Hell (Messy Code)

```
function orderPizza(callback) {  
  console.log("🍕 Pizza order kiya...");  
  setTimeout(() => {  
    console.log("✅ Pizza ready!");  
    callback(); // Dusra kaam tab hoga jab ye complete hoga  
  }, 2000);  
}  
  
function eatPizza() {  
  console.log("😋 Pizza kha raha hoon...");  
}  
  
orderPizza(eatPizza);
```

Problem? Agar multiple callbacks ho gaye toh Callback Hell create ho jata hai!

```
orderPizza(() => {  
  makeBurger(() => {  
    orderCoffee(() => {  
      eatFood();  
    });  
  });  
});  
}); //unreadable code
```


Promises – The Better Way

Promise ek commitment ki tarah hai—ya toh resolve (kaam ho gaya) ya reject (kaam fail ho gaya).

Basic Example:

```
let pizzaPromise = new Promise((resolve, reject) => {  
  console.log("🍕 Pizza order kiya...");  
  setTimeout(() => {  
    let success = true; // Try `false` to test rejection  
    if (success) {  
      resolve("✅ Pizza ready!");  
    } else {  
      reject("❌ Pizza jal gaya!");  
    }  
  }, 2000);  
});
```

// Promise ko handle karna

pizzaPromise

```
.then((message) => console.log(message)) // when resolve  
.catch((error) => console.log(error)); // When Reject
```

Things to remember?

- `resolve()` → Jab kaam ho jaye.
- `reject()` → Jab koi error aaye.
- `.then()` → Jab promise complete ho jaye.
- `.catch()` → Jab koi error ho.

Advantage: Callback Hell avoid ho gaya, code readable ho gaya!

Async/Await – The Best & Cleanest

Promises ka aur bhi easy aur clean version! 🔥

Example:

```
function orderPizza() {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      resolve("✅ Pizza ready!");  
    }, 2000);  
  });  
}
```

```
async function serveFood() {  
  console.log("🍕 Pizza order kiya...");  
  let pizza = await orderPizza();  
  console.log(pizza);  
  console.log("🍔 Burger order kiya...");  
}
```

```
serveFood();
```

Things to remember ?

- async function ke andar await ka use hota hai jo promise complete hone tak wait karega.
- **.then()** ki zaroorat nahi, code readable ho gaya!

Best Practice: Jab bhi asynchronous code likhna ho, toh async/await ka use karo! ✅

Some More Advanced Promise Concepts

Promise.all() – Parallel Execution

Agar multiple promises ek saath chalane ho, toh Promise.all() use karo.

```
let pizza = new Promise((resolve) => setTimeout(() =>
resolve("🍕 Pizza"), 2000));
```

```
let burger = new Promise((resolve) => setTimeout(() =>
resolve("🍔 Burger"), 1000));
```

```
Promise.all([pizza, burger]).then((foods) => {
  console.log("Sab kuch ready:", foods);
});
```

// Output (after 2 sec): Sab kuch ready: ['🍕 Pizza', '🍔 Burger']

Things to remember?

- Saare promises ek saath start ho gaye.
- Jab sab complete ho gaye, tab ek saath result mil gaya!

Promise.race() – Fastest Promise Wins

Agar multiple requests me sirf jo pehle aaye wahi chahiye, toh Promise.race() use karo.

```
let fast = new Promise((resolve) => setTimeout(() =>
resolve(" Fastest!"), 1000));
```

```
let slow = new Promise((resolve) => setTimeout(() =>
resolve("🐌 Slow!"), 3000));
```

```
Promise.race([fast, slow]).then((winner) => {
  console.log("Winner:", winner);
});
```

// Output (after 1 sec): Winner: 🚀 Fastest!

Things to remember ?

- Jo sabse fast complete hoga, wahi return ho jayega!

Scope in JavaScript

Scope ka matlab hai "ek variable kahaan tak dikh sakta hai aur use ho sakta hai". JavaScript me 3 tareeke ke scopes hote hain:

Global Scope – Pura code access kar sakta hai.

Function Scope – Sirf ek function ke andar accessible hota hai.

Block Scope – {} ke andar defined hota hai (let & const ke saath).

Example:

Maan lo tumhare Ghar (Global Scope) me ek Locker (Variable) hai.

- Tumhare family ke sab log locker use kar sakte hain (Global Scope).
- Tumhare Papa ka ek private locker hai jo sirf unko access hai (Function Scope).
- Tumhare room ke andar ek chhota locker hai jo sirf tum use kar sakte ho (Block Scope).

Chalo, ab isko code se samajhte hain!

Global Scope (Sabko Dikhega)

Global Scope me jo variable declare hota hai, usko pura JavaScript code access kar sakta hai.

```
let globalVar = "🌍 I am a global variable";
```

```
function showGlobal() {  
    console.log(globalVar); // ✅ Accessible  
}
```

```
showGlobal();
```

```
console.log(globalVar); // ✅ Accessible
```

Yahaan globalVar har jagah accessible hai!

Function Scope (Sirf Function Ke Andar)

Function ke andar declare kiya gaya variable sirf usi function me access ho sakta hai.

```
function parentFunction() {  
    let parentVar = "😬 I am a parent variable";  
    console.log(parentVar); // ✅ Accessible inside function  
}
```

```
parentFunction();
```

```
console.log(parentVar); // ❌ ERROR! parentVar is not  
defined
```

Problem? parentVar function ke bahar access nahi ho sakta!

Block Scope (let & const wale)

Agar let ya const use karte ho, toh woh sirf usi {} block ke andar accessible hota hai.

```
{
```

```
  let blockVar = "🔒 I exist only inside this block";
```

```
  console.log(blockVar); // ✅ Accessible
```

```
}
```

```
console.log(blockVar); // ❌ ERROR! blockVar is not defined
```

⚡ var block scope follow nahi karta, woh function scope follow karta hai!

```
{
```

```
  var blockVar2 = "😈 I can escape the block!";
```

```
}
```

```
console.log(blockVar2); // ✅ Accessible! (VAR se bacha rehna)
```

👉 Best Practice: Hamesha let aur const ka use karo, var se door raho!

Scope Chain & Lexical Environment

Lexical Environment – Variables Stored Hote Hain

JavaScript me har function apna ek alag memory space create karta hai, jisme uske variables store hote hain. Isko Lexical Environment kehte hain.

Scope Chain – JavaScript Variables Ko Kaise Dhundta Hai

Jab JavaScript kisi variable ko access karne ki koshish karti hai, toh ye pehle apne function me dhoondhti hai, fir parent function me, aur agar waha bhi nahi mila toh global scope me dekhti hai.

```
let globalVar = "🌍 I am global";
```

```
function parentFunction() {
```

```
    let parentVar = "👤 I am a parent variable";
```

```
    function childFunction() {
```

```
        let childVar = "👶 I am a child variable";
```

```
        console.log(childVar); // ✅ Available (Child Scope)
```

```
        console.log(parentVar); // ✅ Available (Parent Scope)
```

```
        console.log(globalVar); // ✅ Available (Global Scope)
```

```
    }
```

```
    childFunction();
```

```
}
```

```
parentFunction();
```


Things to remember ?

1. **childFunction()** ke andar sabse pehle apne variables check honge.
2. Agar koi variable waha nahi mila, toh ye **parentFunction()** me check karega.
3. Agar waha bhi nahi mila, toh ye global scope me jayega.
4. Agar kisi bhi scope me nahi mila, toh Error aayega.

Parent Function Cannot Access Child Function's Variable

```
function parentFunction() {  
  function childFunction() {  
    let childVar = "🤖 I am a child";  
  }  
  console.log(childVar); // ❌ ERROR! childVar is not defined  
}
```

👉 JavaScript me child function ka variable parent function me access nahi hota!

What is a Closure?

Simple Definition:

Closure ek function ka ability hota hai ki wo apne parent function ke variables ko yaad rakhe, even after parent function execute ho chuka ho.

Ek function + uska lexical scope = Closure

Real-Life Example:

Maan lo tum ek bank locker ka access le rahe ho.

- Locker (parent function) band ho gaya, but tumhare paas ab bhi chabi hai (closure function).
- Tum hamesha us locker ko open kar sakte ho, even after original owner chala gaya!

Example 1: Basic Closure

```
function parent() {
```

```
    let parentVar = "👤 I am a parent variable"; // Parent  
scope variable
```

```
function child() {
```

```
    console.log(parentVar); // Child function ko parentVar  
accessible hai
```

```
}
```

```
    return child; // Child function ko return kar rahe hain
```

```
}
```

```
let childFunction = parent(); // Parent function execute ho  
gaya
```

```
childFunction(); // Output: "👤 I am a parent variable"
```

Things to remember ?

1. parent() execute ho gaya, lekin parentVar delete nahi hua.
2. childFunction ke paas parentVar ka reference still available hai, isi ko closure kehte hain!
3. Closure ka magic ye hai ki child function ko parent function ke variables access karne ka power milta hai! 🚀

Example 2: Closure with Counter

```
function createCounter() {  
    let count = 0; // Private variable  
    return function () {  
        count++;  
        console.log(`Count: ${count}`);  
    };  
}
```

```
let counter = createCounter();  
counter(); // Output: Count: 1  
counter(); // Output: Count: 2  
counter(); // Output: Count: 3
```

Things to remember ?

- count variable parent function ke andar hai, but child function usko modify kar sakta hai!
- Jab bhi counter() call hota hai, count value increase hoti hai, even after parent function execute ho chuka hai!

👉 Yahi closure ka main use hai – private variables banana jo sirf specific functions access kar sakein!

Example 3: Closures in SetTimeout (Async Closures)

```
function delayMessage(msg, delay) {  
    setTimeout(function () {  
        console.log(msg);  
    }, delay);  
}  
  
delayMessage("🚀 Hello after 2 seconds!", 2000);
```

Things to remember ?

- setTimeout() ke andar jo function hai, wo closure bana leta hai jo msg ko yaad rakhta hai.
- 2 second baad bhi msg variable exist karta hai, aur print hota hai!

Example 4: Function Factory (Multiple Closures)

Maan lo tum ek function create kar rahe ho jo different discount percentages return kare.

```
function discountCreator(discount) {  
    return function (price) {  
        return price - (price * discount) / 100;  
    };  
}  
  
let tenPercentDiscount = discountCreator(10);  
let twentyPercentDiscount = discountCreator(20);  
  
console.log(tenPercentDiscount(1000)); // Output: 900  
console.log(twentyPercentDiscount(1000)); // Output: 800
```

Things to remember ?

- discountCreator() ek closure bana raha hai jo discount value store karta hai.
- Har function ka apna apna lexical environment hai!

👉 Closures ko smart tareeke se use karke code ko reusable bana sakte ho!

Call, Apply and Bind

Problem Kya Hai?

Normal cases me, jab tum ek object ke andar function banate ho, toh this usi object ko refer karta hai. Lekin kabhi kabhi tum chahte ho ki kisi aur object ka method kisi doosre object ke liye use ho jaye.



Example:

Tumhare do dost hain – Shaaz aur Asad.

- Shaaz ke paas ek method hai jo full name print karta hai.
- Ab tumhe chahiye ki yehi method Asad ke liye bhi chale.
- Yahi kaam Call, Apply aur Bind karte hain!

Call() Method – Function Ko Turant Call Karo

Working of Call:

- call() function turant execute hota hai.
- this ka reference badalne ke liye call() ka use hota hai.
- Arguments ek-ek karke pass karne hote hain.

Example:

```
const person1 = {  
  fName: "Shaaz",  
  lName: "Akhtar",  
  printFullName: function (hometown, country) {  
    console.log(this.fName + " " + this.lName + " from " +  
hometown + ", " + country);  
  }  
};
```

```
const person2 = {  
  fName: "Asad",  
  lName: "Ahmad"  
};
```

// person1 ka method use karke person2 ka full name print karna

```
person1.printFullName.call(person2, "Bihar", "India");
```

// Output: Asad Ahmad from Bihar, India

Things to remember ?

- **person1.printFullName()** method originally sirf Shaaz ke liye tha.
- **call()** ka use karke yehi method person2 (Asad) ke liye bhi kaam kar raha hai!
- Extra parameters ("Bihar", "India") bhi as arguments pass ho rahe hain! ✅

💡 Call ka simple formula:

```
functionName.call(object, arg1, arg2, ...);
```


Apply() Method – Arguments as Array

Apply:

- call() aur apply() me sirf ek difference hai:
 - call() me arguments alag-alag pass hote hain.
 - apply() me arguments ek array ke andar pass hote hain.
- apply() useful hota hai jab arguments pehle se ek array me stored ho.

Example:

```
person1.printFullName.apply(person2, ["Bihar", "India"]);
```

```
// Output: Asad Ahmad from Bihar, India
```

Things to remember ?

- call() aur apply() ka result same hai, bas arguments pass karne ka tareeka alag hai!

Apply ka simple formula:

```
functionName.apply(object, [arg1, arg2, ...]);
```


Bind() Method – Function Store Karo, Baad Me

Call Karo

Bind ka kaam:

- bind() immediately function ko call nahi karta, balki ek naye function ko return karta hai.
- Jab tumhe future me function execute karna ho, tab bind() ka use hota hai!
- Isko ek variable me store karke baad me call kar sakte ho.

Example:

```
const result = person1.printFullName.bind(person2, "Bihar", "India");
```

```
console.log(result); // Output: [Function: bound printFullName] (Function return ho gaya)
```

```
result(); // Output: Asad Ahmad from Bihar, India
```

Things to remember ?

1. bind() ne function ko turant call nahi kiya, balki ek new function return kiya.
2. result() jab call kiya tab actual execution hua.

Bind ka simple formula:

```
let newFunction = functionName.bind(object, arg1, arg2, ...);  
newFunction();
```


Real-World Examples

Function Borrowing (Call & Apply)

Maan lo ek object me method hai, aur tum doosre object me bina dobara likhe use karna chahte ho.

```
const student = {  
  name: "Shaaz",  
  greet: function(subject) {  
    console.log(`Hello, I am ${this.name} and I love  
    ${subject}!`);  
  }  
};
```

```
const teacher = {  
  name: "Rahul Sir"  
};
```

// Student ka method teacher ke liye bhi use ho raha hai!

```
student.greet.call(teacher, "JavaScript"); // Output: Hello, I  
am Rahul Sir and I love JavaScript!
```


Using Bind for Event Listeners

Agar tum chahte ho ki ek object ka function event listener me bhi kaam kare, toh bind() useful hota hai.

```
const button = document.getElementById("myButton");
```

```
const obj = {  
  message: "Button clicked!",  
  showMessage: function () {  
    console.log(this.message);  
  }  
};
```

// Bind ka use karke ensure kiya ki `this` obj ko refer kare!

```
button.addEventListener("click",  
obj.showMessage.bind(obj));
```


Partial Functions using Bind

Maan lo tum ek function ko pre-filled arguments ke saath store karna chahte ho.

```
function multiply(a, b) {  
    return a * b;  
}
```

```
const double = multiply.bind(null, 2); // a = 2 fix ho gaya  
console.log(double(5)); // Output: 10  
console.log(double(10)); // Output: 20
```



Things to remember ?

- bind(null, 2) ne first argument 2 fix kar diya.
- Ab double() jab bhi call hoga, wo $2 \times$ given number return karega!

This in JavaScript

Simple Definition:

👉 JavaScript me this ek special keyword hai jo kisi function ya object ko refer karta hai.

👉 Kis object ko refer karega? Ye uske "execution context" pe depend karta hai!

💡 Ek simple rule yaad rakho:

"this ka value depend karta hai ki function kaise call kiya gaya hai!" ✅

🔥 Different Ways to Use this in JavaScript

this by itself (Global Scope)

Agar this global scope me likha hai, toh ye global object ko refer karega.

- Browser me this = window object.
- Node.js me this = empty object {}.

Example:

console.log(this); // Output: window (browser me)

Things to remember?

- JavaScript ka sabse bada object window hota hai, jisme console, setTimeout(), aur bahut saari cheezein hoti hain!

this inside an Object Method

Agar this kisi object ke method me likha ho, toh wo usi object ko refer karega.

Example:

```
const person = {  
  firstName: "Shaaz",  
  lastName: "Akhtar",  
  fullName: function () {  
    return this.firstName + " " + this.lastName;  
  }  
};
```

```
console.log(person.fullName()); // Output: "Shaaz Akhtar"
```

Things to remember ?

- this.firstName ka matlab hai ki wo person object ke andar firstName property ko refer kar raha hai!

this inside a Function (Alone)

Agar this ek normal function ke andar likha ho, toh global object (window) ko refer karega.

Example:

```
function showThis() {  
  console.log(this);  
}  
  
showThis(); // Output: window (browser me)
```

Things to remember ?

- Normal function ke andar this window ko point karta hai!

this inside a Function (Strict Mode)

Agar strict mode enable ho, toh this undefined ho jata hai.

👁👁 Example:

"use strict";

```
function showThisStrict() {  
  console.log(this);  
}
```

showThisStrict(); // Output: undefined

Things to remember ?

- "use strict" mode me global scope me this undefined ho jata hai!

this inside Arrow Functions (Lexical this)

👉 Arrow functions this ko bind nahi karte, wo apne parent scope se this le lete hain! ✅

Example:

```
const person = {  
  firstName: "Shaaz",  
  lastName: "Akhtar",  
  fullName: function () {  
    const arrowFunc = () => {  
      console.log(this.firstName + " " + this.lastName);  
    };  
    arrowFunc();  
  }  
};
```

person.fullName(); // Output: "Shaaz Akhtar"

Things to remember ?

- Arrow function me this apne parent function (fullName()) se lega!
- Arrow function ka this kabhi change nahi hota!


this in Event Listeners

Agar this event listener me use kiya jaye, toh ye us element ko refer karega jisme event laga hai.

Example:

```
document.getElementById("myButton").addEventListener(  
"click", function () {  
    console.log(this); // `this` button element ko refer karega  
});
```

Things to remember ?

- Event listener ke andar this wo element hoga jisme event laga hai! 

this with Call, Apply, and Bind

Agar tum manually this ka reference change karna chahte ho, toh call(), apply(), aur bind() ka use kar sakte ho! 🚀

Example:

```
const person1 = {  
  firstName: "Shaaz",  
  lastName: "Akhtar",  
};
```

```
const person2 = {  
  firstName: "Asad",  
  lastName: "Ahmad",  
};
```

```
function printName(city, country) {  
  console.log(this.firstName + " " + this.lastName + " from " + city +  
    ", " + country);  
}
```

// Call

```
printName.call(person2, "Bihar", "India"); // Output: "Asad Ahmad  
from Bihar, India"
```

// Apply (arguments array me pass hote hain)

```
printName.apply(person2, ["Bihar", "India"]);
```

// Bind (function return karta hai, turant call nahi hota)

```
const newFunc = printName.bind(person2, "Bihar", "India");  
newFunc(); // Output: "Asad Ahmad from Bihar, India"
```

Things to remember ?

- call() aur apply() turant function call kar dete hain.
- bind() ek naya function return karta hai, jo baad me call hota hai.

Object-Oriented Programming (OOP)

OOP Kya Hai?

- 👉 OOP ek programming style hai jo objects ka use karke code likhne ki technique hai.
- 👉 Objects ek tarah ke "containers" hain jisme data aur methods (functions) hoti hain.
- 👉 OOP ka goal hota hai code ko zyada organized, reusable aur maintainable banana. ✅

Example:

Maan lo tum ek car manufacturing company ho.

- Har car me kuch common properties hoti hain – color, brand, speed.
- Har car ke paas kuch actions hote hain – start(), stop(), accelerate().
- Har naye model ka ek base structure hota hai, usme naye features add kiye ja sakte hain.
- Yehi OOP ka concept hai!

JavaScript OOP Concepts

JavaScript me OOP ke 4 main pillars hote hain:

Encapsulation – Data ko ek object me bundle karna.

Inheritance – Ek class dusri class se properties aur methods inherit kar sakti hai.

Polymorphism – Ek method ka different objects pe alag behavior ho sakta hai.

Abstraction – Unnecessary details hide karna aur sirf necessary cheezein expose karna.

Chalo, har concept ko deep me samajhte hain!

Creating Objects (Object Literals)

Sabse simple tareeka objects banane ka {} notation hai.

Example:

```
const person = {  
  firstName: "Shaaz",  
  lastName: "Akhtar",  
  fullName: function () {  
    return this.firstName + " " + this.lastName;  
  }  
};  
  
console.log(person.fullName()); // Output: "Shaaz Akhtar"
```

Things to remember ?

- Object ke andar properties aur methods ho sakte hain.
- Methods ek function hote hain jo object ke andar hote hain.

Constructor Functions (Multiple Objects Banane Ka Tareeka)

Agar tumhe bahut saare similar objects banane hain, toh constructor function ka use karo! ✓

Example:

```
function Person(firstName, lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.fullName = function () {  
        return this.firstName + " " + this.lastName;  
    };  
}
```

```
let person1 = new Person("Shaaz", "Akhtar");
```

```
let person2 = new Person("Asad", "Ahmad");
```

```
console.log(person1.fullName()); // Output: "Shaaz Akhtar"
```

```
console.log(person2.fullName()); // Output: "Asad Ahmad"
```

Things to remember?

- new keyword ek naya object banata hai.
- Constructor functions ek tarah ke "**blueprint**" hote hain jisme se multiple objects ban sakte hain. ✓

Prototypes (Memory Efficient Objects)

Agar tum constructor function ke andar directly function likhoge, toh har object ek alag copy store karega, jo memory waste karega.

Iska solution hai prototype ka use!

Example:

```
function Person(firstName, lastName) {
```

```
  this.firstName = firstName;
```

```
  this.lastName = lastName;
```

```
}
```

```
// Prototype me method add karna
```

```
Person.prototype.fullName = function () {
```

```
  return this.firstName + " " + this.lastName;
```

```
};
```

```
let person1 = new Person("Shaaz", "Akhtar");
```

```
console.log(person1.fullName()); // Output: "Shaaz Akhtar"
```

Things to remember ?

- Prototype ka use karke sabhi objects ek hi method share karte hain!
- Yeh memory ko save karta hai aur execution fast hota hai.

Inheritance (Ek Class Dusre Se Properties Aur Methods Leta Hai)

👉 Maan lo ek "Animal" class hai, aur tum "Dog" class create karna chahte ho jo Animal ke features le sake!

Example:

```
function Animal(name) {  
    this.name = name;  
}
```

```
Animal.prototype.speak = function () {  
    console.log(`${this.name} makes a sound.`);  
};
```

// Dog ko Animal se inherit karna

```
function Dog(name, breed) {  
    Animal.call(this, name); // Animal constructor call karna  
    this.breed = breed;  
}
```

// Inheritance set karna

```
Dog.prototype = Object.create(Animal.prototype);
```

```
Dog.prototype.constructor = Dog;
```

```
Dog.prototype.bark = function () {  
    console.log(`${this.name} barks loudly!`);  
};
```

```
let dog1 = new Dog("Bruno", "Labrador");
```

```
dog1.speak(); // Output: "Bruno makes a sound."
```

```
dog1.bark(); // Output: "Bruno barks loudly!"
```

Things to remember ?

- Dog class ne Animal class ke properties aur methods inherit kiye!
- call() ka use karke parent class ka constructor call kiya.

Encapsulation (Data Hide Karna)

☞ Encapsulation ka matlab hai data ko private banana taki wo sirf controlled methods se access ho sake.

Example:

```
function BankAccount(owner, balance) {
```

```
    let _balance = balance; // Private variable
```

```
    this.owner = owner;
```

```
    this.getBalance = function () {
```

```
        return _balance;
```

```
    };
```

```
    this.deposit = function (amount) {
```

```
        if (amount > 0) _balance += amount;
```

```
    };
```

```
}
```

```
let account = new BankAccount("Shaaz", 5000);
```

```
console.log(account.getBalance()); // Output: 5000
```

```
account.deposit(2000);
```

```
console.log(account.getBalance()); // Output: 7000
```

```
console.log(account._balance); // ❌ Undefined (Direct  
access not allowed)
```

Things to remember?

- Private variables function ke andar define hote hain.
- Unko sirf getter aur setter methods ke through access kiya ja sakta hai.

Polymorphism (Ek Method Ka Alag-Alag Behavior)

☞ Same method ka alag-alag classes me different behavior ho sakta hai!

Example:

```
class Animal {  
    speak() {  
        console.log("Animal makes a sound");  
    }  
}
```

```
class Dog extends Animal {  
    speak() {  
        console.log("Dog barks");  
    }  
}
```

```
class Cat extends Animal {  
    speak() {  
        console.log("Cat meows");  
    }  
}
```

```
let myDog = new Dog();
```

```
let myCat = new Cat();
```

```
myDog.speak(); // Output: "Dog barks"
```

```
myCat.speak(); // Output: "Cat meows"
```

Things to remember?

- Different classes me same method ka different behavior ho sakta hai!

JavaScript Modules

JavaScript Modules Kya Hain?

👉 Modules ka matlab hai "Code ko alag-alag files me tod kar organize karna"

👉 Ek module ek file hoti hai jo variables, functions, ya classes ko export ya import karti hai.

👉 Modules ka use karne se code organized, reusable aur maintainable banta hai!

Real-Life Example:

Maan lo tum ek ghar bana rahe ho.

- Bedroom, Kitchen, Bathroom alag-alag hote hain (Modules).
- Har room ka apna alag kaam hota hai – Bedroom me sona, Kitchen me khana banana.
- Poora ghar tabhi complete hoga jab sab rooms connect honge (Application).
- Modules bhi isi tarah alag-alag files me hote hain, jo ek dusre se connect hote hain!

Why Use JavaScript Modules?

✓ **Organized Code** – Ek jagah sab kuch likhne se better hai alag-alag files me likhna.

✓ **Reusability** – Ek baar likho, har jagah use karo!

✓ **Maintainability** – Code ko samajhna aur update karna easy hota hai.

✓ **Performance** – Modules ko lazy load kar sakte ho, jo performance improve karta hai.

Creating & Using JavaScript Modules

Creating a Module File (math.js)

Maan lo tum ek math utilities ka module bana rahe ho.

Is file me kuch useful functions define karenge aur export karenge.

Example:

```
// math.js
```

```
export function add(a, b) {  
    return a + b;  
}
```

```
export function subtract(a, b) {  
    return a - b;  
}
```

```
export function multiply(a, b) {  
    return a * b;  
}
```

```
export function divide(a, b) {  
    if (b !== 0) return a / b;  
    else return "Cannot divide by zero!";  
}
```

Things to remember ?

- export keyword ka use karke functions ko available banaya.
- Ab ye functions kisi bhi dusre file me import ho sakte hain!

Creating Another Module (greeting.js)

Ek greeting module bana rahe hain jo user ko welcome karega.

Example:

```
// greeting.js
```

```
export const greetingMessage = "Hello! Welcome to  
JavaScript Modules.";
```

```
export function greetUser(name) {  
  return `Hello, ${name}! Welcome to our website.`;  
}
```

Things to remember ?

- export se variables aur functions ko accessible banaya.
- Ab hum inko kisi bhi file me use kar sakte hain!

Importing Modules (main.js)

Ab hum math.js aur greeting.js ke functions ko import karenge aur use karenge.

Example:

```
// main.js
```

```
import { add, subtract, multiply, divide } from './math.js';
```

```
import { greetingMessage, greetUser } from './greeting.js';
```

```
console.log(add(5, 3)); // Output: 8
```

```
console.log(subtract(10, 4)); // Output: 6
```

```
console.log(multiply(6, 7)); // Output: 42
```

```
console.log(divide(20, 5)); // Output: 4
```

```
console.log(greetingMessage); // Output: "Hello! Welcome to  
JavaScript Modules."
```

```
console.log(greetUser("Shaaz")); // Output: "Hello, Shaaz!  
Welcome to our website."
```

🤔 Things to remember ?

- import se hum exported functions aur variables use kar sakte hain.
- Curly braces {} ka use sirf named exports ke liye hota hai.

Default Exports – Ek File Ka Main Function Export Karna

Kabhi-kabhi tum ek module me sirf ek hi function ya class export karna chahte ho, uske liye default export ka use hota hai.

Creating a Default Export (utils.js)

```
// utils.js
```

```
export default function multiply(a, b) {  
    return a * b;  
}
```

Things to remember ?

- Yahan sirf ek function export ho raha hai, isliye export default use kiya.

Importing Default Export (main.js)

```
// main.js
```

```
import multiply from './utils.js';  
console.log(multiply(4, 5)); // Output: 20
```

Things to remember?

- Default import me {} lagane ki zaroorat nahi hoti!

Dynamic Imports – Jab Zaroorat Ho Tabhi Load Karo

Agar tum modules ko demand pe load karna chahte ho, toh dynamic imports ka use kar sakte ho.

Example (main.js)

```
// main.js
```

```
import('./math.js').then(module => {  
    console.log(module.add(2, 3)); // Output: 5  
});
```

Things to remember ?

- Ye approach page ki speed improve karta hai, kyunki module tabhi load hota hai jab zaroorat ho.

Modules in HTML (Browser)

Agar tum browser me modules use kar rahe ho, toh **<script>** me type="**module**" likhna zaroori hai.

Example:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
<title>JavaScript Modules</title>  
</head>  
<body>  
<script type="module" src="main.js"></script>  
</body>  
</html>
```

Things to remember ?

- type="module" likhna compulsory hai, tabhi import statements kaam karegi.

Error Handling

Error Handling Kya Hai?

- 👉 Error Handling ka matlab hai code me aane wale unexpected errors ko properly manage karna.
- 👉 Agar koi error aata hai, toh usko handle karna zaroori hai, nahi toh pura program crash ho sakta hai!
- 👉 Isse code zyada secure, debug-friendly aur user-friendly hota hai!

Real-Life Example:

Maan lo tum ek car drive kar rahe ho, aur achanak ek bada gadda road pe aa gaya!

- Agar brake nahi lagega, toh tum accident kar doge!
- Brake (Error Handling) laga ke safely gadi control kar sakte ho!
- Agar gadda na hota, tab bhi brake system (finally block) waisa hi kaam karega!

Why Use Error Handling?

- ✓ Prevent Crashes – Error aane par pura program band na ho.
- ✓ Debugging Easy – Errors ko track karna aur fix karna easy ho.
- ✓ Better User Experience – Users ko error messages diye ja sakein instead of sudden crashes.

Basic Error Handling in JavaScript

try...catch – Galti Pakdo Aur Handle Karo

try block me jo code error de sakta hai, usko likho.

Agar error aata hai, toh catch block usko handle karega aur program crash hone se bacha lega.

Example:

```
try {  
    let result = 5 / 0;  
    console.log("Result:", result); // Output: Infinity (No Error)  
    let name = undefined;  
    console.log(name.toUpperCase()); // ERROR! Cannot read  
property 'toUpperCase' of undefined  
} catch (error) {  
    console.log("Error Occurred:", error.message);  
}
```

Things to remember ?

- try me jo error aayega, wo catch block me handle ho jayega!
- Pura program crash nahi hoga, bas error ko gracefully handle kiya jayega.

finally – Har Haal Me Chalega!

Agar tumhe koi aisa code likhna hai jo chahe error aaye ya na aaye, hamesha chale, toh finally use karo.

Example:

```
try {  
    let x = 10;  
    console.log(x.toUpperCase()); // ERROR!  
} catch (error) {  
    console.log(" Error Caught:", error.message);  
} finally {  
    console.log(" Cleanup Complete. This will run no matter  
what!");  
}
```

Things to remember ?

- finally block hamesha chalega, chahe try me error aaye ya na aaye!
- Iska use cleanup operations ke liye hota hai (like closing files, disconnecting DB etc.).

throw – Apna Khud Ka Custom Error Banayein

Kabhi-kabhi tumhe apna khud ka custom error throw karna pad sakta hai.

throw ka use karke tum custom error messages generate kar sakte ho.

Example:

```
function checkAge(age) {  
    if (age < 18) {  
        throw new Error("You must be 18 or older to vote!"); //
```

Custom error throw kiya

```
    }  
    return "✅ You are eligible to vote!";  
}  
  
try {  
    console.log(checkAge(16)); // Error  
} catch (error) {  
    console.log("❌ Error:", error.message);  
}
```

Things to remember ?

- throw se hum apni marzi ka custom error generate kar sakte hain.
- Agar condition satisfy nahi hoti, toh error throw ho jata hai!

Advanced Error Handling in JavaScript

try...catch with async/await

Agar tum asynchronous code use kar rahe ho, toh error handling aur bhi important ho jata hai!

Example:

```
async function fetchData() {  
  try {  
    let response = await fetch("https://invalid-url.com"); //  
    Invalid URL  
    let data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.log("❌ Error fetching data:", error.message);  
  }  
}
```

fetchData();

Things to remember ?

- Agar fetch() me koi error aata hai, toh catch block usko handle karega.
- Agar try...catch nahi lagate, toh pura program crash ho jata!

Custom Error Classes (Advanced Level)

Agar tum apna custom error type banana chahte ho, toh Error class ko extend kar sakte ho!

Example:

```
class CustomError extends Error {  
    constructor(message) {  
        super(message);  
        this.name = "CustomError";  
    }  
}  
  
try {  
    throw new CustomError("This is a custom error!");  
} catch (error) {  
    console.log(`${error.name}: ${error.message}`);  
}
```

Things to remember ?

- Custom error classes bana sakte hain jo Error class ko extend karein!
- Custom error messages aur handling aur bhi powerful ban sakti hai.

Common Errors in JavaScript

✓ **ReferenceError** – Jab koi undefined variable access karne ki koshish hoti hai.

```
console.log(myVar); // ReferenceError: myVar is not defined
```

✓ **TypeError** – Jab kisi variable ka expected type match nahi hota.

```
let num = 10;
```

```
num.toUpperCase(); // TypeError: num.toUpperCase is not a  
function
```

✓ **SyntaxError** – Jab code me koi syntax mistake hoti hai.

```
try {  
    eval("alert('Hello'); // SyntaxError: missing closing bracket  
} catch (error) {  
    console.log(error.message);  
}
```

✓ **RangeError** – Jab value allowed range se bahar ho.

```
let arr = new Array(-1); // RangeError: Invalid array length
```


setTimeout & setInterval

What are setTimeout and setInterval?

👉 **setTimeout** – Ek function ko kuch der baad ek baar chalata hai.

👉 **setInterval** – Ek function ko baar-baar ek fixed interval pe chalata hai.

Real-Life Example:

Maan lo tum pizza order kar rahe ho. 🍕

- Tumne call kiya aur bola "20 min baad pizza deliver karna" – Ye setTimeout hai.
- Tumne subscription liya jo har 1 mahine baad automatically renew hoga – Ye setInterval hai.

Samajh gaye? Chalo code ke saath explore karte hain!

setTimeout – Function Ko Delay Se Chalana

👉 setTimeout(function, delay) ka use function ko ek specific time ke baad run karne ke liye hota hai.

Example:

```
console.log("🚀 Start...");
```

```
setTimeout(() => {  
    console.log("⌚ This message will appear after 3  
seconds!");  
}, 3000);
```

```
console.log("✅ Code running...");
```

Things to remember ?

- Pehle "Start..." print hoga, fir "Code running..." print hoga, aur 3 second baad "This message will appear after 3 seconds!" aayega!
- JavaScript asynchronous hai, toh setTimeout background me chalta hai!

clearTimeout – setTimeout Ko Cancel Karna

Agar tum setTimeout ko chalne se pehle hi cancel karna chahte ho, toh clearTimeout() ka use hota hai.

Example:

```
console.log("🚀 Start...");
```

```
let timeoutId = setTimeout(() => {  
    console.log("⌚ This message won't appear!");  
}, 5000);
```

```
clearTimeout(timeoutId); // Timeout cancel ho gaya!  
console.log("🛑 Timeout cleared!");
```

Things to remember ?

- setTimeout ek ID return karta hai, jisko clearTimeout() se cancel kar sakte hain.
- Yahan 5 second baad message print hona tha, but clearTimeout(timeoutId) se cancel ho gaya!

setInterval – Function Ko Baar-Baar Chalana

👉 setInterval(function, interval) ek function ko repeatedly ek fixed interval ke baad chalata hai.

Example:

```
console.log("🚀 Countdown Started...");
```

```
let count = 1;
```

```
let intervalId = setInterval(() => {
```

```
    console.log(`⌚ Countdown: ${count}`);
```

```
    count++;
```

```
    if (count > 5) {
```

```
        clearInterval(intervalId); // 🛑 Stop after 5 times
```

```
        console.log("✅ Countdown Stopped!");
```

```
    }
```

```
}, 1000);
```

Things to remember ?

- Yeh function har 1 second baad chalega aur count print karega.
- 5 baar print hone ke baad clearInterval() se band ho jayega!

clearInterval – setInterval Ko Rukna

Agar tum kisi interval ko manually stop karna chahte ho, toh clearInterval() ka use hota hai!

Example:

```
let intervalId = setInterval(() => {
```

```
    console.log("🔄 This message will keep appearing!");
```

```
}, 2000);
```

```
setTimeout(() => {
```

```
    clearInterval(intervalId); // ❌ Stop interval after 7 seconds
```

```
    console.log("🛑 Interval cleared!");
```

```
}, 7000);
```

Things to remember ?

- Message har 2 second baad print hoga.
- 7 second ke baad clearInterval(intervalId) se interval stop ho jayega!

Spread & Rest Operators

What are Spread (...) & Rest (...) Operators?

☞ Spread Operator (...) – Ek array ya object ko tod ke individual elements me expand karta hai.

☞ Rest Operator (...) – Multiple values ko ek array me collect karta hai.

Real-Life Example:

Maan lo tum pizza bana rahe ho. 🍕

- Tumhare paas ek ready-made base (spread operator) hai jisme tum cheese, toppings aur sauce mila sakte ho!
- Tumne bahut saare ingredients liye aur ek plate me rakh diye (rest operator).

Spread Operator (...)

☞ Spread Operator ka use arrays aur objects ko expand karne ke liye hota hai.

☞ Ye kisi array ya object ke saare elements ko individual elements me tod deta hai.

Combining Arrays (Array Merge)

Maan lo tumhare paas do arrays hain aur unko merge karna hai.

Example:

```
const nums1 = [1, 2, 3, 4, 5];
```

```
const nums2 = [6, 7, 8, 9];
```

```
const joinedArray = [...nums1, ...nums2];
```

```
console.log(joinedArray);
```

```
// Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Things to remember ?

- Spread Operator ne dono arrays ko merge kar diya bina kisi loop ke!

Copying an Array (Immutable Copy)

Agar tum array ki ek copy bana ke modify karna chahte ho bina original array ko change kiye, toh spread ka use karo.

Example:

```
const originalArray = [10, 20, 30];
```

```
const copiedArray = [...originalArray];
```

```
copiedArray.push(40);
```

```
console.log(originalArray); // Output: [10, 20, 30] (Original  
remains same)
```

```
console.log(copiedArray); // Output: [10, 20, 30, 40] (New  
modified copy)
```

Things to remember ?

- Agar = operator se copy karte, toh dono arrays linked hote.
- Spread operator se ek alag naya array bana jo independent hai.

Adding New Properties to Objects

Agar tum ek object me naye properties add karna chahte ho bina original object ko modify kiye, toh spread ka use karo.

Example:

```
const user = { name: "Shaaz", age: 23 };
```

```
const updatedUser = { ...user, city: "Mumbai" };
```

```
console.log(updatedUser);
```

```
// Output: { name: "Shaaz", age: 23, city: "Mumbai" }
```

Things to remember ?

- Original object ko modify kiye bina naye properties add kar sakte hain.

Overwriting Properties in Objects

Agar tum kisi object ki existing properties ko overwrite karna chahte ho, toh spread ka use karo.

Example:

```
const user = { name: "Shaaz", age: 23 };
```

```
const updatedUser = { ...user, age: 25 }; // Age update ho gayi
```

```
console.log(updatedUser);
```

```
// Output: { name: "Shaaz", age: 25 }
```

Things to remember ?

- Spread operator se object ki properties ko easily update kar sakte hain.

Rest Operator (...)

☞ Rest Operator ka use multiple values ko ek single array me collect karne ke liye hota hai.

☞ Function parameters ke saath use hota hai jab tumhe unknown number of arguments handle karne ho.

Function Parameters with Rest Operator

Agar tum function me unlimited parameters pass karna chahte ho, toh rest operator use karo!

Example:

```
function addNumbers(...nums) {  
    return nums.reduce((sum, num) => sum + num, 0);  
}
```

```
console.log(addNumbers(1, 2, 3, 4, 5));
```

// Output: 15

Things to remember ?

- ...nums function ke saare arguments ko ek array me collect kar raha hai.
- reduce() ka use karke unko sum kiya gaya.

Separating First Element from Rest

Agar tumhe array ke first element ko alag karna ho aur baaki elements ek alag array me store karna ho, toh rest operator use karo!

Example:

```
const [first, ...rest] = [100, 200, 300, 400];
```

```
console.log(first); // Output: 100
```

```
console.log(rest); // Output: [200, 300, 400]
```

Things to remember ?

- first variable me array ka first element store ho gaya.
- rest me baaki ke elements store ho gaye.

Real-World Example: Combining Both

Maan lo tum ek food delivery app bana rahe ho jisme pizza ka order process kar rahe ho.

Example:

```
function orderPizza(base, ...toppings) {  
  console.log(`🍕 Your ${base} pizza with ${toppings.join(",  
  ")} is ready!`);  
}
```

```
orderPizza("Thin Crust", "Cheese", "Olives", "Mushrooms",  
"Chicken");
```

// Output: 🍕 Your Thin Crust pizza with Cheese, Olives, Mushrooms, Chicken is ready!

Things to remember ?

- Base (First argument) alag store ho gaya.
- Baaki toppings ...toppings array me store ho gayi.

API (Data Fetching)

API (Data Fetching) Kya Hota Hai?

- ☞ API (Application Programming Interface) ka matlab hai do applications ke beech communication.
- ☞ Server se data lane ke liye hum API calls karte hain.
- ☞ Data ko hum JSON format me receive karte hain.

Real-Life Example:

Maan lo tum Swiggy/Zomato app use kar rahe ho

- Jab tum restaurant search karte ho, toh app server ko ek request bhejti hai.
- Server wapas JSON format me restaurant ka data bhejta hai.
- App is data ko UI par dikhata hai!

Yehi API Fetching hota hai!

Data Fetching Ke Popular Methods

Fetch API (Modern & Built-in JavaScript method)

Axios (Popular Library, Cleaner Syntax)

AJAX (Old method, still used in some places)

Chalo, in teeno ka deep dive karte hain!

Fetch API (Modern JavaScript Way)

`fetch()` ek built-in JavaScript function hai jo server se data fetch karne ke liye use hota hai.

Example: Fetching Data from API

```
fetch("https://jsonplaceholder.typicode.com/posts/1")  
  .then(response => response.json()) // Convert response to  
JSON  
  .then(data => console.log(data)) // Print data  
  .catch(error => console.log(" Error:", error));
```

Things to remember ?

- `fetch()` ek request bhejta hai.
- Pehla `.then(response => response.json())` JSON format me convert karta hai.
- Dusra `.then(data => console.log(data))` actual data dikhata hai.
- Agar koi error aata hai, toh `.catch(error => console.log(error))` usko handle karta hai.

Fetch API with async/await (Better Syntax)

☞ async/await ka use karke code aur clean aur readable ban jata hai!

Example:

```
async function fetchData() {  
  try {  
    let response = await  
fetch("https://jsonplaceholder.typicode.com/posts/1");  
    let data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.log(" Error:", error);  
  }  
}  
  
fetchData();
```

Things to remember ?

- async function me await ka use karke hum promise ko properly handle kar rahe hain!
- Try-catch block ka use karke error handling bhi ho rahi hai!

Axios (Popular & Cleaner Way)

- 👉 Axios ek third-party library hai jo Fetch API se zyada easy aur powerful hai!
- 👉 Ye automatically response ko JSON me convert kar deta hai!

Installation (Browser & Node.js)

Browser me directly use karne ke liye CDN add karo:

```
<script  
src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js">  
</script>
```

Node.js me install karna ho toh:

npm install axios

🔥 Axios GET Request

- 👉 Axios me .then() aur async/await dono kaam karte hain!

Example:

```
axios.get("https://jsonplaceholder.typicode.com/posts/1")  
  .then(response => console.log(response.data))  
  .catch(error => console.log(" Error:", error));
```

Things to remember ?

- Axios me response automatic JSON me convert hota hai, response.json() ki zaroorat nahi!

Axios GET with async/await

👉 Async/await se code aur clean aur readable ban jata hai.

Example:

```
async function fetchData() {  
  try {  
    let response = await  
    axios.get("https://jsonplaceholder.typicode.com/posts/1");  
    console.log(response.data);  
  } catch (error) {  
    console.log(" Error:", error);  
  }  
}
```

fetchData();

Things to remember ?

- Axios me async/await ka use karke error handling aur readable ho gayi!

Axios POST Request (Server Par Data Bhejna)

👉 Agar tumhe server par data send karna ho, toh axios.post() ka use karo.

Example:

```
async function sendData() {  
  try {  
    let response = await  
    axios.post("https://jsonplaceholder.typicode.com/posts", {  
      title: "New Post",  
      body: "This is a new post",  
      userId: 1  
    });  
    console.log("Data Sent:", response.data);  
  } catch (error) {  
    console.log("Error:", error);  
  }  
}
```

sendData();

Things to remember ?

- axios.post() ka use karke hum server par data send kar sakte hain.

AJAX (Old Way, Still Used)

☞ AJAX (Asynchronous JavaScript and XML) ek purana tareeka hai data fetch karne ka.

☞ Aaj bhi kuch old applications me iska use hota hai!

Fetching Data Using XMLHttpRequest (Old Method)

```
let xhr = new XMLHttpRequest();

xhr.open("GET",
"https://jsonplaceholder.typicode.com/posts/1", true);
xhr.onreadystatechange = function () {
    if (xhr.readyState === 4 && xhr.status === 200) {
        console.log(JSON.parse(xhr.responseText));
    }
};

xhr.send();
```

Things to remember ?

- Ye purana tareeka hai jo ab modern methods (fetch(), axios) se replace ho gaya hai!

Hoisting

What is Hoisting?

- 👉 Hoisting ek process hai jisme JavaScript execution ke dauraan variables aur functions ko scope ke top pe move kar deta hai.
- 👉 Matlab, declare ki gayi cheezein use hone se pehle hi JavaScript ke dimaag me pahuch jaati hain. 🤖
- 👉 Lekin dikkat yeh hai ki variables ki sirf declaration hoist hoti hai, unki values nahi!

Real-Life Example:

Maan lo tum exam dene gaye ho, aur tum teacher se pehle hi answer sheet maang rahe ho!

- Agar tumne var use kiya hai, toh teacher boleگا "Haan, sheet toh le lo, but abhi khaali hai (undefined)".
- Agar tumne let ya const use kiya, toh teacher boleگا "Abhi sheet mili hi nahi, error aayega!"
- Agar tum function call kar rahe ho jo function declaration hai, toh teacher boleگا "Haan bhai, likho!"

Hoisting with var

☞ var ka declaration hoist hota hai, lekin assignment nahi!

Example:

```
console.log(a); // Output: undefined
```

```
var a = 5;
```

```
console.log(a); // Output: 5
```

JavaScript Isko Kaise Dekhta Hai?

```
var a; // Hoisting: Declaration upar chali gayi
```

```
console.log(a); // undefined
```

```
a = 5; // Assignment yahan ho rahi hai
```

```
console.log(a); // 5
```

Things to remember ?

- var ka declaration hoist hota hai, but value nahi assign hoti.
- Isliye pehla console.log(a); undefined print karega!

Hoisting with let & const

👉 let aur const bhi hoist hote hain, but unhe "**Temporal Dead Zone (TDZ)**" me rakha jata hai.

👉 TDZ ka matlab hai ki jab tak variable declare nahi ho jata, tab tak usko access nahi kar sakte.

Example with let:

```
console.log(a); // ❌ ReferenceError: Cannot access 'a' before  
initialization
```

```
let a = 5;
```

JavaScript Isko Kaise Dekhta Hai?

// Hoisting ho rahi hai, but Temporal Dead Zone (TDZ) me hai!

```
let a;
```

```
console.log(a); // ❌ ReferenceError
```

```
a = 5;
```

Things to remember ?

- let aur const ka declaration hoist hota hai, but TDZ me hota hai, isliye access nahi kar sakte!

Hoisting with Function Declarations

👉 Function declarations hoist hoti hain, matlab function ko define karne se pehle bhi call kar sakte ho!

Example:

helloWorld(); // Works fine

```
function helloWorld() {  
    console.log("Hello World");  
}
```

helloWorld(); // ✅ Works fine

JavaScript Isko Kaise Dekhta Hai?

```
function helloWorld() { // Function pura ka pura hoist ho gaya!  
    console.log("Hello World");  
}
```

helloWorld(); // ✅ Works

Things to remember ?

- Function declaration hoist hoti hai, isliye isko pehle call karna allowed hai.

Hoisting with Function Expressions (var)

👉 Function expressions var ke saath hoist hoti hain, lekin function ka value hoist nahi hota!

Example:

```
helloWorld(); // ❌ TypeError: helloWorld is not a function
var
helloWorld = function() {
    console.log("Hello World");
};
```

JavaScript Isko Kaise Dekhta Hai?

var helloWorld; // Hoisting: Bas variable declare hua, function nahi

helloWorld(); // ❌ TypeError: helloWorld is not a function

```
helloWorld = function() {
    console.log("Hello World");
};
```

Things to remember ?

- Function expression me function ka value hoist nahi hota, sirf var helloWorld; hoist hota hai.

Hoisting with Function Expressions (let & const)

👉 Function expressions jo let ya const ke saath hain, wo bhi hoist hote hain, lekin TDZ me rehte hain!

Example:

helloWorld(); // ❌ ReferenceError: Cannot access 'helloWorld' before initialization

```
let helloWorld = () => {  
  console.log("Hello World");  
};
```

JavaScript Isko Kaise Dekhta Hai?

let helloWorld; // Hoisting ho rahi hai, but TDZ me hai

helloWorld(); // ❌ ReferenceError

```
helloWorld = () => {  
  console.log("Hello World");  
};
```

Things to remember ?

- let aur const ke saath function expressions hoist to hote hain, but access nahi kiya ja sakta jab tak assign na ho.

Array Destructuring (Smart Way)

```
let colors = ["Red", "Green", "Blue"];
```

```
let [firstColor, secondColor, thirdColor] = colors;
```

```
console.log(firstColor, secondColor, thirdColor);
```

// Output: Red Green Blue

- Ek hi line me array se values extract ho gayi!

Object Destructuring

👉 Agar tumhe ek object ke properties extract karni hain, toh destructuring ka use karo.

Example: Normal Object Extraction

```
let person = {
```

```
    name: "Shaaz",
```

```
    age: 23,
```

```
    city: "Mumbai"
```

```
};
```

```
let name = person.name;
```

```
let age = person.age;
```

```
let city = person.city;
```

```
console.log(name, age, city);
```

// Output: Shaaz 23 Mumbai

- Har property ke liye alag-alag likhna pad raha hai! ❌

Array Destructuring (Smart Way)

```
let colors = ["Red", "Green", "Blue"];
```

```
let [firstColor, secondColor, thirdColor] = colors;
```

```
console.log(firstColor, secondColor, thirdColor);
```

// Output: Red Green Blue

- Ek hi line me array se values extract ho gayi!

Object Destructuring

👉 Agar tumhe ek object ke properties extract karni hain, toh destructuring ka use karo.

Example: Normal Object Extraction

```
let person = {
```

```
    name: "Shaaz",
```

```
    age: 23,
```

```
    city: "Mumbai"
```

```
};
```

```
let name = person.name;
```

```
let age = person.age;
```

```
let city = person.city;
```

```
console.log(name, age, city);
```

// Output: Shaaz 23 Mumbai

- Har property ke liye alag-alag likhna pad raha hai! ❌

Object Destructuring (Smart Way)

```
let person = {  
  name: "Shaaz",  
  age: 23,  
  city: "Mumbai"  
};
```

```
let { name, age, city } = person;
```

```
console.log(name, age, city);
```

```
// Output: Shaaz 23 Mumbai
```

Ek hi line me object se values extract ho gayi!

Nested Destructuring

☞ Agar object ya array ke andar aur bhi objects ya arrays hain, toh nested destructuring ka use hota hai.

Example:

```
let user = {  
  id: 1,  
  personalInfo: {  
    firstName: "Shaaz",  
    lastName: "Akhtar",  
    address: {  
      city: "Mumbai",  
      country: "India"  
    }  
  }  
};
```

```
let { personalInfo: { firstName, lastName, address: { city,  
country } } } = user;
```

```
console.log(firstName, lastName, city, country);
```

// Output: Shaaz Akhtar Mumbai India

- Nested destructuring ka use karke deep properties bhi extract ho gayi!

Destructuring with Default Values

👉 Agar destructuring ke waqt value undefined ho, toh default value set kar sakte ho.

Example:

```
let person = { name: "Shaaz" };  
  
let { name, age = 25 } = person;  
  
console.log(name, age);
```

// Output: Shaaz 25 (Default value set ho gayi)

Agar age nahi mila, toh default 25 set ho gaya!

Destructuring Function Parameters

👉 Agar function ko object pass ho raha hai, toh destructuring se direct properties extract kar sakte ho. ✅

Example:

```
function displayUser({ name, age, city = "Unknown" }) {  
    console.log(`${name} is ${age} years old from ${city}.`);  
}  
  
let user = { name: "Shaaz", age: 23 };  
  
displayUser(user);
```

// Output: Shaaz is 23 years old from Unknown.

- Object ke properties function ke andar destructure ho rahi hain!

Rest Operator in Destructuring

☞ Agar tumhe sirf kuch values chahiye aur baaki values ek alag array/object me store karni ho, toh rest operator (...) ka use kar sakte ho. ✓

Example: Array Destructuring with Rest

```
let numbers = [10, 20, 30, 40, 50];
```

```
let [first, second, ...rest] = numbers;
```

```
console.log(first, second); // Output: 10 20
```

```
console.log(rest); // Output: [30, 40, 50]
```

- Pehle do values alag nikal li, baaki sab rest me store ho gayi!

Example: Object Destructuring with Rest

```
let person = { name: "Shaaz", age: 23, city: "Mumbai",  
country: "India" };
```

```
let { name, age, ...rest } = person;
```

```
console.log(name, age); // Output: Shaaz 23console.log(rest);
```

```
// Output: { city: "Mumbai", country: "India" }
```

- Pehli do properties extract ho gayi, baaki sab rest object me store ho gayi!

Deep Copy vs. Shallow Copy

What is Shallow Copy vs Deep Copy?

☞ Shallow Copy – Bas surface level pe copy hoti hai, original aur copied data linked hote hain.

☞ Deep Copy – Ek bilkul naya object ya array create hota hai, original se koi link nahi hota.

💡 Real-Life Example:

- Shallow Copy: Socho tum WhatsApp pe ek forwarded message bhejte ho. Agar koi usko edit kare, toh original message bhi change ho jata hai!
- Deep Copy: Socho tumne ek message likha aur uska screenshot bhej diya. Ab screenshot change karne ka koi chance nahi!

Shallow Copy (Beware! Linked Data)

👉 Shallow copy sirf reference ko copy karta hai, values ko nahi!

Example: Shallow Copy of Objects

```
let fruit1 = { name: "Apple", color: "Red" };
```

```
let fruit2 = fruit1; // ❌ Shallow Copy (Reference Copy Ho Raha Hai)
```

```
fruit2.color = "Green"; // Changing fruit2
```

```
console.log(fruit1); // Output: { name: "Apple", color: "Green" }
```

❌ (Original bhi change ho gaya!)

```
console.log(fruit2); // Output: { name: "Apple", color: "Green" }
```

Problem Kya Hai?

- fruit1 aur fruit2 same memory ko point kar rahe hain.
- Agar fruit2 me kuch change karo, toh fruit1 bhi change ho jata hai!

Shallow Copy Techniques (Beware of Reference)

Agar tum Object.assign() ya Spread Operator (...) ka use kar rahe ho, toh yeh sirf shallow copy bana raha hai.

Example: Shallow Copy Using Object.assign()

```
let person1 = { name: "Shaaz", age: 23 };
```

```
let person2 = Object.assign({}, person1); // ❌ Shallow Copy
```

```
person2.age = 30;
```

```
console.log(person1); // { name: "Shaaz", age: 23 } ✅ (Original safe hai!)
```

```
console.log(person2); // { name: "Shaaz", age: 30 }
```

✅ Yahan koi problem nahi thi, kyunki object me nested (deep) values nahi thi!

Deep Copy (Safe Copy, No Links)

👉 Deep copy ek bilkul independent naya object ya array create karta hai.

👉 Agar tum deep copy banao, toh original safe rahta hai.

🔥 Deep Copy Using `JSON.parse(JSON.stringify())` (Easy Trick)

```
let user1 = {  
  name: "Rahul",  
  address: {  
    city: "Delhi",  
    country: "India"  
  }  
};
```

```
let user2 = JSON.parse(JSON.stringify(user1)); // Deep Copy  
user2.address.city = "Mumbai"; // Changing city in user2  
console.log(user1.address.city); // Output: Delhi (Original safe  
hai!)  
console.log(user2.address.city); // Output: Mumbai
```

Things to remember ?

- `JSON.stringify()` object ko string me convert karta hai.
- `JSON.parse()` us string ko dobara object me convert karta hai.
- Isse reference toot jata hai, aur puri tarah se naye object ban jata hai! ✅

🚩 Downside:

- Functions ya undefined values remove ho jati hain.

Shallow vs Deep Copy for Arrays

👉 Shallow Copy se arrays linked hote hain, Deep Copy me naye arrays bante hain.

👁️ Shallow Copy Example (**Using slice() & concat()**)

```
let arr1 = [1, 2, 3, [4, 5]];
```

```
let arr2 = arr1.slice(); // Shallow Copy
```

```
arr2[3][0] = 100;
```

```
console.log(arr1); // Output: [1, 2, 3, [100, 5]] ❌ (Original bhi change ho gaya!)
```

```
console.log(arr2); // Output: [1, 2, 3, [100, 5]]
```

- Nested arrays linked rahe, toh original bhi change ho gaya!

Deep Copy for Arrays

👉 Agar tum JSON.parse(JSON.stringify()) ya Lodash use karoge, toh deep copy milega.

Example:

```
let arr1 = [1, 2, 3, [4, 5]];
```

```
let arr2 = JSON.parse(JSON.stringify(arr1)); // Deep Copy
```

```
arr2[3][0] = 100;
```

```
console.log(arr1); // Output: [1, 2, 3, [4, 5]] (Original safe hai!)
```

```
console.log(arr2); // Output: [1, 2, 3, [100, 5]]
```

- Ab original array change nahi hua, kyunki deep copy hui thi!

Event Loop & Microtasks

What is the Event Loop?

☞ JavaScript ka event loop ek "traffic controller" ki tarah kaam karta hai.

☞ Ye ensure karta hai ki saare tasks ek sequence me execute ho.

☞ Main concepts:

- Call Stack – Functions ko execute karta hai.
- Web APIs – Asynchronous tasks handle karta hai (setTimeout, Promises, etc.)
- Callback Queue – Background se aane wale tasks ko stack me bhejta hai.
- Microtask Queue – Promises aur other microtasks ko priority pe execute karta hai.

Real-Life Example:

Socho tum Domino's me pizza order kar rahe ho.

- **Call Stack** – Tumhara order liya ja raha hai.
- **Web APIs** – Order kitchen me chala gaya (background processing).
- **Callback Queue** – Tumhari order ready hoke queue me aa gayi.
- **Microtask Queue** – VIP customers (Promises, queueMicrotask()) ko fast serve kiya ja raha hai!

JavaScript Execution Flow (Event Loop in Action)

JavaScript ka kaam hota hai synchronous tasks complete karna aur asynchronous tasks ko background me process karna.

Example: Synchronous Code Execution

```
console.log("Start");  
console.log("Processing...");  
console.log("End");
```

✓ Output:

Start

Processing...

End

- Jo bhi synchronous hai wo direct execute hota hai!

Asynchronous Tasks & Event Loop

👉 Asynchronous functions Web APIs me chali jati hain, aur later execute hoti hain.

Example:

```
console.log("Start");  
setTimeout(() => {  
    console.log("Inside setTimeout");  
}, 0);  
console.log("End");
```

✓ Output:

Start

End

Inside setTimeout

- setTimeout Web API me chala gaya, aur baad me execute hua.
- Event Loop tab tak stack clear hone ka wait karega, fir callback queue se setTimeout execute karega.

Call Stack, Web APIs, Callback Queue & Microtask Queue (Deep Dive)

👉 Yeh sab milke event loop ko smooth banate hain!

Call Stack

- JavaScript ka main execution area hai.
- Jo bhi function execute hota hai wo Call Stack pe aata hai.

Web APIs

- Browser ke paas setTimeout, DOM Events, Fetch API jese async tasks handle karne ka alag system hota hai.
- Ye tasks Web APIs me process hote hain, na ki Call Stack me.

Callback Queue

- Web APIs me complete hone wale tasks callback queue me jate hain.
- Event loop check karta hai ki Call Stack empty hai ya nahi, fir Callback Queue execute hoti hai.

Microtask Queue

- Promises aur queueMicrotask() yaha store hote hain.
- Microtasks, Callback Queue se pehle execute hote hain.

Microtasks vs Macrotasks (Super Important)

👉 Microtasks zyada priority pe hote hain.

👉 Macrotasks (setTimeout, setInterval) baad me execute hote hain.

Example:

```
console.log("Start");
```

```
setTimeout(() => console.log("Inside setTimeout"), 0);
```

```
Promise.resolve().then(() => console.log("Inside Promise"));
```

```
console.log("End");
```

✅ Output:

Start

End

Inside Promise

Inside setTimeout

Things to remember ?

- Promise ka microtask queue zyada priority pe hai, isliye wo setTimeout se pehle execute hoga!

Real-World Example: Event Loop Execution

👉 Chalo ek aur example lete hain jo pura event loop explain karega.

Example:

```
console.log("1");
```

```
setTimeout(() => console.log("2"), 0);
```

```
Promise.resolve().then(() => console.log("3"));
```

```
console.log("4");
```

✅ Output:

1

4

3

2

Things to remember ?

1. Synchronous Code (`console.log("1")` & `console.log("4")`)
pehle execute hota hai.
2. Promise Microtask Queue me jata hai (priority high).
3. `setTimeout` Macrotask Queue me jata hai (priority low).
4. Promise execute hota hai (Microtask first).
5. Baad me `setTimeout` execute hota hai.

Debouncing & Throttling

What are Debouncing & Throttling?

👉 **Debouncing** – Multiple calls hone se rokta hai aur last call ko execute karta hai.

👉 **Throttling** – Calls ka gap fix karta hai, taaki frequent execution slow ho sake.

💡 Real-Life Example:

- **Debouncing** – Socho tumhara boss tumhe baar-baar kaam ke liye yaad dila raha hai, but tum sirf tab reply karte ho jab last baar yaad dilaaye!
- **Throttling** – Socho tum Instagram pe scroll kar rahe ho aur API sirf har 2 second me ek baar call ho rahi hai, chahe kitna bhi fast scroll karo!

Debouncing in JavaScript

👉 Debouncing ek technique hai jo tabhi function ko execute karti hai jab last call ke baad ek certain time tak koi aur call nahi aayi.

👉 Main use case: Search Bar Suggestions, Window Resize Events, Button Clicks Optimization, etc.

Example: Search Bar Without Debouncing (Problem)

```
function searchAPI(query) {  
    console.log(`Searching for: ${query}`);  
}  
  
document.getElementById("search").addEventListener("input",  
(e) => {  
    searchAPI(e.target.value); // Har keypress pe API call ho rahi hai  
});
```


Debouncing Solution (Fixing the Issue)

👉 Debouncing se sirf last keystroke ke baad API call hogi, agar user kuch second tak kuch na type kare.

Example:

```
function debounce(func, delay) {  
  let timer;  
  return function (...args) {  
    clearTimeout(timer); // Pehle ka timer clear kar do  
    timer = setTimeout(() => {  
      func(...args); // Naya timer set karo  
    }, delay);  
  };  
}  
  
// API Call Function  
function searchAPI(query) {  
  console.log(`Searching for: ${query}`);  
}  
  
// Debounced Function  
const debouncedSearch = debounce(searchAPI, 500);  
  
// Event Listener  
document.getElementById("search").addEventListener("input", (e) => {  
  debouncedSearch(e.target.value);  
});
```

Things to remember ?

- Jab tak user type karta rahega, API call nahi hogi!
- Jab user ruk jayega, tabhi last value ke liye API call hogi!
- Performance improve ho gayi, unnecessary API calls band ho gayi!

Throttling in JavaScript

☞ Throttling ek technique hai jo function ko fix interval me execute karti hai, chahe kitni bhi calls aaye.

☞ Main use case: Scroll Events, Window Resize, Button Spam Prevention, etc.

Example: Scroll Event Without Throttling (Problem)

```
window.addEventListener("scroll", () => {  
    console.log("User is scrolling...");  
});
```

Problem:

- Jab bhi user scroll karega, har millisecond me event fire ho raha hai!
- Agar heavy calculations ya animations chal rahi ho toh page slow ho sakta hai!

Throttling Solution (Fixing the Issue)

☞ Throttling se sirf har fixed interval me function execute hoga, chahe kitni bhi calls aaye.

Example:

```
function throttle(func, interval) {  
    let lastExecuted = 0;  
    return function (...args) {  
        let now = Date.now();  
        if (now - lastExecuted >= interval) {  
            lastExecuted = now;  
            func(...args);  
        }  
    };  
}
```

// Scroll Function

```
function onScroll() {  
    console.log("User is scrolling...");  
}
```

// Throttled Function

```
const throttledScroll = throttle(onScroll, 1000);
```

// Event Listener

```
window.addEventListener("scroll", throttledScroll);
```

Things to remember ?

- Ab har second me sirf ek baar function chalega, chahe kitna bhi scroll ho!
- Performance zyada smooth ho gayi, unnecessary function calls block ho gayi!

Introduction to React.js

☞ React ek JavaScript library hai jo Facebook ne develop ki thi, taaki fast aur interactive user interfaces ban sakein.

☞ React ka main use hota hai Single Page Applications (SPA) banane me!

☞ React me hum components ka use karke UI ko chhoti-chhoti independent files me tod sakte hain!

Real-Life Example:

Socho tum LEGO ke blocks se ek ghar bana rahe ho

- Har block ek component hai!
- Tum blocks ko reusable bana sakte ho!
- Agar ek block ka design change karna ho, toh poore ghar ko rebuild karne ki zaroorat nahi!

Installing & Setting Up React.js

👉 React install karne ke liye hum Vite ka use karenge, jo React ko fast aur optimized rakhta hai!

📌 Step 1: React App Create Karo

```
npm create vite@latest my-react-app --template react
```

Yeh ek naye React project ka setup kar dega!

📌 Step 2: Project Folder Me Enter Karo

```
cd my-react-app
```

📌 Step 3: Dependencies Install Karo

```
npm install
```

📌 Step 4: Development Server Start Karo

```
npm run dev
```

Ab tumhara React app chalne ke liye ready hai!

What is a Single Page Application (SPA)?

- ☞ SPA ek aisi website hoti hai jo bina full-page reload kare naye content ko load kar sakti hai!
- ☞ Sirf body ka content reload hota hai, pura page nahi!

Real-Life Example:

Socho tum YouTube chal rahe ho

- Tum "Movies" tab pe ho, phir "Songs" tab pe click karte ho
- Page reload nahi hota, sirf body ka content change hota hai!

React isliye popular hai kyunki ye fast SPAs banata hai!

What is a Component in React?

- ☞ Component ek independent, reusable code ka tukda hota hai jo UI ka ek part represent karta hai.
- ☞ React me har UI element ek component ho sakta hai (button, navbar, footer, form, etc.)

Real-Life Example:

Socho tum Netflix ka UI design kar rahe ho!

- Ek component ho sakta hai "Movie Card"
- Ek component ho sakta hai "Navbar"
- Ek component ho sakta hai "Search Bar"

Example: Creating a Simple Component

Step 1: RedButton.jsx Component Banao

```
import React from 'react';
```

```
function RedButton() {
```

```
  return
```

```
<button style={{ backgroundColor: 'red', color: 'white' }}>
```

```
  Click Me
```

```
</button>;
```

```
}
```

```
export default RedButton;
```

Step 2: App.jsx Me Component Use Karo

```
import React from 'react';
```

```
import RedButton from './RedButton';
```

```
function App() {
```

```
  return (
```

```
    <div>
```

```
      <h1>Welcome to My App</h1>
```

```
      <RedButton /><RedButton />
```

```
    </div>
```

```
  );
```

```
}
```

```
export default App;
```

Ab tumne ek reusable component bana diya, jo kahin bhi use ho sakta hai!

What is JSX? (JavaScript XML)

- ☞ JSX ek syntax extension hai jo HTML ko JavaScript me likhne ki suvidha deta hai.
- ☞ JSX ka use React me UI components likhne ke liye hota hai.

Real-Life Example:

Socho tum React likh rahe ho bina JSX ke!

- Tumhe JavaScript me document.createElement() use karna padega har element ke liye!
- JSX isse easy bana deta hai, jaise hum normally HTML likhte hain!

Example: JSX Ka Use

```
import React from 'react';
```

```
function App() {  
  const name = 'Shaaz';  
  
  return  
  <h1>Hello, {name}!</h1>;  
}
```

```
export default App;
```

JSX me curly brackets {} ke andar JavaScript likh sakte hain!

Rules of JSX

1. Har JSX Code Ko Ek Parent Element Me Wrap Karna

Zaroori Hai!

```
function App() {  
  return (  
    <div>  
      <h1>Hello, World!</h1>  
      <p>Welcome to React!</p>  
    </div>  
  );  
}
```

Har JSX return statement ek hi parent element ke andar hona chahiye!

2. JSX Me if-else Ke Jagah Ternary Operator Ka Use Karein

✗ Wrong:

```
function App({ isLoggedIn }) {  
  if (isLoggedIn) {  
    return  
      <h1>Welcome Back!</h1>;  
  } else {  
    return  
      <h1>Please Sign In</h1>;  
  }  
}
```


✓ Correct (Using Ternary Operator):

```
function App({ isLoggedIn }) {  
  return (  
    <div>  
      {isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please Sign  
In</h1>}  
    </div>  
  );  
}
```

Ternary Operator React me zyada readable hota hai!

Short Summary

- ✓ React ek JavaScript library hai jo interactive user interfaces banane ke liye use hoti hai.
- ✓ React me hum Single Page Applications (SPA) banate hain jo fast aur smooth hoti hain.
- ✓ Component ek reusable UI piece hota hai jo baar-baar use kiya ja sakta hai.
- ✓ JSX ka use React me HTML likhne ke liye hota hai, jo JavaScript ke andar likha ja sakta hai.
- ✓ JSX ke kuch rules hote hain, jaise single parent element aur ternary operator ka use.

Props & Prop Drilling

What are Props in React?

☞ Props ka full form hota hai "Properties"

☞ Props ka use ek component se doosre component me data bhejne ke liye hota hai

☞ Props immutable hote hain – iska matlab hai ki component props ko change nahi kar sakta!

Real-Life Example:

Socho tum Pizza Order kar rahe ho

- Tum Domino's ko order doge (Props pass karoge)
- Domino's sirf wahi pizza bana ke dega jo tumne order kiya hai!

Matlab, props sirf data bhejne ka kaam karte hain, aur component use kar sakta hai par modify nahi kar sakta!

Props Kaise Use Karein?

☞ Props function components me directly {} ke andar receive hote hain.

☞ Props parent component se child component me pass hote hain.

Example: Props Ka Basic Use

Step 1: `User.jsx` Component Banao

```
import React from 'react';

function User({ name, age }) {
  return (
    <div>
      <h1>Name: {name}</h1>
      <p>Age: {age}</p>
    </div>
  );
}

export default User;
```

✔ Props ko curly brackets {} me destructure kiya gaya!

Step 2: `App.jsx` Me Props Pass Karo

```
import React from 'react';
import User from './User';

function App() {
  return <User name="Shaaz" age={24} />;
}

export default App;
```

✔ Ab name="Shaaz" aur age={24} User component me pass ho raha hai!

✔ React automatically User component me ye values bhej dega!

Example: Multiple Props Pass Karna

```
function User({ name, age, hobby }) {  
  return (  
    <div>  
      <h1>Name: {name}</h1>  
      <p>Age: {age}</p>  
      <p>Hobby: {hobby}</p>  
    </div>  
  );  
}
```

```
function App() {  
  return  
    <User name="Shaaz" age={24} hobby="Coding" />;  
}
```

export default App;

✓ Ab ek aur prop hobby add kar diya jo User component receive karega!

Props Immutable Hote Hain (Change Nahi Kar Sakte)

👉 Props ko child component ke andar modify nahi kar sakte!

❌ **Wrong:**

```
function User({ name, age }) {  
  name = "Asad"; // ❌ Props ko modify nahi kar sakte!  
  return <h1>{name}</h1>;  
}
```

✅ **Correct:**

```
function User({ name }) {  
  return <h1>{name}</h1>; // ✅ Props sirf read kar sakte hain!  
}
```

Props ek baar set ho jaye, toh child component usko change nahi kar sakta!

What is Prop Drilling?

👉 Prop Drilling ka matlab hai ek prop ko multiple components ke through pass karna, jabki uska use sirf ek deeply nested child component me ho.

👉 Jitne beech ke components hain, wo bas props ko aage pass karte hain bina use kiye!

Real-Life Example:

Socho tum Apne Papa se Paise maang rahe ho!

- Papa ne Dadaji se maanga!
- Dadaji ne ParDadaji se maanga!
- ParDadaji ne Tumhe finally de diya!

Example: Prop Drilling Problem

```
function GreatGrandParent() {  
  const message = "Hello from Great Grandparent!";  
  return <GrandParent message={message} />;  
}
```

```
function GrandParent({ message }) {  
  return <Parent message={message} />;  
}
```

```
function Parent({ message }) {  
  return <Child message={message} />;  
}
```

```
function Child({ message }) {  
  return <h1>{message}</h1>;  
}
```

```
export default GreatGrandParent;
```

Things to remember ?

- message ko har component manually pass kar raha hai!
- Agar 10 components ho, toh ye aur zyada complex ho jayega!

Prop Drilling Ka Solution

👉 Prop drilling ko avoid karne ke liye context API ya state management tools ka use karte hain!

👉 Context API ka use karke direct data pass kar sakte hain bina beech ke components ko involve kiye!

Solution: React Context API

```
import React, { createContext, useContext } from 'react';
```

```
// Step 1: Create Context
```

```
const MessageContext = createContext();
```

```
function GreatGrandParent() {  
  const message = "Hello from Great Grandparent!";  
  return (  
    <MessageContext.Provider value={message}>  
    <GrandParent /></MessageContext.Provider>  
  );  
}
```

```
function GrandParent() {  
  return <Parent />;  
}
```

```
function Parent() {  
  return <Child />;  
}
```


Conditional Rendering

What is Conditional Rendering in React?

- ☞ Matlab kisi condition ke base pe component ya element show ya hide karna!
- ☞ Agar koi user logged in hai toh "Welcome!" dikhana, warna "Please Log In" dikhana.
- ☞ React me hum multiple tareeke se Conditional Rendering kar sakte hain!

Real-Life Example:

Socho tum Netflix chal rahe ho!

- Agar tum logged in ho, toh "Continue Watching" wala section dikh raha hoga.
- Agar tum logged out ho, toh "Sign In to Continue" likha aayega.

Matlab, UI dynamically change ho rahi hai based on login status!

If/Else Statement (Sabse Basic Tarika)

👉 Yeh approach simple hai, lekin multiple return statements use karta hai jo best practice nahi hai!

Example: Using If/Else

```
function Greeting({ isLoggedIn }) {  
  if (isLoggedIn) {  
    return <h1>Welcome Back!</h1>;  
  } else {  
    return <h1>Please Log In</h1>;  
  }  
}
```

```
function App() {  
  return <Greeting isLoggedIn={true} />;  
}
```

export default App;

✅ Pros: Basic & Easy to Understand!

❌ Cons: Multiple return statements hone ki wajah se readability reduce hoti hai!

Using Element Variables

👉 Isme ek variable me JSX store karke use karte hain!

👉 Thoda better hai if/else se, par ab bhi best nahi hai!

Example: Using Element Variables

```
function Greeting({ isLoggedIn }) {  
  let message;  
  if (isLoggedIn) {  
    message = <h1>Welcome Back!</h1>;  
  } else {  
    message = <h1>Please Log In</h1>;  
  }  
  
  return <div>{message}</div>;  
}
```

```
function App() {  
  return <Greeting isLoggedIn={false} />;  
}
```

```
export default App;
```

✅ Pros: Ek hi return hai, readability improve hoti hai!

❌ Cons: Extra variable message banana padta hai, jo unnecessary lag sakta hai!

Using Ternary Operator (Recommended Approach)

👉 **Ternary Operator ? :** ka use karke hum ek concise aur readable tarika bana sakte hain!

👉 Ye ek hi line me conditional rendering kaam kar deta hai! 🚀

Example: Using Ternary Operator

```
function Greeting({ isLoggedIn }) {  
  return (  
    <h1>{isLoggedIn ? "Welcome Back!" : "Please Log In"}</h1>  
  );  
}
```

```
function App() {  
  return <Greeting isLoggedIn={true} />;  
}
```

export default App;

✅ Pros: Short aur zyada readable code!

✅ Cons: Sirf simple conditions ke liye best hai, complex cases me readability lose ho sakti hai!

Using Short Circuit Operator (&&)

☞ Agar bas ek hi condition check karni hai bina else part ke, toh yeh best hai!

☞ `true && <Component />` sirf tabhi render hoga jab condition true ho!

Example: Using Short Circuit (&&)

```
function Notification({ unreadMessages }) {  
  return (  
    <div>  
      <h1>Welcome to React!</h1>  
      {unreadMessages.length > 0 &&  
      <p>You have {unreadMessages.length} new messages.</p>}  
    </div>  
  );  
}
```

```
function App() {  
  return <Notification unreadMessages={["Message 1",  
"Message 2"]} />;  
}
```

export default App;

✅ Pros: Short aur effective jab else ka zaroorat na ho!

❌ Cons: Kabhi-kabhi code readability thodi down ho sakti hai!

Nested Conditions (Multiple Conditions Handle Karna)

👉 Agar tumhe ek se zyada condition check karni ho, toh nested ternary ka use kar sakte ho!

Example: Multiple Conditions

```
function Status({ isLoggedIn, isAdmin }) {  
  return (  
    <h1>  
      {isLoggedIn ? (isAdmin ? "Welcome, Admin!" : "Welcome,  
User!") : "Please Log In"}  
    </h1>  
  );  
}  
  
function App() {  
  return <Status isLoggedIn={true} isAdmin={false} />;  
}  
  
export default App;
```

✅ Pros: Ek hi line me multiple conditions handle ho sakti hain!

❌ Cons: Zyada complex ho jaye toh readability khatam ho sakti hai!

React Hooks

What are Hooks in React?

👉 Hooks ek tarika hai jisse functional components me state aur lifecycle methods ka use kiya jata hai.

👉 Hooks se functional components zyada dynamic aur re-usable bante hain.

👉 Hooks ka naam hamesha "use" se start hota hai, jaise useState, useEffect, useRef etc.

Real-Life Example:

Socho tum gaming khel rahe ho , aur tumhare pass cheat codes hain jo game easy bana dete hain!

- Hooks bhi React ke "cheat codes" hain jo functional components ko zyada powerful banate hain!

useState() – React ka Sabse Important Hook!

👉 useState ka use component me state manage karne ke liye hota hai!

👉 Jab bhi state update hoti hai, component automatically re-render hota hai!

💡 Real-Life Example:

Socho tum Instagram pe ho!

- Agar tum "Like ❤️" button dabaoge, toh instantly count update ho jata hai!
- Yeh magic "state" ki wajah se hota hai!

Example: Simple Counter App

```
import React, { useState } from "react";
```

```
function Counter() {  
  const [count, setCount] = useState(0);  
  
  return (  
    <div>  
      <h2>Count: {count}</h2>  
      <button  
        onClick={() => setCount(count + 1)}>  
        Increment  
      </button>  
      <button  
        onClick={() => setCount(count - 1)}>Decrement  
      </button>  
    </div>  
  );  
}
```

```
export default Counter;
```

Har baar button click hone par count update hoga aur UI re-render hoga!

useEffect() – Lifecycle Hook!

👉 useEffect ka use side effects (jaise API calls, timers, events) handle karne ke liye hota hai!

👉 Class components me `componentDidMount`, `componentDidUpdate`, `componentWillUnmount` hota tha, par hooks me sab useEffect se handle hota hai!

```
import React, { useState, useEffect } from "react";
```

```
function FetchData() {
```

```
  const [data, setData] = useState([]);
```

```
  useEffect(() => {
```

```
    fetch("https://jsonplaceholder.typicode.com/posts")
```

```
      .then((response) => response.json())
```

```
      .then((json) => setData(json));
```

```
  }, []);
```

```
  return (
```

```
    <div>
```

```
      <h2>Fetched Data:</h2>
```

```
      <ul>
```

```
        {data.slice(0, 5).map((item) => (
```

```
          <li key={item.id}>{item.title}</li>
```

```
        )))
```

```
      </ul>
```

```
    </div>
```

```
  );
```

```
}
```

```
export default FetchData;
```


useRef() – DOM Reference Hold Karne Ka Hook!

👉 useRef se tum kisi bhi DOM element ya variable ka reference hold kar sakte ho bina re-render kare!

```
import React, { useRef } from "react";

function FocusInput() {
  const inputRef = useRef(null);

  const handleFocus = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef}
        type="text" placeholder="Type something..." />
      <button onClick={handleFocus}>Focus Input</button>
    </div>
  );
}

export default FocusInput;
```

useRef se input field ka reference store kiya aur bina re-render kare usko focus kiya!

useReducer() – useState Ka Advanced Version!

👉 useReducer tab use hota hai jab state ka logic complex ho!

```
import React, { useReducer } from "react";
```

```
const initialState = 0;
```

```
const reducer = (state, action) => {
```

```
  switch (action.type) {
```

```
    case "increment":
```

```
      return state + 1;
```

```
    case "decrement":
```

```
      return state - 1;
```

```
    default:
```

```
      return state;
```

```
  }
```

```
};
```

```
function Counter() {
```

```
  const [count, dispatch] = useReducer(reducer, initialState);
```

```
  return (
```

```
    <div>
```

```
      <h2>Count: {count}</h2>
```

```
      <button onClick={() => dispatch({ type: "increment"
```

```
    })}>Increment</button>
```

```
      <button onClick={() => dispatch({ type: "decrement"
```

```
    })}>Decrement</button>
```

```
    </div>
```

```
  );
```

```
}
```

```
export default Counter;
```


useReducer() – useState Ka Advanced Version!

👉 useReducer tab use hota hai jab state ka logic complex ho!

```
import React, { useReducer } from "react";
```

```
const initialState = 0;
```

```
const reducer = (state, action) => {
```

```
  switch (action.type) {
```

```
    case "increment":
```

```
      return state + 1;
```

```
    case "decrement":
```

```
      return state - 1;
```

```
    default:
```

```
      return state;
```

```
  }
```

```
};
```

```
function Counter() {
```

```
  const [count, dispatch] = useReducer(reducer, initialState);
```

```
  return (
```

```
    <div>
```

```
      <h2>Count: {count}</h2>
```

```
      <button onClick={() => dispatch({ type: "increment"
```

```
    })}>Increment</button>
```

```
      <button onClick={() => dispatch({ type: "decrement"
```

```
    })}>Decrement</button>
```

```
    </div>
```

```
  );
```

```
}
```

```
export default Counter;
```

✅ Yahan dispatch() function se actions bhej rahe hain, jo reducer() handle karega!

useMemo() – Performance Optimization Hook!

👉 useMemo() expensive calculations optimize karta hai taaki baar-baar calculate na karna pade!

Example: Expensive Calculation Optimization

```
import React, { useState, useMemo } from "react";
```

```
function ExpensiveCalculation({ num }) {
```

```
  const slowFunction = (num) => {
```

```
    console.log("Running Slow Function...");
```

```
    return num * 2;
```

```
  };
```

```
  const memoizedValue = useMemo(() => slowFunction(num),  
[num]);
```

```
  return <h2>Result: {memoizedValue}</h2>;
```

```
}
```

```
export default ExpensiveCalculation;
```

✅ useMemo() function unnecessary re-runs avoid karega!

useCallback() – Function Memoization Hook!

👉 useCallback() function ko cache karta hai, jisse unnecessary re-renders avoid hote hain!

Example: Preventing Unnecessary Re-renders

```
import React, { useState, useCallback } from "react";
```

```
function Button({ handleClick }) {  
  return <button onClick={handleClick}>Click Me</button>;  
}
```

```
function App() {  
  const [count, setCount] = useState(0);  
  
  const memoizedFunction = useCallback(() => {  
    setCount((prev) => prev + 1);  
  }, []);
```

```
  return (  
    <div><h2>Count: {count}</h2>  
    <Button handleClick={memoizedFunction} />  
  </div>  
);  
}
```

```
export default App;
```

✅ useCallback() function ko cache karega taaki unnecessary re-renders na ho!

Custom Hooks

What are Custom Hooks in React?

- 👉 Custom Hooks ek tarika hai apni logic ko reusable banane ka!
- 👉 Agar kisi logic ka use multiple jagah hota hai, toh usko ek hook me likh ke baar-baar reuse kar sakte hain!
- 👉 Custom Hooks ek normal function ki tarah hote hain, but inka naam use se start hota hai (jaise useFetch, useLocalStorage etc.)

💡 Real-Life Example:

Socho tum ek YouTube Video Editor ho!

- Har video pe tum "Subscribe Now!" ka watermark lagate ho.
- Agar tum ye manually har video me karoge, toh bahut time lagega! ⌚
- Ek automated tool (custom hook) bana lo jo ye kaam kare, toh mast ho jayega!

Custom Hooks bhi isi tarah React me code ko optimize aur reusable banate hain!

Why Use Custom Hooks?

- 👉 Agar ek hi logic multiple components me repeat ho rahi hai, toh usko ek custom hook me likho aur reuse karo!
- 👉 Ye code ko clean aur maintainable banata hai!
- 👉 Ekbaar likhne ke baad usko har jagah use kar sakte ho bina copy-paste kiye!

Creating a Custom Hook

👉 Ek custom hook ek normal function ki tarah hota hai jo React Hooks ka use karta hai.

👉 Iska naam hamesha use se start hota hai, jaise useFetch, useTheme, useCounter etc.

Example: useCounter Hook (Re-usable Counter Logic)

Step 1: Custom Hook Banao – useCounter.js

```
import { useState } from "react";
```

```
function useCounter(initialValue = 0) {  
  const [count, setCount] = useState(initialValue);
```

```
  const increment = () => setCount(count + 1);
```

```
  const decrement = () => setCount(count - 1);
```

```
  const reset = () => setCount(0);
```

```
  return { count, increment, decrement, reset };  
}
```

```
export default useCounter;
```

✅ Yeh ek custom hook hai jo counter logic handle karega!

✅ Isme useState ka use kiya hai aur count, increment, decrement, reset return kiya hai!

Step 2: Custom Hook Ka Use Karo – App.jsx

```
import React from "react";
```

```
import useCounter from "../useCounter";
```

```
function App() {
```

```
  const { count, increment, decrement, reset } =
```

```
  useCounter(10);
```

```
  return (
```

```
    <div>
```

```
    <h1>Custom Counter Hook</h1>
```

```
    <h2>Count: {count}</h2>
```

```
    <button onClick={increment}>Increment</button>
```

```
    <button onClick={decrement}>Decrement</button><button
```

```
    onClick={reset}>Reset</button></div>
```

```
  );
```

```
}
```

```
export default App;
```

✅ Ab useCounter() ko har jagah use kar sakte hain bina same code likhe!

Custom Hook for Fetching API Data (useFetch)

👉 Agar multiple components API se data fetch kar rahe hain, toh us logic ko ek custom hook me likho!

📌 Step 1: useFetch.js – Custom Hook

```
import { useState, useEffect } from "react";
```

```
function useFetch(url) {  
  const [data, setData] = useState(null);  
  const [loading, setLoading] = useState(true);  
  const [error, setError] = useState(null);  
  
  useEffect(() => {  
    async function fetchData() {  
      try {  
        let response = await fetch(url);  
        let result = await response.json();  
        setData(result);  
        setLoading(false);  
      } catch (err) {  
        setError(err);  
        setLoading(false);  
      }  
    }  
    fetchData();  
  }, [url]);  
  
  return { data, loading, error };  
}
```

```
export default useFetch;
```

✅ Yeh hook kisi bhi API se data fetch karega aur data, loading, error return karega!

Step 2: App.jsx – Hook Ko Use Karo

import **React** from "react";

import **useFetch** from **"./useFetch"**;

function **App**() {

const { **data**, **loading**, **error** } = **useFetch**(
 "https://jsonplaceholder.typicode.com/posts"
);

if (**loading**) return **<h2>Loading...</h2>**;

if (**error**) return **<h2>Error fetching data!</h2>**;

return (

<div>

<h1>Fetched Data:</h1>

{**data.slice(0, 5).map**((**item**) => (
 <li key={item.id}>{item.title}
))}

</div>

);

}

export default **App**;

✅ Ab har jagah useFetch() use karke API fetch kar sakte ho
bina extra code likhe!

Custom Hook for Local Storage (useLocalStorage)

👉 Agar tumhe data ko browser ke local storage me save karna ho, toh har jagah localStorage.setItem likhna padega! 😫

👉 Isko avoid karne ke liye ek custom hook useLocalStorage bana lo!

📌 Step 1: useLocalStorage.js

```
import { useState } from "react";
```

```
function useLocalStorage(key, initialValue) {  
  const [storedValue, setStoredValue] = useState(() => {  
    const savedValue = localStorage.getItem(key);  
    return savedValue ? JSON.parse(savedValue) :  
initialValue;  
  });
```

```
  const setValue = (value) => {  
    setStoredValue(value);  
    localStorage.setItem(key, JSON.stringify(value));  
  };  
  
  return [storedValue, setValue];  
}
```

```
export default useLocalStorage;
```

✅ Yeh hook local storage se data save/retrieve karega!

Step 2: App.jsx – Hook Ko Use Karo

```
import React from "react";
```

```
import { useState } from "react";
```

```
function App() {
```

```
  const [name, setName] = useState("");
```

```
  return (
```

```
    <div>
```

```
      <h1>Local Storage Hook</h1>
```

```
      <input type="text"
```

```
        placeholder="Enter your name"
```

```
        value={name}
```

```
        onChange={(e) => setName(e.target.value)}
```

```
      />
```

```
      <p>Saved Name: {name}</p></div>
```

```
    );
```

```
  }
```

```
export default App;
```

✅ Ab browser ko refresh karne ke baad bhi input field me value rahegi!

Context API

What is Context API in React?

👉 Context API ek tarika hai jisse ek global state create karke multiple components ke beech data share kiya jata hai, bina props drill kiye!

👉 Ye ek React ke built-in feature hai, jo Redux jaise external state management tools ka lightweight alternative hai!

💡 Real-Life Example:

Socho tum McDonald's 🍔 me ho, aur ek billing counter hai jo sab orders process karta hai!

- Agar har ek table pe ek separate billing machine ho, toh confusion badh jayegi!
- Isliye ek centralized counter (Context API) hota hai, jo sab tables (components) ke liye orders process karta hai!

Matlab Context API ek centralized store hai jo har component ke liye easily accessible hota hai!

Problem Without Context API (Prop Drilling)

👉 Agar ek deeply nested component ko data chahiye, toh hume parent se child, child se uske child tak props pass karna padta hai!

Example: Prop Drilling Problem

```
function App() {  
  const user = "Shaaz";  
  
  return <Parent user={user} />;  
}  
  
function Parent({ user }) {  
  return <Child user={user} />;  
}  
  
function Child({ user }) {  
  return <GrandChild user={user} />;  
}  
  
function GrandChild({ user }) {  
  return <h2>Welcome, {user}!</h2>;  
}  
  
export default App;
```

✅ Har component me user ko manually props ke through pass karna pad raha hai!

⚠ Problem:

- Agar humare components aur deeply nested hote, toh prop drilling aur jyada complex ho jata!

Solution: Context API!

👉 Context API se ek baar global store bana do, fir koi bhi component us data ko directly access kar sakta hai, bina prop drilling kiye!

📌 Step 1: Create a Context (Global Store)

```
import React, { createContext } from "react";
```

```
// Context banaya
```

```
const UserContext = createContext();
```

```
export default UserContext;
```

✅ Ye ek global state hai jo har component ke liye accessible hogi!

📌 Step 2: Wrap Components Inside a Provider

```
import React, { useState } from "react";
```

```
import UserContext from "../UserContext";
```

```
import Child from "../Child";
```

```
function App() {
```

```
  const [user, setUser] = useState("Shaaz");
```

```
  return (
```

```
    <UserContext.Provider value={user}>
```

```
      <Child />
```

```
    </UserContext.Provider>
```

```
  );
```

```
}
```

```
export default App;
```

✅ Ab UserContext.Provider ke andar jitne bhi components honge, wo user value ko access kar sakenge!

Step 3: Access Context in Child Components

```
import React, { useContext } from "react";  
  
import UserContext from "../UserContext";  
  
function Child() {  
  const user = useContext(UserContext);  
  return <h2>Welcome, {user}!</h2>;  
}  
  
export default Child;
```

✅ Ab useContext(UserContext) se bina props pass kiye user ko access kar sakte hain!

Updating Context Value (Theme Switcher Example)

👉 Context API sirf data share karne ke liye nahi, balki state update karne ke liye bhi use hota hai!

👉 Ek toggleTheme function bana kar, hum dark/light mode switch kar sakte hain! 🚀

Step 1: Create Theme Context

```
import React, { createContext, useState } from "react";

const ThemeContext = createContext();

export function ThemeProvider({ children }) {
  const [theme, setTheme] = useState("light");

  const toggleTheme = () => {
    setTheme(theme === "light" ? "dark" : "light");
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

export default ThemeContext;
```

✔ Yeh ek global theme context hai jo sab components ko theme aur toggleTheme function provide karega!

Step 2: Wrap App Inside ThemeProvider

```
import React from "react";
```

```
import { ThemeProvider } from "../ThemeContext";
```

```
import ThemeSwitcher from "../ThemeSwitcher";
```

```
function App() {
```

```
  return (
```

```
    <ThemeProvider><ThemeSwitcher /></ThemeProvider>
```

```
  );
```

```
}
```

```
export default App;
```



Ab har component theme aur toggleTheme function use kar sakta hai!

Step 3: Access Theme Context & Toggle Theme

```
import React, { useContext } from "react";

import ThemeContext from "../ThemeContext";

function ThemeSwitcher() {

  const { theme, toggleTheme } =
    useContext(ThemeContext);

  return (

    <div style={{backgroundColor: theme === "light" ? "#fff" :
"#333",

      color: theme === "light" ? "#000" : "#fff",
      padding: "20px",
      textAlign: "center"
    }}>

      <h2>Current Theme: {theme}</h2>

      <button onClick={toggleTheme}>Toggle Theme</button>

    </div>

  );
}

export default ThemeSwitcher;
```

✓ Ab button click karne pe theme switch ho jayega!

React Forms

What are Forms in React?

- ☞ Forms ka use user se input collect karne ke liye hota hai!
- ☞ HTML forms me data directly DOM se liya jata hai, but React me state ke through manage hota hai!
- ☞ Forms ko efficiently handle karne ke liye React me do approaches hain – Controlled aur Uncontrolled Components!

Real-Life Example:

Socho tum Pizza Order Kar Rahe Ho!

- Tumhe Name, Address, Pizza Type aur Payment Details bharni hai.
- Agar tumne form submit kiya aur wo data lost ho gaya, toh order cancel ho sakta hai!
- Isliye React Forms me data ko state me store kiya jata hai, taaki kahin bhi lost na ho!

Controlled Components – State Se Form Control Karo!

☞ Controlled components wo hote hain jo React state se fully controlled hote hain.

☞ Matlab jo bhi input field me likhoge, wo React state me store hoga aur React ke control me rahega.

Example: Simple Controlled Input Form

```
import React, { useState } from "react";
```

```
function ControlledForm() {  
  const [name, setName] = useState("");  
  
  const handleChange = (event) => {  
    setName(event.target.value);  
  };  
  
  const handleSubmit = (event) => {  
    event.preventDefault();  
    alert(`Hello, ${name}!`);  
  };  
}
```



```
return (  
  <form onSubmit={handleSubmit}>  
    <label>Name: </label>  
    <input type="text" value={name} onChange={handleChange}  
    />  
    <button type="submit">Submit</button>  
  </form>  
  );  
}  
export default ControlledForm;
```

- ✓ Yahan name ka value state me store ho raha hai, aur user jo bhi type karega, wo update hoga!
- ✓ onChange event state ko update kar raha hai, aur onSubmit se alert show ho raha hai!

Handling Multiple Input Fields in Forms

- ☞ Agar form me multiple fields hain (jaise email, password, address), toh har ek ke liye useState nahi banana chahiye!
- ☞ Instead, ek single object state bana ke usko dynamically update karna better hota hai!

Example: Multi-Field Controlled Form

```
import React, { useState } from "react";
```

```
function MultiFieldForm() {  
  const [formData, setFormData] = useState({  
    name: "",  
    email: "",  
    password: "",  
  });  
  
  const handleChange = (event) => {  
    setFormData({ ...formData, [event.target.name]:  
event.target.value });  
  };  
  
  const handleSubmit = (event) => {  
    event.preventDefault();  
    alert(`Welcome, ${formData.name}! Your email is  
${formData.email}`);  
  };  
}
```



```
return (  
  <form  
    onSubmit={handleSubmit}>  
    <label>Name: </label>  
    <input type="text" name="name" value={formData.name}  
      onChange={handleChange} />  
  
    <label>Email: </label>  
    <input type="email" name="email" value={formData.email}  
      onChange={handleChange} />  
  
    <label>Password: </label>  
    <input type="password" name="password" value=  
      {formData.password}  
      onChange={handleChange} />  
  
    <button type="submit">Submit</button>  
  </form>  
);  
}  
export default MultiFieldForm;
```

✅ Ek single state object me name, email aur password store ho raha hai!

✅ handleChange function dynamically values update kar raha hai using [event.target.name]!

Uncontrolled Components – DOM Se Direct Data Lo!

☞ Uncontrolled components me form elements ka state

React se manage nahi hota, balki directly DOM se data fetch hota hai!

☞ Isme useRef() ka use karke input field ka reference liya jata hai.

Example: Uncontrolled Form Using useRef

```
import React, { useRef } from "react";

function UncontrolledForm() {

  const nameRef = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Hello, ${nameRef.current.value}!`);
  };

  return (
    <form
      onSubmit={handleSubmit}>
      <label>Name: </label>
      <input type="text" ref={nameRef} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

export default UncontrolledForm;

✅ useRef() se input field ka reference leke value fetch ho rahi hai!

✅ React state use nahi ho rahi, directly DOM se data fetch ho raha hai!

Difference Between Controlled & Uncontrolled

Components

- ✓ **Controlled Components:** Form data React state me store hota hai!
- ✓ **Uncontrolled Components:** Data directly DOM se fetch hota hai, React state ka use nahi hota!
- ✓ Controlled Components jyada predictable aur recommended hote hain, kyunki data React ke control me rehta hai! 🔥

Handling Checkbox, Radio & Select Dropdown

👉 Forms me sirf text fields nahi hote, balki checkboxes, radio buttons aur dropdowns bhi hote hain!

Example: Checkbox & Radio Button Handling

```
import React, { useState } from "react";
```

```
function PreferencesForm() {  
  const [preferences, setPreferences] = useState({  
    newsletter: false,  
    gender: "",  
  });  
  
  const handleChange = (event) => {  
    const { name, type, checked, value } = event.target;  
    setPreferences({  
      ...preferences,  
      [name]: type === "checkbox" ? checked : value,  
    });  
  };  
};
```



```
return (  
  <form>  
    <label>  
      <input type="checkbox" name="newsletter" checked=  
        {preferences.newsletter} onChange={handleChange} />  
      Subscribe to Newsletter  
    </label>  
  
    <label>  
      <input type="radio" name="gender" value="male" checked=  
        {preferences.gender === "male"} onChange={handleChange} />  
      Male  
    </label>  
  
    <label>  
      <input  
        type="radio" name="gender" value="female" checked=  
        {preferences.gender === "female"}  
        onChange={handleChange} />  
      Female  
    </label>  
  
    <p>  
      Newsletter: {preferences.newsletter ? "Subscribed" : "Not  
      Subscribed"}  
    </p>  
  
    <p>Gender: {preferences.gender}</p></form>  
  );  
}  
  
export default PreferencesForm;
```

✔ Checkbox ke liye checked ka use kiya hai, aur radio button ke liye value check kiya hai!

Creating a Registration Form in React

👉 Sabse pehle ek simple form banayenge jo user se Name, Email, aur Password lega!

Example: Basic Registration Form

```
import React, { useState } from "react";
```

```
function RegistrationForm() {
```

```
  const [formData, setFormData] = useState({
```

```
    name: "",
```

```
    email: "",
```

```
    password: "",
```

```
    confirmPassword: "",
```

```
  });
```

```
  const handleChange = (event) => {
```

```
    const { name, value } = event.target;
```

```
    setFormData({ ...formData, [name]: value });
```

```
  };
```

```
  const handleSubmit = (event) => {
```

```
    event.preventDefault();
```

```
    alert(`Welcome, ${formData.name}! Your email is  
${formData.email}`);
```

```
  };
```



```
return (  
  <form  
    onSubmit={handleSubmit}>  
  
    <label>Name: </label>  
    <input type="text" name="name" value={formData.name}  
onChange={handleChange} required />  
  
    <label>Email: </label>  
    <input type="email" name="email" value={formData.email}  
onChange={handleChange} required />  
  
    <label>Password: </label>  
    <input type="password" name="password" value=  
{formData.password} onChange={handleChange} required />  
    <label>Confirm Password: </label>  
    <input type="password" name="confirmPassword" value=  
{formData.confirmPassword} onChange={handleChange}  
required />  
  
    <button type="submit">Register</button>  
  </form>  
);  
}  
  
export default RegistrationForm;
```

✅ Yahan har input field ka value state me store ho raha hai, aur onChange se update ho raha hai! 🔄

✅ Form submit hone par alert me naam aur email show ho raha hai!

Adding Basic Form Validations

👉 Ab validation lagayenge taaki user galat data na bhare!

👉 Yeh check karenge ki:

✓ Koi bhi field empty na ho

✓ Email format sahi ho (@ aur . hona chahiye)

✓ Password aur Confirm Password match karein

Example: Form with Validations

```
import React, { useState } from "react";
```

```
function RegistrationForm() {
```

```
  const [formData, setFormData] = useState({
```

```
    name: "",
```

```
    email: "",
```

```
    password: "",
```

```
    confirmPassword: "",
```

```
  });
```

```
  const [errors, setErrors] = useState({});
```

```
  const validate = () => {
```

```
    let newErrors = {};
```

```
    if (!formData.name.trim()) {
```

```
      newErrors.name = "Name is required!";
```

```
    }
```



```
if (!formData.email.includes("@") ||
!formData.email.includes("."))
{ newErrors.email = "Enter a valid email!"; }

if (formData.password.length < 6)
{ newErrors.password = "Password must be at least 6
characters!";
}

if (formData.password !== formData.confirmPassword) {
newErrors.confirmPassword = "Passwords do not match!";
}

setErrors(newErrors);
return Object.keys(newErrors).length === 0; };

const handleChange = (event) => {
setFormData({ ...formData, [event.target.name]:
event.target.value });
};

const handleSubmit = (event) =>
{
event.preventDefault();

if (validate()) {
alert(`Welcome, ${formData.name}! Your registration is
successful.`); }
};
```



```
return (  
  
<form  
  
onSubmit={handleSubmit}>  
  
<label>Name: </label>  
  
<input type="text" name="name" value={formData.name}  
onChange={handleChange} />  
  
{errors.name && <p style={{ color: "red" }}>{errors.name}</p>}  
  
  
<label>Email: </label>  
  
<input type="email" name="email" value={formData.email}  
onChange={handleChange} />  
  
{errors.email && <p style={{ color: "red" }}>{errors.email}</p>}  
  
  
  
<label>Password: </label>  
  
<input type="password" name="password" value=  
{formData.password}  
onChange={handleChange} />  
  
{errors.password && <p style={{ color: "red" }}>{errors.password}  
</p>}  
  
  
  
<label>Confirm Password: </label>  
  
<input type="password" name="confirmPassword" value=  
{formData.confirmPassword}  
onChange={handleChange} />  
  
{errors.confirmPassword && <p style={{ color: "red" }}>  
{errors.confirmPassword}</p>}  
  
  
  
<button type="submit">Register</button>  
  
</form>  
  
) ; }
```


Rendering a List

What is List Rendering in React?

👉 List rendering ka matlab hai kisi bhi array ke multiple items ko loop karke JSX me display karna!

👉 React me hum `map()` ka use karte hain jo har ek item ke liye ek JSX element return karta hai!

💡 Real-Life Example:

Socho tum IPL ke sabhi teams ka naam ek list me dikhana chahte ho! 🖊️

- Har ek team ek `h1` tag me dikhna chahiye!
- React me iske liye `map()` ka use karenge!

Basic List Rendering in React

👉 Sabse pehle ek simple example dekhte hain jisme IPL teams ka naam render karenge! ✅

👁️ Example: Rendering List of IPL Teams


```
import React from "react";
```

```
function IPLTeams() {
```

```
  const IPL = ["Mumbai Indians", "Chennai Super Kings", "Royal  
Challengers Bangalore", "Delhi Capitals"];
```

```
  return (
```

```
    <div>
```

```
      <h2>IPL Teams:</h2>
```

```
      {IPL.map((team, index) => (
```

```
        <h1 key={index}>{team}</h1>
```

```
      )))
```

```
    </div>
```

```
  );
```

```
}
```

```
export default IPLTeams;
```

✓ map() se har ek IPL team ko ek h1 tag me render kar diya!

✓ Har element ko key={index} diya jo React ko batata hai ki ye ek unique item hai!

Using Key in React Lists

- 👉 React me key ek special attribute hota hai jo har list item ko unique banata hai!
- 👉 Agar key unique nahi hai, toh React ko properly DOM update karne me problem hoti hai! ⚠️

Example: Agar keys same ho, toh kya hoga?

```
{IPL.map(({team}) => (  
  <h1 key="1">{team}</h1> // ⚠️ Har ek element ka same key  
  hone se React confused ho jayega!  
))}
```

⚠️ Problem:

- React ko ye samajhne me dikkat hogi ki kaunsa item change ho raha hai!
- Ye React ki performance aur rendering ko affect karega!

✓ Solution: Har item ko ek unique key do!

Example: Using Unique ID as Key (Best Practice)

```
import React from "react";
```

```
function IPLTeams() {  
  const IPL = [  
    { id: 1, name: "Mumbai Indians" },  
    { id: 2, name: "Chennai Super Kings" },  
    { id: 3, name: "Royal Challengers Bangalore" },  
    { id: 4, name: "Delhi Capitals" },  
  ];  
  
  return (  
    <div>  
      <h2>IPL Teams:</h2>  
      {IPL.map((team) => (  
        <h1 key={team.id}>{team.name}</h1>  
      ))}  
    </div>  
  );  
}  
  
export default IPLTeams;
```

✓ Har team ka ek unique id diya, jo key ke liye best practice hai!

Rendering a List of Objects (More Data)

👉 Agar tumhare array ke andar multiple properties ho (jaise team ka naam aur wins), toh kaise render karoge?

👁️ **Example: Rendering List of IPL Teams with Wins**

```
import React from "react";
```

```
function IPLTeams() {
```

```
  const IPL = [
```

```
    { id: 1, name: "Mumbai Indians", wins: 5 },
```

```
    { id: 2, name: "Chennai Super Kings", wins: 5 },
```

```
    { id: 3, name: "Royal Challengers Bangalore", wins: 0 },
```

```
    { id: 4, name: "Delhi Capitals", wins: 0 },
```

```
  ];
```

```
  return (
```

```
    <div>
```

```
      <h2>IPL Teams & Wins:</h2><ul>
```

```
        {IPL.map((team) => (
```

```
          <li key={team.id}>
```

```
            {team.name} - 🏆 {team.wins} Titles
```

```
          </li>
```

```
        )))
```

```
      </ul>
```

```
    </div>
```

```
  );
```

```
}
```

```
export default IPLTeams;
```

✅ Ab har team ka naam aur uske wins ko list (ul) ke andar show kiya!

Conditional Rendering with Lists

👉 Agar tumhe sirf wo teams dikhani ho jinme 1 se jyada wins hain, toh filter() ka use karo!

Example: Filtering and Rendering List

```
import React from "react";

function IPLWinners() {

  const IPL = [

    { id: 1, name: "Mumbai Indians", wins: 5 },
    { id: 2, name: "Chennai Super Kings", wins: 5 },
    { id: 3, name: "Royal Challengers Bangalore", wins: 0 },
    { id: 4, name: "Delhi Capitals", wins: 0 },
  ];

  const winners = IPL.filter((team) => team.wins > 0);

  return (
    <div>

      <h2>IPL Winners:</h2>

      <ul>

        {winners.map((team) => (
          <li key={team.id}>

            {team.name} - 🏆 {team.wins} Titles

          </li>

        ))}

      </ul>

    </div>

  );
}

export default IPLWinners;
```

✅ Ab sirf wo teams show ho rahi hain jinke pass wins > 0 hain!

React Router

What is React Router?

👉 React Router ek library hai jo tumhe React app me multiple pages banane aur navigate karne ki facility deti hai!

👉 Ye bina page reload kiye pages switch karne deta hai!

Real-Life Example:

Socho tum Netflix dekh rahe ho!

- Agar tum "Home" se "Movies" tab pe click karte ho, toh pura page reload nahi hota!
- Sirf content change hota hai, aur URL bhi update ho jata hai!
- Ye magic hota hai React Router ka!

Install React Router

👉 Sabse pehle React Router ko install karna padega!

npm install react-router-dom

✅ Ab tum React Router ka use kar sakte ho!

Basic Routing Setup

👉 Sabse pehle ek basic routing setup banayenge!

Example: Basic Routing

```
import React from "react";

import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";

function Home() {
  return <h2><img alt="house icon" data-bbox="188 175 212 190"/> Welcome to Home Page</h2>;
}

function About() {
  return <h2><img alt="book icon" data-bbox="188 288 212 303"/> About Us</h2>;
}

function Contact() {
  return <h2><img alt="phone icon" data-bbox="188 398 212 413"/> Contact Us</h2>;
}

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link> |
        <Link to="/about">About</Link> |
        <Link to="/contact">Contact</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </Router>
  );
}

export default App;
```


- ✔️ BrowserRouter se routing enable hoti hai!
- ✔️ Routes ke andar multiple Route hote hain jo path aur component define karte hain!
- ✔️ Link ka use karke navigation create kiya, jo a tag se better hai kyunki page reload nahi hota!

Dynamic Routing in React Router

👉 Agar tumhe ek specific user ka profile dikhana ho, toh kaise karoge?

👉 Iske liye dynamic routing ka use hota hai!

Example: Dynamic Routing

```
import React from "react";

import { BrowserRouter as Router, Routes, Route, Link,
useParams } from "react-router-dom";

function UserProfile() {

  const { username } = useParams(); // URL se username get
  karna

  return

  <h2>👤 Welcome, {username}!</h2>;

}
```



```
function App() {  
  return (  
    <Router>  
      <nav>  
        <Link to="/user/Shaaaz">Shaaz's Profile</Link> |  
        <Link to="/user/Asad">Asad's Profile</Link>  
      </nav>  
      <Routes>  
        <Route path="/user/:username" element={<UserProfile  
/>} />  
      </Routes>  
    </Router>  
  );  
}
```

export default App;

✓ Agar /user/Shaaaz open karoge, toh "Welcome, Shaaz!"
dikhenga!

✓ Agar /user/Asad open karoge, toh "Welcome, Asad!"
dikhenga!

Redirecting & Not Found Page (404)

👉 Agar user koi invalid page open kare toh usko 404 Page Not Found dikhana chahiye!

👉 Agar kisi page pe automatically redirect karna ho toh Navigate ka use karna hota hai!

Example: Redirect & 404 Page

```
import React from "react";
```

```
import { BrowserRouter as Router, Routes, Route, Navigate }  
from "react-router-dom";
```

```
function Home() {
```

```
  return <h2>🏠 Welcome to Home Page</h2>;
```

```
}
```

```
function NotFound() {
```

```
  return <h2>❌ 404 - Page Not Found</h2>;
```

```
}
```

```
function Dashboard() {
```

```
  const isLoggedIn = false; // Change this to true for access
```

```
  return
```

```
  isLoggedIn ? <h2>📊 Dashboard</h2> : <Navigate to="/" />;
```

```
}
```



```
function App() {  
  return (  
    <Router>  
      <Routes>  
        <Route path="/" element={<Home />} />  
        <Route path="/dashboard" element={<Dashboard />} />  
        <Route path="*" element={<NotFound />} />  
      </Routes>  
    </Router>  
  );  
}
```

export default App;

- ✓ Agar user /dashboard pe jaaye aur logged in na ho, toh use automatically home page pe redirect kar diya jayega!
- ✓ Agar user koi invalid URL pe jaaye, toh 404 - Page Not Found dikhega!

Nested Routes in React Router

👉 Agar tum ek page ke andar aur sub-pages banana chahte ho, toh nested routes ka use hota hai!

Example: Nested Routes

```
import React from "react";
```

```
import { BrowserRouter as Router, Routes, Route, Link, Outlet } from "react-router-dom";
```

```
function Dashboard() {
```

```
  return (
```

```
<div>
```

```
  <h2> 🏠 Dashboard</h2>
```

```
  <nav>
```

```
    <Link to="profile">Profile</Link> |
```

```
    <Link to="settings">Settings</Link>
```

```
  </nav>
```

```
  <Outlet /> { /* Ye nested routes ko dikhayega */ }
```

```
</div>
```

```
);
```

```
}
```

```
function Profile() {
```

```
  return <h3> 👤 User Profile</h3>;
```

```
}
```

```
function Settings() {
```

```
  return <h3> ⚙️ Settings</h3>;
```

```
}
```



```
function App() {  
  return (  
    <Router>  
      <Routes>  
        <Route path="/dashboard" element={<Dashboard />}>  
          <Route path="profile" element={<Profile />} />  
          <Route path="settings" element={<Settings />} />  
        </Route>  
      </Routes>  
    </Router>  
  );  
}  
  
export default App;
```

✅ Outlet ka use karke nested routes show kiye bina parent component ko modify kiye!

✅ Agar user /dashboard/profile pe jaaye toh User Profile dikhega!

✅ Agar user /dashboard/settings pe jaaye toh Settings dikhega!

Short Summary

✓ React Router bina page reload kiye multiple pages navigate karne me help karta hai!

✓ Basic Routing: Routes aur Route ka use karke multiple pages create kar sakte ho!

✓ Dynamic Routing: useParams() ka use karke URL se dynamic data fetch kar sakte ho!

✓ Redirect & 404: Navigate ka use karke user ko automatically kisi page pe bhej sakte ho!

✓ Nested Routes: Outlet ka use karke ek page ke andar multiple sub-pages show kar sakte ho!

React Redux

What is Redux?

- 👉 Redux ek state management library hai jo React me data ko centralize aur manage karne me help karti hai!
- 👉 Redux ek "Global Store" provide karta hai jisse kisi bhi component ko directly data mil sakta hai!
- 👉 Redux 3 cheezon pe kaam karta hai – Store, Actions aur Reducers!

Real-Life Example:

Socho tum ek shopping mall me ho!

- Mall ka ek centralized PA system (Redux Store) hota hai jo sabko announcement sunata hai!
- Mall ka manager (Action) announce karta hai ki "Sale chal rahi hai!"
- Mall ke har store ka employee (Reducer) us announcement ko sunn ke apna kaam karta hai!

Installing Redux in React

- 👉 Sabse pehle Redux ko install karna padega!

npm install @reduxjs/toolkit react-redux

- ✅ Ab tum Redux ka use kar sakte ho!

Redux Ki Basic Architecture

Redux 3 cheezon se milkar banta hai:

Store: Jisme data hota hai

Actions: Jo batate hain ki kya change karna hai

Reducers: Jo store ko update karte hain

Creating a Simple Redux Store in React

👉 Sabse pehle ek simple counter example banayenge jo Redux se manage hoga!

Step 1: Create Redux Store

```
// src/redux/store.js
```

```
import { configureStore } from "@reduxjs/toolkit";
```

```
import counterReducer from "../counterSlice";
```

```
const store = configureStore({  
  reducer: {  
    counter: counterReducer,  
  },  
});
```

```
export default store;
```

✅ Yahan configureStore() se ek Redux store banaya jo counterReducer use kar raha hai!

Step 2: Create Redux Slice (Reducers + Actions)

```
// src/redux/counterSlice.js
```

```
import { createSlice } from "@reduxjs/toolkit";
```

```
const counterSlice = createSlice({
```

```
  name: "counter",
```

```
  initialState: { count: 0 },
```

```
  reducers: {
```

```
    increment: (state) => {
```

```
      state.count += 1;
```

```
    },
```

```
    decrement: (state) => {
```

```
      state.count -= 1;
```

```
    },
```

```
    reset: (state) => {
```

```
      state.count = 0;
```

```
    },
```

```
  },
```

```
});
```

```
export const { increment, decrement, reset } =
```

```
counterSlice.actions;
```

```
export default counterSlice.reducer;
```

✅ createSlice() ka use karke counterSlice banaya jisme actions aur reducers dono hain!

✅ 3 actions hain – increment, decrement, aur reset jo state ko modify karenge!

Step 3: Provide Redux Store to React App

```
// src/main.jsx
```

```
import React from "react";
```

```
import ReactDOM from "react-dom/client";
```

```
import { Provider } from "react-redux";
```

```
import store from "../redux/store";
```

```
import App from "../App";
```

```
ReactDOM.createRoot(document.getElementById("root")).render(  
  <Provider store={store}>  
    <App />  
  </Provider>  
);
```

✓ Provider ka use karke Redux store ko poore React app me available banaya!

Step 4: Using Redux State and Dispatching Actions

```
// src/App.jsx
```

```
import React from "react";
```

```
import { useSelector, useDispatch } from "react-redux";
```

```
import { increment, decrement, reset } from  
"./redux/counterSlice";
```

```
function App() {
```

```
  const count = useSelector((state) => state.counter.count);
```

```
  const dispatch = useDispatch();
```

```
  return (
```

```
    <div>
```

```
      <h1>🔥 Redux Counter: {count}</h1>
```

```
      <button onClick={() => dispatch(increment())}>+
```

Increment

```
    </button>
```

```
    <button onClick={() => dispatch(decrement())}>-
```

Decrement

```
    </button>
```

```
    <button
```

```
      onClick={() => dispatch(reset())}>🔄 Reset
```

```
    </button>
```

```
  </div>
```

```
);
```

```
}
```

```
export default App;
```

✅ useSelector() ka use karke Redux store se count liya! 🔥

✅ useDispatch() ka use karke Redux ke actions ko call kiya! 279

Redux Toolkit with Async Thunk (API Call)

👉 Redux Toolkit ka ek powerful feature hai – `createAsyncThunk()`, jo API calls handle karta hai!

👉 Agar tumhe kisi API se data fetch karna ho aur Redux store me rakhna ho, toh ye best approach hai!

Example: Fetching API Data with Redux

```
// src/redux/userSlice.js
```

```
import { createSlice, createAsyncThunk } from  
"@reduxjs/toolkit";
```

```
// API Call using createAsyncThunk
```

```
export const fetchUsers =  
createAsyncThunk("users/fetchUsers", async () => {  
  const response = await  
fetch("https://jsonplaceholder.typicode.com/users");  
  return response.json();  
});
```



```
const userSlice = createSlice({  
  name: "users",  
  initialState: { users: [], loading: false },  
  reducers: {},  
  extraReducers: (builder) => {  
    builder.addCase(fetchUsers.pending, (state) => {  
      state.loading = true;  
    })  
    .addCase(fetchUsers.fulfilled, (state, action) => {  
      state.loading = false;  
      state.users = action.payload;  
    })  
    .addCase(fetchUsers.rejected, (state) => {  
      state.loading = false;  
    });  
  },  
});
```

export default **userSlice.reducer;**

✅ fetchUsers async action banaya jo API call karega aur Redux store me data save karega!

Using API Data in Component

```
// src/UserList.jsx
```

```
import React, { useEffect } from "react";

import { useSelector, useDispatch } from "react-redux";

import { fetchUsers } from "../redux/userSlice";

function UserList() {

  const dispatch = useDispatch();

  const { users, loading } = useSelector((state) => state.users);

  useEffect(() => {

    dispatch(fetchUsers());

  }, [dispatch]);

  return (

    <div>

      <h2>👤 User List</h2>

      {loading ?

        <p>Loading...</p> : users.map((user) =>

          <p key={user.id}>{user.name}</p>

        )}

    </div>

  );

}
```

```
export default UserList;
```

✅ Component me fetchUsers() dispatch kiya aur Redux se API data fetch kiya!

Redux Toolkit

What is Redux Toolkit?

- 👉 Redux Toolkit ek modern way hai Redux manage karne ka!
- 👉 Ye `createSlice()`, `configureStore()`, aur `createAsyncThunk()` provide karta hai jo Redux ko easy banata hai!
- 👉 Redux ka pura boilerplate (extra code) remove karne ke liye isko banaya gaya hai!

Real-Life Example:

Socho tum Swiggy 🍕 se online order kar rahe ho!

- Cart me item add karna (State Management)
- Item remove karna (Actions & Reducers)
- Cart ka total price calculate karna (Selector)
- Backend se menu fetch karna (Async API Call)

🔥 Agar tumhari Swiggy app me Redux Toolkit ka use ho, toh ye sab manage karna easy ho jata hai!

Installing Redux Toolkit in React

👉 Sabse pehle Redux Toolkit ko install karna padega!

npm install @reduxjs/toolkit react-redux

✅ Ab Redux Toolkit ka use kar sakte ho!

Redux Toolkit Ki Basic Architecture

Redux Toolkit 3 main cheezon se bana hota hai:

Store: Jisme data hota hai

Slices: Jo state, actions aur reducers ko ek jagah rakhta hai

Dispatch & Selector: Jo component ko Redux store se

connect karta hai

Creating a Simple Redux Store (Swiggy Cart Example)

👉 Ab ek real-world example banayenge jo Swiggy ka cart manage karega!

Step 1: Create Redux Store

```
// src/redux/store.js
```

```
import { configureStore } from "@reduxjs/toolkit";
```

```
import cartReducer from "./cartSlice";
```

```
const store = configureStore({
```

```
  reducer: {
```

```
    cart: cartReducer,
```

```
  },
```

```
});
```

```
export default store;
```


Step 2: Create Redux Slice for Cart

```
// src/redux/cartSlice.js
```

```
import { createSlice } from "@reduxjs/toolkit";
```

```
const cartSlice = createSlice({  
  name: "cart",  
  initialState: { items: [], totalAmount: 0 },  
  reducers: {  
    addItem: (state, action) => {  
      state.items.push(action.payload);  
      state.totalAmount += action.payload.price;  
    },  
    removeItem: (state, action) => {  
      const index = state.items.findIndex((item) => item.id ===  
action.payload);  
      if (index !== -1) {  
        state.totalAmount -= state.items[index].price;  
        state.items.splice(index, 1);  
      }  
    },  
    clearCart: (state) => {  
      state.items = [];  
      state.totalAmount = 0;  
    },  
  },  
});
```

```
export const { addItem, removeItem, clearCart } = cartSlice.actions;  
export default cartSlice.reducer;
```

✅ Yahan createSlice() ka use karke Redux slice banaya jo cart ke state aur actions handle karega!

✅ 3 actions hain – addItem, removeItem, aur clearCart jo cart manage karenge!

Step 3: Provide Redux Store to React App

```
// src/main.jsx
```

```
import React from "react";
```

```
import ReactDOM from "react-dom/client";
```

```
import { Provider } from "react-redux";
```

```
import store from "../redux/store";
```

```
import App from "../App";
```

```
ReactDOM.createRoot(document.getElementById("root")).render(  
  <Provider store={store}>  
    <App />  
  </Provider>  
);
```

✔ Provider ka use karke Redux store ko poore React app me available banaya!

Step 4: Using Redux State and Dispatching Actions

```
// src/App.jsx
```

```
import React from "react";
```

```
import { useSelector, useDispatch } from "react-redux";
```

```
import { addItem, removeItem, clearCart } from
```

```
"./redux/cartSlice";
```

```
function App() {
```

```
  const items = useSelector((state) => state.cart.items);
```

```
  const totalAmount = useSelector((state) =>  
state.cart.totalAmount);
```

```
  const dispatch = useDispatch();
```

```
  const sampleItem = { id: 1, name: "Pizza 🍕", price: 300 };
```



```
return (  
  <div>  
    <h1>🛒 Swiggy Cart</h1>  
    <button  
      onClick={() => dispatch(addItem(sampleItem))}>+ Add  
Pizza  
    </button>  
    <button  
      onClick={() => dispatch(removeItem(sampleItem.id))}>-  
Remove Pizza  
    </button>  
    <button  
      onClick={() =>  
        dispatch(clearCart())}>🗑️ Clear Cart  
    </button>  
    <h2>Cart Items:</h2>  
    <ul>  
      {items.map((item, index) => (  
        <li key={index}>{item.name} - ₹{item.price}</li>  
      ))}  
    </ul>  
    <h3>Total Amount: ₹{totalAmount}</h3>  
  </div>  
);  
}
```

export default App;

- ✅ useSelector() ka use karke Redux store se cart items aur total amount fetch kiya!
- ✅ useDispatch() ka use karke Redux ke actions ko call kiya!

Redux Toolkit with API Calls (createAsyncThunk)

👉 Agar tumhe kisi API se menu fetch karna ho, toh createAsyncThunk() ka use kar sakte ho!

Example: Fetching Menu from API

```
// src/redux/menuSlice.js
```

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
```

```
// API Call using createAsyncThunk
```

```
export const fetchMenu = createAsyncThunk("menu/fetchMenu", async () => {  
  const response = await fetch("https://fakestoreapi.com/products");  
  return response.json();  
});
```

```
const menuSlice = createSlice({  
  name: "menu",  
  initialState: { items: [], loading: false },  
  reducers: {},  
  extraReducers: (builder) => {  
    builder  
      .addCase(fetchMenu.pending, (state) => {  
        state.loading = true;  
      })  
      .addCase(fetchMenu.fulfilled, (state, action) => {  
        state.loading = false;  
        state.items = action.payload;  
      })  
      .addCase(fetchMenu.rejected, (state) => {  
        state.loading = false;  
      });  
  },  
});  
export default menuSlice.reducer;
```

✅ API call fetchMenu() se ho rahi hai aur Redux store me data store ho raha hai!

Using API Data in Component

```
// src/MenuList.jsx
```

```
import React, { useEffect } from "react";
```

```
import { useSelector, useDispatch } from "react-redux";
```

```
import { fetchMenu } from "../redux/menuSlice";
```

```
function MenuList() {
```

```
  const dispatch = useDispatch();
```

```
  const { items, loading } = useSelector((state) => state.menu);
```

```
  useEffect(() => {
```

```
    dispatch(fetchMenu());
```

```
  }, [dispatch]);
```

```
  return (
```

```
    <div>
```

```
      <h2>Menu List</h2>
```

```
      {
```

```
loading ?
```

```
<p>Loading...</p> : items.map((item) =>
```

```
<p key={item.id}>{item.title}</p>)
```

```
}
```

```
    </div>
```

```
  );
```

```
}
```

```
export default MenuList;
```

✅ API se fetched menu items Redux store me store ho rahe hain!

Lazy Loading

What is Lazy Loading?

- ☞ Lazy Loading ek technique hai jisme component ko tabhi load kiya jata hai jab user ko zaroorat ho!
- ☞ Ye React ke `React.lazy()` aur `Suspense` se hota hai!
- ☞ Isse app fast load hoti hai aur unnecessary components load nahi hote!

Real-Life Example:

Socho tum Amazon shopping app use kar rahe ho!

- Agar tum sirf "Electronics" section dekh rahe ho, toh "Fashion" section ke images aur products load karke kya fayda?
- Lazy Loading yahi karta hai – sirf wahi cheez load karta hai jo user dekhe!

How to Use Lazy Loading in React?

👉 React me lazy loading karne ke liye `React.lazy()` aur `Suspense` ka use hota hai!

Example: Basic Lazy Loading

```
import React, { lazy, Suspense } from "react";
```

```
// Lazy Loaded Component
```

```
const About = lazy(() => import("./About"));
```

```
function App() {
```

```
  return (
```

```
<div>
```

```
<h1>🏠 Home Page</h1>
```

```
<Suspense fallback={<h2>Loading...</h2>}>
```

```
  <About />
```

```
</Suspense>
```

```
</div>
```

```
);
```

```
}
```

```
export default App;
```

✅ Yahan About component ko lazy load kiya, jo tabhi load hoga jab zaroorat padegi!

✅ Suspense ka fallback tab dikhega jab tak About load nahi hota!

Lazy Loading with React Router

👉 Agar tumhe different pages lazy load karne hain, toh `React.lazy()` ko React Router ke saath use kar sakte ho!

Example: Lazy Loading Pages in React Router

```
import React, { lazy, Suspense } from "react";

import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";

const Home = lazy(() => import("./Home"));
const About = lazy(() => import("./About"));
const Contact = lazy(() => import("./Contact"));

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link> |
        <Link to="/about">About</Link> |
        <Link to="/contact">Contact</Link>
      </nav>

      <Suspense fallback={
        <h2>Loading Page...</h2>
      }>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </Suspense>
    </Router>
  );
}

export default App;
```

- ✅ Ab "Home", "About" aur "Contact" pages tabhi load honge jab user unpe jaayega! 🔄
- ✅ fallback tab dikhayega jab tak page load nahi ho raha!

Lazy Loading with Dynamic Imports (Code Splitting)

👉 Agar tumhe kisi ek specific feature ya component ko lazy load karna ho, toh `import()` ka use kar sakte ho!

Example: Dynamic Component Loading

```
import React, { useState, lazy, Suspense } from "react";

const HeavyComponent = lazy(() => import("./HeavyComponent"));

function App() {
  const [showComponent, setShowComponent] = useState(false);

  return (
    <div>
      <h1>🔥 Dynamic Lazy Loading</h1>
      <button
        onClick={() =>
          setShowComponent(true)}>Load Heavy Component
      </button>

      {showComponent && (
        <Suspense fallback={<h2>Loading Component...</h2>}>
          <HeavyComponent />
        </Suspense>
      )}
    </div>
  );
}

export default App;
```

✅ "HeavyComponent" tabhi load hoga jab user "Load Heavy Component" button dabayega!

✅ Isse unnecessary components load nahi honge aur app fast chalegi!

Optimizing Lazy Loading with React.memo()

👉 Agar tumhara component baar-baar re-render ho raha hai bina kisi change ke, toh React.memo() ka use karo!

Example: Preventing Unnecessary Renders

```
import React, { memo } from "react";

const OptimizedComponent = memo(function

OptimizedComponent({ text }) {

  console.log("Rendered: ", text);

  return <h2>{text}</h2>;

});

export default OptimizedComponent;
```

✅ React.memo() component ko unnecessary re-renders se bachata hai! 🔥

Lazy Loading Images in React

👉 Agar tumhari website me bohot saari images hain, toh react-lazy-load-image-component ka use karo!

📌 Install Lazy Load Image Package

npm **install react-lazy-load-image-component**

Example: Lazy Loading Images

```
import React from "react";

import { LazyLoadImage } from "react-lazy-load-image-component";

import "react-lazy-load-image-component/src/effects/blur.css";

function ImageComponent() {
  return (
    <LazyLoadImage
      src="https://source.unsplash.com/random/800x600" alt="Lazy
      Loaded" effect="blur" width="100%" height="auto"
    />
  );
}

export default ImageComponent;
```

✅ Images tabhi load honghi jab wo screen me dikhengi, jisse website fast chalegi!