Front-End Most frequently asked Interview Questions:

HTML Important Interview Topics:

Core HTML Topics

- Structure of an HTML document (including the use of <html>, <head>, and <body> tags).
- Difference between HTML and HTML5, including new elements and deprecated ones.
- Block vs. inline elements, and semantic HTML tags (e.g., <header>, <nav>, <section>, <article>, <aside>, <footer>).
- Common HTML tags and attributes, including how to use elements like <a>, <img>, <form>, <input>, <table>, , and .
- Void elements and self-closing tags (e.g., <img>,
, <hr>).
- The significance and usage of the <!DOCTYPE html> declaration.
- Difference between and <b>, and <i> tags.
- Purpose and use of HTML attributes such as id, class, and data-*.
- Semantic vs. non-semantic tags and the importance of semantic markup for accessibility and SEO.

HTML & Forms

- How to build and structure forms, use various input types, associate labels, and handle attributes like required, readonly, disabled, and placeholder.
- Difference between id and class attributes.
- How to submit forms and basic form validation.

Multimedia and Embedding

- Embedding images, videos, audio, and iframes.
- Use and purpose of the alt attribute for accessibility in images.
- Differences between <script>, <style>, and <noscript> tags.

HTML5 Features and APIs

- New HTML5 features such as <audio>, <video>, <canvas>, <section>, and <article> tags.
- Local Storage, Session Storage, and other Web APIs like Geolocation and Web Workers.
- Drag-and-drop API and making elements draggable.

Advanced and Practical Topics

- Character encoding and the use of <meta charset="UTF-8">.

- Accessibility best practices, including ARIA attributes.
- How to create responsive layouts using HTML and CSS; role of the viewport meta tag.
- Deprecated tags, and common browser compatibility issues.

Most Asked Sample Questions

Topic	Sample Question
HTML Structure	"What is the basic structure of an HTML page?"
Tags & Attributes	"What is the difference between id and class?"
Semantic HTML	"Why use semantic tags? Give examples."
HTML vs. HTML5	"What has changed in HTML5 compared to HTML4?"
Forms	"How do you create a form with validation?"
Multimedia	"How do you embed a video? What is the <iframe> tag used for?"
APIs & Storage	"What is LocalStorage? How do you use it in HTML5?"
Accessibility & SEO	"How do semantic elements influence SEO and accessibility?"
Block vs. Inline Elements	"What's the difference between block and inline elements?"
Character Encoding	"How do you specify the character set in an HTML document?"

This list covers the foundational, advanced, and practical topics as most frequently asked in HTML interviews by a variety of companies and is relevant for both freshers and experienced candidates.

Sample Question and Answers from HTML

Q1. What is the basic structure of an HTML page?

Answer:

Every HTML document has a standard structure that defines how the content is organized.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Web Page</title>
</head>
<body>
```

```
  <h1>Welcome to My Website</h1>
  <p>This is a sample HTML page.</p>
</body>
</html>
```

- <html> → Root element
- <head> → Metadata (title, styles, links, etc.)
- <body> → Visible page content
- <!DOCTYPE html> → Tells browser it's an HTML5 document

2. Difference between HTML and HTML5

Q2. What's the difference between HTML and HTML5?

Answer:

Feature	HTML	HTML5
Doctype	Long and complex	Simple <!DOCTYPE html>
Multimedia	Uses plugins (Flash)	Built-in <audio> and <video>
New Semantic Tags	Not available	<header>, <footer>, <section>, <article>, <nav>
Storage	Cookies only	LocalStorage & SessionStorage
Deprecated Tags	<font>, , <big>	Removed in HTML5

3. Block vs Inline Elements

Q3. What's the difference between block and inline elements?

Answer:

- **Block Elements:** Start on a new line and take full width (, <p>, <h1>).
- **Inline Elements:** Do not start on a new line and take only the required width (, <a>, <img>).

Example:

This is a block element
This is inlineThis link is inline

4. Common HTML Tags & Attributes

Q4. What are some common HTML tags and attributes?

Answer:

Tag	Description	Example
<a>	Link	Visit
<img>	Image	<img src="image.jpg" alt="Example Image">
<form>	Form container	<form action="/submit">
<input>	Input field	<input type="text" placeholder="Enter name">
<table>	Table	<table><tr><td>Data</td></tr></table>
	Generic block container	
	Inline container	Red

5. Void Elements & Self-Closing Tags

Q5. What are void elements in HTML?

Answer:
Void elements are elements that do **not have closing tags** and **cannot have content**.

Examples:

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

<hr>
<input type="text">

6. <!DOCTYPE html> Declaration

Q6. What is the purpose of <!DOCTYPE html>?

Answer:
It tells the browser that the document follows **HTML5 standards**, ensuring consistent rendering across browsers.

7. vs <b>, vs <i>

Q7. What is the difference between and <b>, and and <i>?

Answer:

Tag	Meaning	Function
<b>	Bold text (visual only)	No special meaning
	Important text	Adds semantic importance for SEO & screen readers
<i>	Italic text (visual)	No extra meaning
	Emphasized text	Semantic emphasis for assistive tech

Example:

```
<p><b>Bold</b> vs <strong>Important</strong></p>
<p><i>Italic</i> vs <em>Emphasized</em></p>
```

8. HTML Attributes: id, class, data-*

Q8. What's the difference between id and class?

Answer:

- id → Unique identifier (used once per page)
- class → Can be reused for styling or grouping elements

Example:

```
<div id="header"></div>
<div class="card"></div>
```

Q9. What are data-* attributes used for?

They store **custom data** in HTML elements that can be accessed via JavaScript.

Anuj

9. Semantic vs Non-semantic Tags

Q10. Why use semantic HTML?

Answer:
Semantic tags give **meaning** to content, helping browsers, search engines, and assistive devices.

Examples:

<header>Site Header</header>
<nav>Navigation Links</nav>
<article>Blog Article</article>
<footer>Footer Info</footer>

✅ Improves **SEO**, **Accessibility**, and **Readability**

10. HTML Forms

Q11. How do you create a form with validation?

Answer:

<form action="/submit" method="post">
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required placeholder="Enter your email">
  <button type="submit">Submit</button>
</form>

- required → Mandatory field
- readonly → Cannot edit

- disabled → Disabled input

11. Multimedia and Embedding

Q12. How do you embed videos, audio, and iframes?

Answer:

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

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

<iframe src="https://example.com" width="400" height="300"></iframe>
```

Q13. What is the purpose of the alt attribute in <img>?

It provides **text alternatives** for screen readers and shows text if the image fails to load — important for **accessibility**.

12. HTML5 Features & APIs

Q14. What are new HTML5 features?

Answer:

- **New tags:** <article>, <section>, <header>, <footer>, <nav>
- **APIs:** LocalStorage, Geolocation, Web Workers
- **Multimedia:** <audio>, <video>, <canvas>

Q15. What is LocalStorage?

LocalStorage allows storing data **in the browser** with **no expiration**.

Example:

```
localStorage.setItem("username", "Anuj");
console.log(localStorage.getItem("username"));
```

13. Advanced & Practical Topics

Q16. What is <meta charset="UTF-8"> used for?

It specifies **character encoding** so the browser correctly displays special characters like emojis or accents.

```
<meta charset="UTF-8">
```

Q17. What are accessibility best practices?

- Use semantic HTML
- Add alt text to images
- Use ARIA roles (aria-label, aria-hidden)
- Ensure proper color contrast

Example:

```
<button aria-label="Search">🔍</button>
```

Q18. How do you make a webpage responsive?

Use the **viewport meta tag** and **CSS media queries**.

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

Q19. What are deprecated tags in HTML5?

Deprecated Tag	Replacement
<font>	CSS font-family
	CSS text-align: center;
<big>	CSS font-size

20. Common Interview Rapid Questions

Topic	Question	Short Answer
HTML Structure	What is <!DOCTYPE html>?	Declares HTML5 document type
Tags	What is used for?	Container for grouping elements
Attributes	What is the use of data-*?	Store custom data
Forms	What's the purpose of action and method?	action defines URL; method defines HTTP type
Storage	Difference between LocalStorage and SessionStorage?	Local = persistent, Session = temporary
Block vs Inline	Give examples	Block: , Inline:
Semantic HTML	Example	<article>, <footer>, <header>
Accessibility	What is ARIA?	Attributes for assistive tech
HTML5 APIs	What is <canvas> used for?	Drawing graphics with JS
SEO	Why use semantic tags?	Helps search engines understand content

1. What is HTML?

Answer:
HTML (HyperText Markup Language) is the **standard language for creating web pages**. It defines the structure of content using tags.

```
<!DOCTYPE html>
<html>
 <body>
  <h1>Hello, HTML!</h1>
 </body>
</html>
```

2. What are HTML tags?

Answer:
Tags define elements in HTML.
They are written inside < > and usually come in pairs:

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

3. What is an attribute in HTML?

Answer:
Attributes provide **extra information** about elements.

```
<img src="logo.png" alt="Website logo">
```

- src and alt are attributes.

4. What is <!DOCTYPE html>?

Answer:
It tells the browser the document type and version — in HTML5, it's written as:

<!DOCTYPE html>

5. What is the structure of an HTML document?

```
<!DOCTYPE html>
<html>
 <head>
  <title>Title Here</title>
 </head>
 <body>
  <h1>Page Content</h1>
 </body>
</html>
```

6. What is the difference between HTML and HTML5?

Feature	HTML	HTML5
Doctype	Long and complex	Simple
Multimedia	Needs Flash	Has <audio> and <video>
Storage	Cookies only	LocalStorage, SessionStorage
New Tags	No	<section>, <article>, <nav>
Deprecated	<font>,	Removed

7. What are semantic tags?

Answer:
Semantic tags clearly define their purpose:

<header>, <footer>, <article>, <section>, <nav>

They improve SEO and accessibility.

8. What is the difference between block and inline elements?

Block	Inline
Takes full width	Takes only necessary width
Starts on a new line	Stays on same line
Example:	Example:

9. What is the difference between and ?

- → Only makes text **bold**
- → Adds **semantic importance**

Visual Bold
Important Bold

10. What is the difference between <i> and ?

- <i> → Italic style
- → Emphasized text (semantic)

11. What are void elements?

Answer:
Elements that **don’t have closing tags**.
Examples:
, <hr>, <img>, <input>

12. What are self-closing tags?

Same as void elements, written as:

13. What is the <meta charset="UTF-8"> tag for?

Answer:
It defines **character encoding** to support special characters.

<meta charset="UTF-8">

14. What is the use of the <title> tag?

Displays the title in the browser tab and is important for SEO.

15. What are HTML comments?

<!-- This is a comment -->

16. What is the difference between and ?

Block-level	Inline-level
Groups larger sections	Groups small text or inline elements

17. What are id and class attributes?

- id → unique identifier
- class → reusable styling or grouping

18. What is the <a> tag used for?

Creates hyperlinks.

Visit Site

19. What is the target attribute in <a>?

Open in new tab

20. What is the difference between relative and absolute URLs?

- **Relative:** href="about.html" → inside same site
- **Absolute:** href="https://example.com/about.html"

21. What is the alt attribute used for in <img>?

It provides **alternative text** for screen readers and SEO.

<img src="photo.jpg" alt="Profile picture">

22. How do you create a table in HTML?

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

23. What is the difference between <th> and <td>?

- <th> → Table Header (bold & centered)
- <td> → Table Data (normal cell)

24. What are HTML lists?

- **Ordered List:**
- **Unordered List:**

Apple
Banana

25. What is the <form> tag used for?

It collects user input.

```
<form action="/submit" method="POST">
  <input type="text" name="user">
  <button type="submit">Submit</button>
</form>
```

26. What are input types in HTML5?

text, password, email, number, date, range, color, checkbox, radio, etc.

27. What is the placeholder attribute?

Shows hint text in inputs.

```
<input type="text" placeholder="Enter name">
```

28. What is the difference between readonly and disabled?

- readonly: user cannot change but value submitted
- disabled: user cannot change & not submitted

29. What is the required attribute?

Ensures the field is not empty before submission.

```
<input type="email" required>
```

30. What are label and for attributes?

They connect a label to an input.

```
<label for="email">Email</label>
<input id="email" type="email">
```

31. What are semantic HTML5 tags?

<header>, <nav>, <main>, <section>, <article>, <aside>, <footer>

32. How do you embed a video in HTML5?

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

33. How do you embed audio?

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

34. What is the <iframe> tag used for?

Embeds another webpage.

```
<iframe src="https://example.com" width="400" height="300"></iframe>
```

35. What is the <canvas> tag used for?

Used with JavaScript to draw graphics or animations.

37. What is the <noscript> tag used for?

Displays fallback content if JavaScript is disabled.

<noscript>Please enable JavaScript</noscript>

38. What is character encoding?

Defines how characters are stored and displayed.
HTML5 uses UTF-8:

<meta charset="UTF-8">

39. What are ARIA attributes?

Help improve accessibility for screen readers.

<button aria-label="Search">🔍</button>

40. How do you make a page responsive?

Use:

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

And CSS media queries.

41. What is the difference between <script> and <style>?

- <script> → Contains JavaScript

- <style> → Contains CSS

42. What is the difference between inline, internal, and external CSS?

- **Inline:** style="color:red;"
- **Internal:** Inside <style> tag
- **External:** Linked using <link rel="stylesheet">

43. What are deprecated tags in HTML5?

Examples: <font>, , <big>, <marquee>

44. What are empty elements?

Same as void elements — no content or closing tag (
, <img>, <hr>)

45. What is HTML5 Web Storage?

It allows storing data in browser memory — more secure than cookies.

46. What is the difference between <header> and <head>?

- <head> → Metadata
- <header> → Visible top section of webpage

47. What is <main> used for?

Defines the main content of the webpage.

48. What is the use of <progress> and <meter> tags?

<progress value="60" max="100"></progress>
<meter value="0.8">80%</meter>

49. What is drag and drop in HTML5?

Allows elements to be draggable.

Drag me

50. How does HTML support accessibility and SEO?

- Use **semantic tags**
- Provide **alt text**
- Use **heading hierarchy**
- Use **ARIA attributes**

✅ **BONUS TIP FOR INTERVIEWS:**
Interviewers often ask you to:

- **Create a simple form with validation**
- **Explain semantic tags**
- **Embed an image/video**
- **Describe the role of <meta> and accessibility tags**

CSS Important Topics

Core CSS Topics

- CSS selectors: type, class, ID, attribute, pseudo-class, and pseudo-element selectors.
- The CSS box model: content, padding, border, margin, and how box-sizing affects layout.
- Ways to include CSS: inline, internal (embedded), and external stylesheets.
- Specificity and how CSS "cascades" with inheritance and overriding rules.
- Positioning: static, relative, absolute, fixed, and sticky.
- Display property: block, inline, inline-block, none, and their differences.
- Flexbox and CSS Grid for layout, including their main differences and use-cases.
- Responsive design: media queries, units (em, rem, %, vw, vh), viewport meta tag, and mobile-first design strategies.
- CSS3 features: transitions, transforms, animations, gradients, box-shadow, and border-radius.
- *CSS variables (custom properties) and preprocessors (Sass, LESS).*
- z-index property and stacking context.

Frequently Asked CSS Interview Questions

Topic	Sample Question
Selectors	"How do you select an element with a specific class in CSS?"
Box Model	"Can you explain the CSS box model and how box-sizing works?"
Positioning	"What's the difference between position absolute, relative, fixed, and sticky?"
Display	"What is the difference between block and inline elements?"
Flexbox/Grid	"When would you use Grid vs Flexbox?"

Responsive Design	"How do you make a website responsive using CSS?"
Specificity	"What is CSS specificity and how does it work?"
CSS3 Features	"What are CSS transitions and how are they used?"
Variable/Preprocessors	"What are CSS variables? What are preprocessors like SASS/LESS?"
Optimization	"How do you optimize CSS for large and/or fast websites?"

These topics are widely considered essential for all levels of CSS interviews, whether for freshers or experienced developers

Q1: What are the different types of CSS selectors?

Answer:
CSS selectors target HTML elements for styling.
Common types:

- **Type selector:** Targets elements by name

```
p { color: blue; }
```

- **Class selector:** Targets elements with a specific class

```
.btn { background-color: green; }
```

- **ID selector:** Targets an element with a specific id

```
#main { font-size: 20px; }
```

- **Attribute selector:**

```
input[type="text"] { border: 1px solid gray; }
```

- **Pseudo-class:** Targets elements in a specific state

a:hover { color: red; }

- **Pseudo-element:** Targets a part of an element

p::first-line { font-weight: bold; }

2. CSS Box Model

Q2: What is the box model in CSS?

Answer:
Each element is a box consisting of:

1. **Content** – text or image
2. **Padding** – space inside border
3. **Border** – around padding
4. **Margin** – space outside border

```
div {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
  margin: 10px;
}
```

Total width = 200 + 20 + 20 + 5 + 5 + 10 + 10 = 270px

3. Ways to Include CSS

Q3: What are the 3 ways to include CSS in HTML?

1. **Inline CSS**

<h1 style="color:red;">Hello</h1>

2. **Internal CSS**

<style> h1 { color: red; } </style>

3. **External CSS**

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

4. Specificity & Cascade

Q4: How does CSS specificity work?

Answer:
Specificity determines which rule applies when multiple rules target the same element.

Priority order:

1. Inline styles (highest)
2. IDs
3. Classes, pseudo-classes, attributes
4. Elements (lowest)

Example:

```
p { color: blue; }      /* lowest */
p.intro { color: red; } /* higher */
#main p { color: green; } /* highest */
```

5. Positioning

Q5: Explain different CSS position values.

Answer:

```
.static { position: static; }   /* default */
.relative { position: relative; top:10px; }
.absolute { position: absolute; top:0; left:0; }
.fixed { position: fixed; bottom:0; }
.sticky { position: sticky; top:0; }
```

6. Display Property

Q6: Difference between block, inline, inline-block, and none

Value	Behavior
block	Takes full width, new line
inline	Fits content, no new line
inline-block	Acts inline but supports width/height
none	Hides element

Example:

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

7. Flexbox

Q7: What is Flexbox and when do we use it?

Answer:
Flexbox helps align and distribute space among items in a container.

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

Used for one-dimensional layouts (rows or columns).

8. CSS Grid

Q8: What is CSS Grid?

Answer:
Grid is used for two-dimensional layouts (rows & columns).

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  gap: 10px;
}
```

9. Responsive Design

Q9: How to make a website responsive?

- Use **media queries**:

```
@media (max-width: 600px) {
  body { background-color: lightblue; }
}
```

- Use **relative units** like %, em, rem, vw, vh
- Use **viewport meta tag**:

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

10. CSS3 Features

Q10: What are some new features in CSS3?

- **Transitions:**

```
button { transition: background 0.3s; }
button:hover { background: blue; }
```

- **Transforms:**

```
div { transform: rotate(45deg); }
```

- **Animations:**

```
@keyframes move { from {left:0;} to {left:100px;} }
```

- **Gradients, box-shadow, border-radius**

11. CSS Variables

Q11: What are CSS custom properties (variables)?

```
:root {
  --main-color: blue;
}
h1 {
  color: var(--main-color);
}
```

12. z-index and Stacking Context

Q12: What is z-index in CSS?

Answer:
z-index controls the stacking order of elements.
Higher value → appears above.

```
div { position: absolute; z-index: 10; }
```

13. Without Media Queries

Q13: How to make layouts responsive without media queries?

Use **Flexbox** or **Grid** with flexible units (fr, minmax(), auto-fit).

```
.container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
}
```

14. Optimization Strategies

Q14: How to optimize CSS performance?

- Use **external CSS** files
- **Minify CSS**
- **Combine selectors**
- Remove **unused CSS**
- Use **shorthand** properties
- Follow **BEM naming convention**

15. Browser Compatibility

Q15: How do you handle CSS browser issues?

- Use **Autoprefixer**
- Test in multiple browsers
- Use **vendor prefixes** like -webkit-, -moz-

16. Accessibility

Q16: How can CSS improve accessibility?

- Maintain **color contrast**
- Use **focus styles** (:focus-visible)
- Don't rely solely on color for meaning
- Combine with **ARIA roles**

17. CSS Frameworks

Q17: What are CSS frameworks?

Frameworks provide pre-built CSS for fast UI development.
Examples:

- **Bootstrap**
- **Tailwind CSS**
- **Foundation**

18. Common Interview Questions Recap

Topic	Sample Question
Box Model	Explain CSS Box Model with example

Positioning	Difference between absolute and relative positioning
Flexbox	How to center an element using Flexbox
Grid	Difference between Grid and Flexbox
Responsive	What are media queries?
Animation	How to create a hover transition effect
Specificity	Which selector has higher priority?
Variables	How to use custom CSS variables
z-index	What is stacking context?
Frameworks	Difference between Bootstrap and Tailwind

1. What is CSS and why is it used?

Answer:
CSS (Cascading Style Sheets) is used to style HTML elements—control layout, colors, fonts, spacing, etc.

p { color: blue; font-size: 18px; }

2. What are the different ways to include CSS in HTML?

1. **Inline:** <p style="color:red">
2. **Internal:** <style>p{color:red;}</style>
3. **External:** <link rel="stylesheet" href="style.css">

3. What is the CSS Box Model?

It defines how elements are displayed as boxes:

- **Content → Padding → Border → Margin**

div { width:200px; padding:10px; border:5px solid; margin:10px; }

4. What is the difference between id and class selectors?

- id → Unique, used once.
- class → Reusable across multiple elements.

```
#header { color: blue; }
.title { color: green; }
```

5. What is the difference between relative and absolute positioning?

- relative: Moves relative to its original position.
- absolute: Moves relative to the nearest positioned ancestor.

```
.relative { position: relative; top:10px; }
.absolute { position: absolute; top:0; left:0; }
```

6. What are block, inline, and inline-block elements?

Type	Behavior
block	Starts on new line, full width
inline	Doesn't start new line
inline-block	Behaves inline but supports width/height

7. What are pseudo-classes in CSS?

Pseudo-classes define special states of elements.

```
a:hover { color: red; }
input:focus { border: 1px solid blue; }
```

8. What are pseudo-elements?

They style specific parts of elements.

```
p::first-line { color: red; }
p::before { content: "Note: "; }
```

9. What is CSS specificity?

Determines which rule takes precedence:

1. Inline > 2. ID > 3. Class > 4. Tag

Example:

```
p { color: blue; }
p.intro { color: red; } /* Higher specificity */
```

10. What is the difference between padding and margin?

- **Padding:** Space *inside* element border.
- **Margin:** Space *outside* element border.

11. What is the difference between visibility:hidden and display:none?

- visibility: hidden → element occupies space but invisible.
- display: none → element removed from layout.

12. What is the z-index property?

Controls stacking order of elements.

```
div { position: absolute; z-index: 10; }
```

13. How can you center a div using CSS?

```
div {
  width: 200px;
  margin: 0 auto; /* horizontally */
}
```

Or using Flexbox:

```
body {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

14. What is the difference between inline and block-level elements?

- Inline: , <a> – flow in text.
- Block: , <p> – start on new line.

15. What is the float property used for?

To align elements left or right.

```
img { float: right; margin: 10px; }
```

16. What is the clear property used for?

To prevent elements from wrapping around floated elements.

.clearfix { clear: both; }

17. What are CSS combinators?

Used to define relationships between selectors:

- div p → descendant
- div > p → direct child
- div + p → adjacent sibling
- div ~ p → general sibling

18. What is a CSS reset and why is it used?

To remove browser default styles:

* { margin: 0; padding: 0; box-sizing: border-box; }

19. What is box-sizing and its types?

Determines how total width/height is calculated.

box-sizing: content-box; /* default */
box-sizing: border-box; /* includes padding & border */

20. What are CSS units?

- **Absolute:** px, cm
- **Relative:** %, em, rem, vw, vh

Example:

p { font-size: 2em; width: 50%; }

21. What are transitions in CSS?

Used for smooth animations.

```
button {
  transition: background 0.3s;
}
button:hover {
  background: blue;
}
```

22. What are transforms in CSS3?

They modify element shape or position.

```
div { transform: rotate(45deg) scale(1.5); }
```

23. What is the difference between relative and fixed positioning?

- **Relative:** moves within normal flow.
- **Fixed:** stays in place during scrolling.

24. What is a media query?

Used for responsive design.

```
@media (max-width: 600px) {
  body { background: lightblue; }
}
```

25. What is the difference between em and rem units?

- em → relative to parent font size.
- rem → relative to root (html) font size.

26. What is CSS Grid and how is it different from Flexbox?

- **CSS Grid:** Two-dimensional layout (rows & columns).
- **Flexbox:** One-dimensional layout (row OR column).

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  gap: 10px;
}
```

27. What is a responsive layout without media queries?

Use **flexible units** and **Grid/Flexbox**:

```
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
}
```

28. What are CSS variables (custom properties)?

Reusable values for consistency.

```
:root {
  --main-color: #4CAF50;
}
button {
  background-color: var(--main-color);
```

}

29. What are CSS preprocessors?

Languages like **SASS, LESS** that extend CSS with:

- Variables
- Nesting
- Mixins

$color: red;
p { color: $color; }

30. What is a stacking context?

A hierarchy controlling which elements appear above others.
Triggered by:

- position with z-index
- opacity < 1
- transform, filter, flex, grid contexts

31. How to optimize CSS for performance?

- Minify CSS
- Remove unused styles
- Combine selectors
- Use external stylesheets
- Follow **BEM** or **CSS-in-JS**

32. How to handle cross-browser compatibility?

- Use **Autoprefixer** (-webkit-, -moz-)
- Test on multiple browsers
- Provide fallbacks for older browsers

33. What is the difference between inline, inline-block, and block?

Property	inline	inline-block	block
Starts new line	❌	❌	✅
Width/Height	❌	✅	✅
Example		<a>	

34. What is the difference between relative, absolute, fixed, and sticky?

Position	Description
relative	Moves relative to original position
absolute	Relative to nearest positioned ancestor
fixed	Fixed to viewport
sticky	Acts relative until scroll threshold, then sticks

35. How do you center elements with Flexbox?

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

36. What are pseudo-classes vs pseudo-elements?

Type	Example	Purpose
Pseudo-class	a:hover	Target element state
Pseudo-element	p::first-line	Target part of element

37. What is the difference between absolute and relative units?

- **Absolute units:** px, cm → fixed size
- **Relative units:** %, em, rem, vw, vh → scale with parent or viewport

38. How do you create CSS transitions and animations?

Transition Example:

```
div { transition: background 0.5s; }
div:hover { background: red; }
```

Animation Example:

```
@keyframes bounce {
  0% { transform: translateY(0); }
  50% { transform: translateY(-50px); }
  100% { transform: translateY(0); }
}
div { animation: bounce 1s infinite; }
```

39. What is the difference between em and rem?

- em: relative to **parent font-size**
- rem: relative to **root font-size**

40. What is object-fit in CSS?

Controls how images/videos fit their container.

```
img {
  width: 200px; height: 200px;
  object-fit: cover; /* or contain, fill */
}
```

41. How does CSS improve accessibility?

- Maintain **color contrast**
- Provide **focus indicators**
- Use ARIA roles & semantic tags
- Avoid using color as the only visual cue

42. What is the difference between :nth-child() and :nth-of-type()?

- :nth-child(n): selects element by position among all siblings
- :nth-of-type(n): selects element by position among siblings of same type

```
li:nth-child(2) { color: red; }
li:nth-of-type(2) { color: blue; }
```

43. What is the difference between inline, internal, and external CSS?

- **Inline:** <p style="color:red;"> – specific element
- **Internal:** <style> inside <head> – page-wide
- **External:** <link> to CSS file – reusable across pages

44. What are CSS frameworks?

Pre-built libraries for faster UI development.
Examples: Bootstrap, Tailwind CSS, Foundation, Bulma

45. How to create a responsive Grid layout?

```
.container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 20px;
}
```

46. What is the difference between absolute and fixed positioning?

- **Absolute:** moves relative to positioned parent
- **Fixed:** stays fixed on viewport even when scrolling

47. What is the difference between rem and % for font-size?

- rem → relative to root font size
- % → relative to parent element

48. What is the difference between relative and sticky positioning?

- **Relative:** moves relative to original position
- **Sticky:** scrolls normally until threshold, then sticks

49. What are best practices for CSS maintainability?

- Use **modular CSS** (BEM, SMACSS)
- Use **variables** for repeated values
- Minify for production
- Avoid !important unless necessary
- Use semantic class names

50. What are modern CSS layout techniques for responsiveness?

- **Flexbox** → one-dimensional layouts
- **CSS Grid** → two-dimensional layouts
- **Viewport units (vw, vh)**
- **Clamp()** for font scaling
- **Minmax()** for grid flexibility

✅ **Now you have all 50 CSS interview questions** with:

- Core fundamentals (selectors, box model, units, positioning)
- Advanced concepts (Grid, Flexbox, transitions, animations, variables)
- *Practical tips (optimization, responsiveness, accessibility, frameworks)*

JavaScript Important Interview Topics:

Core JavaScript Topics

- Data types (string, number, boolean, null, undefined, object, symbol, bigint).
- Variable declaration and scope (var, let, const).
- Hoisting, closures, and lexical scope.
- The "this" keyword, context, and function binding (call, apply, bind).
- Difference between == and ===.

- Arrow functions and traditional functions.
- Event handling and propagation (bubbling and capturing).
- Arrays and array methods (map, filter, reduce, forEach, etc.).
- Objects, prototypes, and inheritance.
- Asynchronous JavaScript (callbacks, promises, async/await).
- Error handling (try-catch, error objects).
- ES6+ features: destructuring, spread/rest operators, template literals, modules (import/export).
- *Common patterns: debouncing, throttling, polyfills.*

Advanced/Practical Topics

- DOM manipulation, query selectors, event listeners.
- Memory management and garbage collection.
- Closures for private variables and module patterns.
- Differences between synchronous and asynchronous code, the event loop, and *micro/macro tasks.*
- *Browser storage: localStorage, sessionStorage, cookies.*

Frequently Asked Interview Questions

Topic	Sample Question
Data Types	"What are the data types present in JavaScript?"
Hoisting	"Explain hoisting with examples in JavaScript."
Equality	"What is the difference between == and ===?"
Scope	"Explain lexical scope and closures."
this	"How does the 'this' keyword work in different contexts?"
Arrays	"Show how map, filter, and reduce can be used."
Promises	"What is a promise? Difference between promise and callback?"
ES6 Features	"Explain destructuring assignment in ES6."
Async/Await	"What is async/await and when would you use it?"
Error Handling	"How do you handle errors in JavaScript?"
Prototypes	"What is prototype inheritance?"
DOM	"How do you add or remove elements from the DOM?"

Mastering these areas prepares candidates for most JavaScript interview questions posed by top tech companies in recent panels.

Q1: What are the data types in JavaScript?
Answer: JavaScript has **7 primitive data types** and objects:

- **Primitive:** string, number, boolean, null, undefined, symbol, bigint
- **Non-primitive:** object (including arrays, functions, objects)

```
let str = "Hello";   // string
let num = 100;       // number
let bool = true;     // boolean
let n = null;        // null
let u;               // undefined
let sym = Symbol("id"); // symbol
let big = 9007199254740991n; // bigint
let obj = { name: "Alice" }; // object
```

2. Variable Declaration and Scope

Q2: Difference between var, let, and const?

Keyword	Scope	Hoisting	Reassignable
var	Function	Yes (undefined)	Yes
let	Block	Yes (Temporal Dead Zone)	Yes
const	Block	Yes (Temporal Dead Zone)	No

```
function test() {
  var x = 1; // function scoped
  let y = 2; // block scoped
  const z = 3; // block scoped, constant
}
```

3. Hoisting

Q3: What is hoisting in JavaScript?
Answer: Hoisting is moving variable and function declarations to the top of their scope before execution.

```
console.log(a); // undefined
var a = 10;

console.log(b); // ReferenceError
let b = 20;

// Function hoisting
sayHi();
function sayHi() { console.log("Hi"); }
```

4. Lexical Scope and Closures

Q4: Explain closures in JavaScript.

Answer: A closure is a function that retains access to its **outer scope** even after the outer function has executed.

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    console.log(count);
  }
}
const counter = outer();
counter(); // 1
counter(); // 2
```

5. The this Keyword and Function Binding

Q5: How does this work in different contexts?

Context	this value
Global	window (browser) / undefined (strict mode)
Object method	The object itself
Constructor function	New instance of object
Arrow function	Lexical this from enclosing scope

```
const obj = {
  name: "Alice",
  greet: function() { console.log(this.name); }
};
obj.greet(); // Alice

const greetArrow = () => console.log(this.name);
greetArrow(); // undefined in browser strict mode
```

Function Binding:

```
function hello() { console.log(this.name); }
const user = { name: "Bob" };
hello.call(user);  // Bob
hello.apply(user); // Bob
const bound = hello.bind(user);
bound();           // Bob
```

6. Equality Operators

Q6: Difference between == and ===?

- == → Checks **value only**, performs type coercion
- === → Checks **value + type**, strict equality

```
5 == '5'  // true
5 === '5' // false
```

7. Functions and Arrow Functions

Q7: Difference between traditional and arrow functions?

```
// Traditional function
function sum(a, b) { return a + b; }

// Arrow function
const sumArrow = (a, b) => a + b;

// Arrow functions do not have their own 'this'
```

8. Event Handling & Propagation

Q8: Explain event bubbling and capturing.

```
<div id="parent">
  <button id="child">Click</button>
</div>

document.getElementById('parent').addEventListener('click', () => console.log('Parent'));
document.getElementById('child').addEventListener('click', (e) => {
  console.log('Child');
  e.stopPropagation(); // stops bubbling
});
```

- **Bubbling:** Event goes from **target → parent → root**
- **Capturing:** Event goes from **root → parent → target**

9. Arrays and Array Methods

Q9: Examples of map, filter, reduce, forEach

```
const arr = [1,2,3,4];

// map → transform
arr.map(x => x*2); // [2,4,6,8]

// filter → filter based on condition
arr.filter(x => x%2===0); // [2,4]

// reduce → accumulate value
arr.reduce((sum, x) => sum + x, 0); // 10

// forEach → iterate without returning
arr.forEach(x => console.log(x));
```

10. Objects, Prototypes, and Inheritance

Q10: Explain prototype inheritance in JavaScript.

```
function Person(name) { this.name = name; }
Person.prototype.greet = function() { console.log(`Hello, ${this.name}`); }

const p1 = new Person("Alice");
p1.greet(); // Hello, Alice
```

- Objects inherit from Person.prototype.

11. Asynchronous JavaScript

Q11: Difference between callbacks, promises, and async/await

```
// Callback
function getData(callback) { callback("data"); }

// Promise
const promise = new Promise((resolve, reject) => resolve("data"));
promise.then(res => console.log(res));

// Async/Await
async function fetchData() {
  const res = await promise;
  console.log(res);
}
fetchData();
```

12. Error Handling

Q12: How to handle errors in JavaScript?

```
try {
  let x = y; // ReferenceError
} catch(err) {
  console.log(err.message);
} finally {
  console.log("Cleanup code");
}
```

13. ES6+ Features

Q13: Explain destructuring assignment

```
// Array destructuring
const [a,b] = [1,2];

// Object destructuring
const {name, age} = {name:"Alice", age:25};
```

```
// Spread operator
const arr2 = [...arr, 5,6];
```

Modules:

```
// export.js
export const pi = 3.14;

// import.js
import { pi } from './export.js';
```

15. DOM Manipulation

Q15: How to add/remove elements dynamically?

```
const div = document.createElement("div");
div.textContent = "Hello";
document.body.appendChild(div);

document.body.removeChild(div);
```

Query selectors:

```
document.querySelector("#id");
document.querySelectorAll(".class");
```

16. Memory Management

Q16: How does JavaScript handle memory?

- JavaScript has automatic garbage collection
- Unused objects are removed from memory

```
let a = {name:"Alice"};
a = null; // eligible for garbage collection
```

17. Synchronous vs Asynchronous

Q17: Explain event loop, microtask, and macrotask

```
console.log('1');
setTimeout(()=>console.log('2'),0);
Promise.resolve().then(()=>console.log('3'));
console.log('4');
// Output: 1,4,3,2
```

- Microtasks (Promises) run before macrotasks (setTimeout)

18. Browser Storage

Q18: Difference between localStorage, sessionStorage, and cookies

Storage	Lifetime	Size	Accessible in server?
localStorage	persists	~5MB	No
sessionStorage	until tab closes	~5MB	No
cookies	configured by expiration	4KB	Yes

```
localStorage.setItem("name","Alice");
sessionStorage.setItem("age","25");
```

This covers **all core, advanced, and practical JavaScript topics** with **examples**, perfectly aligned for interviews.

1. What are the data types in JavaScript?

Answer: Primitive: string, number, boolean, null, undefined, symbol, bigint
Non-primitive: object (arrays, functions, objects)

```
let str = "Hello";
let num = 100;
let bool = true;
let n = null;
let u;
let sym = Symbol("id");
let big = 9007199254740991n;
let obj = { name: "Alice" };
```

2. Difference between var, let, and const

Keyword	Scope	Hoisting	Reassignable
var	Function	Yes (undefined)	Yes
let	Block	TDZ (Temporal Dead Zone)	Yes
const	Block	TDZ	No

3. What is hoisting?

Variables and function declarations are moved to the top of their scope.

```
console.log(a); // undefined
var a = 10;

sayHi();
function sayHi(){ console.log("Hi"); } // works
```

4. What are closures?

```
function outer(){
  let count=0;
  return function inner(){
    count++;
    console.log(count);
  }
}
const counter = outer();
counter(); // 1
counter(); // 2
```

5. Explain lexical scope

A function can access variables **defined in its outer scope**, forming lexical scoping.

6. How does this work?

```
const obj = {
  name:"Alice",
  greet(){ console.log(this.name); }
};
obj.greet(); // Alice

const greetArrow = () => console.log(this.name);
greetArrow(); // undefined
```

7. Difference between == and ===

```
5 == '5'  // true (type coercion)
5 === '5' // false (strict equality)
```

8. Arrow vs traditional functions

- Arrow functions do **not have their own this**.

```
const sum = (a,b) => a+b;
function sumFunc(a,b){ return a+b; }
```

9. Event handling: bubbling vs capturing

```
document.getElementById('parent').addEventListener('click',()=>console.log('Parent'));
document.getElementById('child').addEventListener('click',e=>{
  console.log('Child');
  e.stopPropagation();
});
```

- **Bubbling:** target → parent → root
- **Capturing:** root → parent → target

10. Array methods: map, filter, reduce, forEach

```
const arr=[1,2,3,4];

arr.map(x=>x*2); // [2,4,6,8]
arr.filter(x=>x%2===0); // [2,4]
arr.reduce((sum,x)=>sum+x,0); // 10
```

```
arr.forEach(x=>console.log(x));
```

11. Objects and prototype inheritance

```
function Person(name){ this.name=name; }
Person.prototype.greet = function(){ console.log(`Hello, ${this.name}`); }
const p1 = new Person("Alice");
p1.greet(); // Hello, Alice
```

12. Difference between synchronous and asynchronous code

- Synchronous → executed sequentially
- Asynchronous → executed later, non-blocking

```
console.log('1');
setTimeout(()=>console.log('2'),0);
console.log('3');
// Output: 1,3,2
```

13. Promises vs callbacks

```
// Callback
function getData(cb){ cb("data"); }

// Promise
const promise = new Promise((res,rej)=>res("data"));
promise.then(console.log);
```

14. Async/Await

```
async function fetchData(){
  const res = await promise;
  console.log(res);
}
fetchData();
```

15. Error handling (try-catch)

```
try {
  let x = y; // ReferenceError
} catch(err){
  console.log(err.message);
} finally{
  console.log("Cleanup code");
}
```

16. ES6 destructuring

```
const [a,b] = [1,2];
const {name,age} = {name:"Alice", age:25};
```

17. Spread and rest operators

```
const arr2 = [...arr,5,6]; // spread
function sum(...args){ return args.reduce((a,b)=>a+b,0); } // rest
```

18. Template literals

```
const name = "Alice";
console.log(`Hello, ${name}`);
```

19. Modules (import/export)

```
// export.js
export const pi = 3.14;

// import.js
import { pi } from './export.js';
```

22. DOM manipulation: add/remove elements

```
const div=document.createElement("div");
div.textContent="Hello";
document.body.appendChild(div);
document.body.removeChild(div);
```

23. Query selectors

```
document.querySelector("#id");
document.querySelectorAll(".class");
```

24. Memory management

```
let obj={name:"Alice"};
obj=null; // eligible for garbage collection
```

25. Event loop, microtask vs macrotask

```
console.log('1');
setTimeout(()=>console.log('2'),0);
Promise.resolve().then(()=>console.log('3'));
console.log('4');
// Output: 1,4,3,2
```

26. localStorage, sessionStorage, cookies

Storage	Lifetime	Size	Server accessible?
localStorage	Persist	~5MB	No
sessionStorage	Tab session	~5MB	No
cookies	Configurable	4KB	Yes

```
localStorage.setItem("name","Alice");
```

27. Difference between null and undefined

```
let a;
console.log(a); // undefined
let b=null;
console.log(b); // null
```

28. Difference between call, apply, bind

```
function greet(msg){ console.log(`${msg}, ${this.name}`); }
const user={name:"Alice"};

greet.call(user,"Hi");  // Hi, Alice
greet.apply(user,["Hello"]); // Hello, Alice
const bound = greet.bind(user,"Hey");
bound(); // Hey, Alice
```

29. Difference between function declaration and expression

```
// Declaration → hoisted
function foo(){}

// Expression → not hoisted
const bar = function(){};
```

30. IIFE (Immediately Invoked Function Expression)

```
(function(){
  console.log("IIFE executed");
})();
```

31. Closures for private variables

```
function counter(){
  let count=0;
```

```
  return { increment:()=>++count, get:()=>count }
}
const c = counter();
c.increment(); // 1
c.get();       // 1
```

33. Difference between synchronous and asynchronous loops

- forEach → synchronous
- setTimeout inside loops → asynchronous

34. typeof vs instanceof

```
typeof [] // "object"
[] instanceof Array // true
```

35. Shallow vs deep copy

```
// Shallow copy
const copy = {...obj};
// Deep copy
const deep = JSON.parse(JSON.stringify(obj));
```

39. Difference between var and function scope

- var → function-scoped
- let/const → block-scoped
- Functions → hoisted

40. ES6 Classes and Inheritance

```
class Person {
  constructor(name){ this.name=name; }
  greet(){ console.log(`Hello ${this.name}`); }
}
class Student extends Person {
  study(){ console.log(`${this.name} is studying`); }
}
```

41. Symbols in JavaScript

Unique and immutable identifiers

```
const sym1 = Symbol("id");
const sym2 = Symbol("id");
console.log(sym1===sym2); // false
```

42. BigInt

```
const big = 9007199254740991n;
```

43. setTimeout vs setInterval

```
setTimeout(()=>console.log("after 1s"),1000);
setInterval(()=>console.log("every 1s"),1000);
```

44. Difference between map and forEach

- map → returns a new array

- forEach → no return

45. Difference between filter and find

arr.filter(x=>x>2); // returns array of matches
arr.find(x=>x>2); // returns first match

46. Difference between splice and slice

arr.slice(1,3); // returns new array
arr.splice(1,2); // removes elements from original array

49. Memory leaks in JS

Caused by:

- Forgotten timers
- Detached DOM nodes
- Global variables

50. Best practices for JS performance

- Avoid global variables
- Use let/const
- Minify JS
- Debounce/throttle events
- Lazy load resources

Important Interview topics on React

Core React Topics

- What is React, its key features, and why it is used (component-based, virtual DOM, JSX, unidirectional data flow).
- Understanding JSX, how it's syntactic sugar for JavaScript, and how it compiles to React.createElement calls.
- Components: Types (Functional and Class components), and the concept of reusable UI components.
- State and Props: How to manage component data and pass data from parent to child.
- Lifecycle methods in Class components (componentDidMount, componentDidUpdate, componentWillUnmount) and their roles.
- Introduction and use of React Hooks like useState, useEffect, useReducer, and useContext in functional components.
- Virtual DOM and how React uses it to efficiently update the UI.
- Event handling in React and JSX event syntax.
- Unidirectional data flow and prop drilling challenges.
- React Router basics for Single Page Application (SPA) navigation.
- Server-Side Rendering (SSR) vs Client-Side Rendering (CSR) and frameworks like Next.js.
- React Strict Mode and its use in identifying potential issues.
- React Portals for rendering outside the main DOM hierarchy (useful in modals and tooltips).

State Management

- Redux: Core concepts (store, actions, reducers) and how it differs from Flux.
- Context API vs Redux: When to use each for state management and avoiding prop drilling.

Sample Frequently Asked Interview Questions

Topic	Sample Question
React Basics	"What is React and why is it used?"
JSX	"What is JSX and how does it differ from HTML?"
Components	"What are the differences between functional and class components?"
Props & State	"How do props differ from state in React?"

Lifecycle Methods	"Explain the different lifecycle methods in React."
Hooks	"What are React Hooks? Explain useState and useEffect."
Virtual DOM	"What is the virtual DOM, and why is it important?"
State Management	"What is Redux and how does it compare to Context API?"
SSR vs CSR	"What are server-side rendering and client-side rendering?"
React Portals	"What are React Portals and when would you use them?"
Event Handling	"How do you handle events in React components?"

Mastering these topics provides strong preparation for React interviews at various experience levels.

What is React? Why is it used?

Q: What is React and why do developers prefer it?
A: React is a **JavaScript library for building user interfaces**, developed by Facebook.
It allows creating **reusable UI components**, uses a **virtual DOM** for performance, and follows a **unidirectional data flow** for predictable state management.

Key Features:

- Component-based architecture
- Virtual DOM
- JSX syntax
- Unidirectional data flow
- High performance

Example:

```
function App() {
  return <h1>Hello, React!</h1>;
}
```

```
export default App;
```

2️⃣ JSX (JavaScript XML)

Q: What is JSX and how does it work?
A: JSX is a syntax extension that lets you write HTML-like code inside JavaScript.
It compiles down to React.createElement() calls.

Example:

```
// JSX
const element = <h1>Hello JSX!</h1>;

// Compiles to
const element = React.createElement('h1', null, 'Hello JSX!');
```

3️⃣ Components in React

Q: What are the types of components in React?
A:

1. **Functional Components** – use functions and hooks.
2. **Class Components** – use ES6 classes and lifecycle methods.

Example:

```
// Functional
function Welcome() {
  return <h2>Hello!</h2>;
}

// Class
```

```
class WelcomeClass extends React.Component {
  render() {
    return <h2>Hello from class!</h2>;
  }
}
```

4 State and Props

Q: Difference between state and props?
A:

- **State**: Managed within a component (mutable).
- **Props**: Passed from parent to child (immutable).

Example:

```
function Child({ name }) {
  return <p>Hello {name}</p>;
}

function Parent() {
  const [user, setUser] = useState('Anuj');
  return <Child name={user} />;
}
```

5 Lifecycle Methods (Class Components)

Q: What are the main lifecycle methods?
A:

- componentDidMount() → runs after component mounts.
- componentDidUpdate() → runs after state/props change.

- componentWillUnmount() → cleanup before unmounting.

Example:

```
class Timer extends React.Component {
 componentDidMount() {
  console.log("Mounted");
 }
 componentDidUpdate() {
  console.log("Updated");
 }
 componentWillUnmount() {
  console.log("Unmounted");
 }
 render() {
  return <p>Lifecycle Example</p>;
 }
}
```

6 React Hooks

Q: What are hooks?
A: Hooks let you use state and lifecycle features in **functional components**.

Common Hooks:

- useState() – for state management
- useEffect() – side effects (API calls, timers)
- useReducer() – complex state logic
- useContext() – global data access

Example:

```
import { useState, useEffect } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  useEffect(() => console.log('Updated:', count), [count]);
  return <button onClick={() => setCount(count + 1)}>Count:
{count}</button>;
}
```

7️⃣ Virtual DOM

Q: How does the Virtual DOM improve performance?
A: React keeps a **virtual copy of the DOM** in memory.
When state changes, it compares (diffing) old vs new Virtual DOM and updates only changed elements in the real DOM (reconciliation).

8️⃣ Event Handling

Q: How is event handling done in React?
A: Use camelCase event names and pass a function reference.

Example:

```
function Clicker() {
  const handleClick = () => alert("Clicked!");
  return <button onClick={handleClick}>Click me</button>;
}
```

9️⃣ Unidirectional Data Flow

Q: What does unidirectional data flow mean?
A: Data flows from **parent → child** via props.
This makes debugging easier but can lead to **prop drilling** (passing data through many layers).

Solution: Context API or Redux.

🔟 React Router (SPA Navigation)

Q: What is React Router?
A: A library for navigation in **Single Page Applications (SPAs)** without reloading the page.

Example:

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

function App() {
  return (
    <BrowserRouter>
      <Link to="/about">About</Link>
      <Routes>
        <Route path="/about" element={<h2>About Page</h2>} />
      </Routes>
    </BrowserRouter>
  );
}
```

1️⃣1️⃣ SSR vs CSR

Q: Difference between SSR and CSR?
A:

- **SSR (Server-Side Rendering):** HTML rendered on the server → faster initial load, SEO-friendly. (e.g., Next.js)
- **CSR (Client-Side Rendering):** Browser renders after JS loads → slower first load but more dynamic.

1️⃣2️⃣ React Strict Mode

Q: What is Strict Mode?
A: A development tool to highlight potential issues like unsafe lifecycles or side effects.

Example:

```
<React.StrictMode>
  <App />
</React.StrictMode>
```

1️⃣3️⃣ React Portals

Q: What are Portals?
A: They allow rendering elements **outside the main DOM hierarchy**, e.g., modals, tooltips.

Example:

```
ReactDOM.createPortal(
  <div className="modal">Hello Modal</div>,
  document.getElementById('modal-root')
```

);

⚙️ State Management

1️⃣4️⃣ Redux

Q: What is Redux?
A: Redux is a predictable **state management library** for JS apps.
It uses a **store**, **actions**, and **reducers**.

Flow:
Action → Reducer → Store → UI update

Example:

```
// action
const increment = () => ({ type: 'INCREMENT' });

// reducer
function counter(state = 0, action) {
  if (action.type === 'INCREMENT') return state + 1;
  return state;
}
```

1️⃣5️⃣ Context API vs Redux

Q: Difference between Context API and Redux?
A:

Feature	Context API	Redux

Complexity	Simple	More structured
Setup	Minimal	Requires configuration
Use Case	Small to medium apps	Large apps
State	Local/global	Centralized global

Example (Context API):

```
const UserContext = React.createContext();

function App() {
 return (
  <UserContext.Provider value="Anuj">
   <Profile />
  </UserContext.Provider>
 );
}

function Profile() {
 const name = React.useContext(UserContext);
 return <h3>Hello {name}</h3>;
}
```

Quick Interview Recap

Topic	Common Question	Key Point
React	What is it?	Library for building UIs
JSX	Why use it?	Looks like HTML, compiles to JS
State/Props	Difference?	State = internal, Props = external
Hooks	What are they?	Functions to use state/lifecycle in functional components
Virtual DOM	Purpose?	Efficient UI updates

Router	SPA navigation	Uses Routes and Links
Redux	Core idea	Predictable state container
Context vs Redux	When to use?	Context for small, Redux for large apps

1. What is React?

Answer: React is a JavaScript library used to build dynamic and reusable user interfaces.

Example:

```
function App() {
  return <h1>Hello React!</h1>;
}
```

2. What are the key features of React?

- Component-based architecture
- Virtual DOM
- Unidirectional data flow
- JSX syntax
- Reusable UI components

3. What is JSX?

Answer: JSX (JavaScript XML) allows writing HTML inside JavaScript.

```
const heading = <h1>Hello JSX!</h1>;
```

4. How does JSX work behind the scenes?

It is **transpiled** to React.createElement() calls by Babel.

React.createElement('h1', null, 'Hello JSX!');

5. What are Components in React?

Components are reusable building blocks.

- Functional Components
- Class Components

```
function Welcome() {
  return <h2>Hello!</h2>;
}
```

6. Difference between Functional and Class Components

Functional	Class
Uses functions	Uses ES6 classes
Uses hooks	Uses lifecycle methods

7. What are Props?

Props are inputs passed to components.

```
function Greet({ name }) {
  return <h2>Hello {name}</h2>;
```

}

8. What is State in React?

State stores dynamic data inside a component.

const [count, setCount] = useState(0);

9. Difference between Props and State

Props	State
Immutable	Mutable
Passed from parent	Managed inside component

10. What are React Hooks?

Functions that let you use state/lifecycle inside functional components.
Examples: useState, useEffect, useContext, useReducer

11. What is useState Hook?

Manages local state.

const [count, setCount] = useState(0);

12. What is useEffect Hook?

Used for side effects like API calls, timers.

```
useEffect(() => {
  console.log("Count changed");
}, [count]);
```

13. What is Virtual DOM?

A lightweight copy of the real DOM. React compares old vs new Virtual DOM and updates only changed parts (diffing + reconciliation).

14. Explain React component lifecycle (Class Components)

- componentDidMount()
- componentDidUpdate()
- componentWillUnmount()

16. What is a Controlled Component?

Form inputs controlled by React state.

```
<input value={name} onChange={e => setName(e.target.value)} />
```

17. What is an Uncontrolled Component?

Form data handled by the DOM itself.

<input type="text" ref={inputRef} />

18. What is lifting state up?

Moving state to a common ancestor to share data between child components.

19. What is React.Fragment?

Used to group multiple elements without extra DOM nodes.

```
<>
  <h1>Title</h1>
  <p>Paragraph</p>
</>
```

20. What are Keys in React?

Unique identifiers for list items.

{items.map(item => <li key={item.id}>{item.name})}

21. What is React Router?

Used for client-side navigation.

<Route path="/about" element={<About />} />

22. What is SPA (Single Page Application)?

An app that loads one HTML file and dynamically updates UI using JS.

23. What is Redux?

A predictable state management library.
Flow: Action → Reducer → Store → UI

const reducer = (state=0, action) => action.type==='INC' ? state+1 : state;

24. What are Actions in Redux?

Plain JS objects that describe what to do.

{ type: 'INCREMENT' }

25. What are Reducers in Redux?

Pure functions that update state based on actions.

function counter(state=0, action){ ... }

26. What is a Redux Store?

Holds the entire app state in one place.

const store = createStore(reducer);

29. What is Prop Drilling?

Passing props through multiple layers unnecessarily.

31. What is useCallback Hook?

Memoizes a function to prevent re-creation.

const handleClick = useCallback(() => setCount(c=>c+1), []);

35. What is React Lazy Loading?

Load components only when needed.

const LazyComp = React.lazy(() => import('./MyComp'));

36. What is Suspense in React?

Used with lazy loading to show a fallback UI.

```
<Suspense fallback={<p>Loading...</p>}>
  <LazyComp />
</Suspense>
```

37. What are Portals in React?

Render components outside main DOM hierarchy.

ReactDOM.createPortal(<Modal />, document.getElementById('root'));

38. What is React.StrictMode?

Highlights potential issues in development.

<React.StrictMode><App /></React.StrictMode>

39. What is Error Boundary?

Catches runtime errors in child components.

componentDidCatch(error, info) { ... }

40. What is Server-Side Rendering (SSR)?

HTML is rendered on the server before being sent to the browser. (Used in **Next.js**)

⚡ 41. Difference between SSR and CSR

SSR	CSR
Faster first load	Slower first load
Better SEO	Needs JS to render

43. What is Refs Forwarding?

Passing refs through a component to child DOM elements.

const Input = React.forwardRef((props, ref) => <input ref={ref} {...props}/>);

44. What is useReducer Hook?

Alternative to useState for complex state logic.

const [state, dispatch] = useReducer(reducer, initialState);

45. What is useContext Hook?

Accesses context data without prop drilling.

const user = useContext(UserContext);

46. What is PureComponent?

Prevents re-rendering if props/state haven't changed.

🧠 47. What is the difference between React and Angular/Vue?

React	Angular	Vue

Library	Framework	Framework
JSX	Templates	Templates
Virtual DOM	Real DOM	Virtual DOM

48. How to handle forms in React?

Controlled inputs using state.

<input value={name} onChange={e=>setName(e.target.value)} />

⚙️ 49. What are Synthetic Events?

React's wrapper around browser events for cross-browser compatibility.

Bonus Interview Tip:
Many recruiters ask to **build a small app (counter, todo list)** — practice writing:

- State updates with hooks
- Passing props between components
- API fetch with useEffect
- Navigation using react-router-dom

Important Interview questions on Redux

Core Redux Concepts

- What is Redux and why is it used? It's a predictable state container for JavaScript applications, often used with React to handle complex state management.
- Single Source of Truth: The entire app state is stored in one object (the store).
- State is Read-Only: State can only be changed by dispatching actions, never mutated directly.
- Changes are Made with Pure Functions: Reducers are pure functions that take the previous state and an action, returning a new state.
- Unidirectional Data Flow: Data in Redux flows in a single direction, making it easier to trace and debug.

Redux Architecture and Flow

- Actions: Plain objects that describe "what happened"; must have a 'type' property.
- Action Creators: Functions that construct and return action objects.
- Reducers: Pure functions that take the state and an action, and return a new state without mutating the original state.
- Store: The single global object holding the state of the application, created with createStore.
- Dispatch: The method for sending actions to the store.
- Selectors: Functions that extract and compute derived data from the Redux store, helping optimize performance.

What is Redux and why is it used?

Answer:
Redux is a **predictable state container** for JavaScript applications.
It helps manage **global state** — especially when multiple components need to share or update the same data.

✅ **Why use Redux?**

- Makes state predictable
- Centralized store (Single Source of Truth)
- Easy to debug (time-travel debugging)
- Consistent data flow

Example:

```
import { createStore } from 'redux';

// Reducer
function counter(state = 0, action) {
  if (action.type === 'INCREMENT') return state + 1;
  return state;
}

// Store
const store = createStore(counter);

// Dispatch action
store.dispatch({ type: 'INCREMENT' });

console.log(store.getState()); // Output: 1
```

Single Source of Truth

Answer:
In Redux, the **entire app's state** is stored in **one object** inside a single **store**.
This makes debugging and sharing data across components easier.

Example:

```
const store = {
  user: { name: "Anuj", loggedIn: true },
  theme: "dark",
  cart: [{ id: 1, name: "Book" }]
};
```

Everything (user info, theme, cart, etc.) is in **one central store**, not scattered in components.

State is Read-Only

Answer:
You **cannot directly modify** the state.
You must **dispatch an action** to describe what should change.
This ensures all state updates are **explicit and trackable**.

Example (❌ Wrong way):

```
store.state.count = 5; // ❌ Never mutate directly
```

✅ **Correct way:**

```
store.dispatch({ type: 'INCREMENT' });
```

Changes are Made with Pure Functions

Answer:
Reducers are pure functions that:

- Take **previous state** and **action**
- Return **new state**
- Never mutate the original state

Example:

```
function counter(state = { count: 0 }, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    default:
      return state;
  }
}
```

👉 Predictable and side-effect free.

Unidirectional Data Flow

Answer:
Redux follows a **one-way data flow**:

Flow:
View (UI) → Dispatch(Action) → Reducer → New State → Re-render View

This makes apps easier to **trace**, **debug**, and **maintain**.

Diagram (Conceptually):

User clicks → Dispatch Action → Reducer updates state → Store → UI updates

Redux Architecture & Flow — Step-by-Step

Actions

Answer:
Actions are **plain JavaScript objects** describing *what happened*.
Each must have a **type** property (string constant).

Example:

```
const INCREMENT = "INCREMENT";
const incrementAction = { type: INCREMENT };
```

Action Creators

Answer:
Action creators are **functions** that return action objects — making code cleaner and reusable.

Example:

```
function increment() {
  return { type: 'INCREMENT' };
}

// Usage
store.dispatch(increment());
```

Reducers

Answer:
Reducers are **pure functions** that define **how state changes** based on actions.
They should **not mutate** the existing state.

Example:

```
function todoReducer(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return [...state, action.payload]; // return new array
    default:
      return state;
  }
}
```

Store

Answer:
The **store** holds the global application state and provides methods like getState(), dispatch(), and subscribe().

Example:

```
import { createStore } from 'redux';
const store = createStore(todoReducer);
```

Dispatch

Answer:
dispatch() sends actions to the store, which triggers the reducer.

Example:

```
store.dispatch({ type: 'ADD_TODO', payload: 'Learn Redux' });
```

Selectors

Answer:
Selectors are **functions** that extract or compute derived data from the store.
They help optimize performance by avoiding unnecessary computations.

Example:

```
const selectCompletedTodos = (state) =>
  state.todos.filter(todo => todo.completed);

const completed = selectCompletedTodos(store.getState());
```

Complete Example — Simple Redux Counter

```
import { createStore } from 'redux';

// 1. Action types
const INCREMENT = "INCREMENT";
const DECREMENT = "DECREMENT";

// 2. Action creators
const increment = () => ({ type: INCREMENT });
const decrement = () => ({ type: DECREMENT });

// 3. Reducer
function counter(state = { count: 0 }, action) {
  switch (action.type) {
    case INCREMENT: return { count: state.count + 1 };
    case DECREMENT: return { count: state.count - 1 };
    default: return state;
  }
}

// 4. Store
const store = createStore(counter);

// 5. Subscribe (for UI re-render)
store.subscribe(() => console.log("State:", store.getState()));

// 6. Dispatch actions
store.dispatch(increment());
store.dispatch(increment());
store.dispatch(decrement());
```

✅ Output:

State: { count: 1 }
State: { count: 2 }
State: { count: 1 }

Quick Interview Recap

Concept	Definition	Example
Redux	Predictable state container	createStore(reducer)
Store	Holds entire state	store.getState()
Action	What happened	{ type: 'ADD_TODO' }
Action Creator	Function returning action	addTodo('Learn')
Reducer	Pure function for updates	(state, action) => newState
Dispatch	Sends action to store	store.dispatch(action)
Selector	Extracts/derives data	selectTodos(state)