Git-GitHub Cheatsheet

Basic Setup

- **git config --global user.name "Your Name"**

Set your Git username.

- **git config --global user.email "your.email@example.com"**

Set your Git email.

- **git config --list**

List all Git configurations.

Initializing and Cloning

- **git init**

Initialize a new Git repository in your project.

- **git clone <repo-url>**

Clone an existing repository.

Working with Changes

- **git add <file>**

Stage a specific file for commit.

- **git add .**

Stage all changes in the current directory.

- **git commit -m "Commit message"**

Commit changes with a message.

- **git commit -am "Message"**

Add and commit tracked files in one step.

- **git commit --amend**

Edit the last commit message or add changes to it.

Handling Merge Conflicts

- **git diff**

Compare working directory changes.

- **git diff <branch1> <branch2>**

Compare two branches.

Resolve conflicts: Open the files,
fix conflicts, then add and commit.

Status & Logs

- **git status**

Show the current status of changes in the working directory.

- **git log**

View commit history.

- **git log --oneline**

Show concise commit history.

Branching & Merging

- **git branch <branch-name>**

Create a new branch.

- **git checkout <branch-name>**

Switch to a specific branch.

- **git checkout -b <branch-name>**

Create and switch to a new branch.

- **git merge <branch-name>**

Merge specified branch into the current branch.

- **git rebase <branch-name>**

Reapply commits on top of another base.

- **git rebase -i HEAD~<n>**

Interactive rebase to edit commit history, rearrange commits, modify commit messages, or squash the last n commits

- **git branch -d <branch-name>**

Delete a local branch (use -D to force delete).

Spoorti Shetty

Undoing Changes

- **git reset <file>**

Unstage a file.

- **git reset --soft HEAD~1**

Undo last commit but keep changes staged.

- **git reset --mixed HEAD~1**

Undo last commit, keep changes in the working directory (unstaged).

- **git reset --hard HEAD~1**

Completely remove the last commit.

- **git revert <commit-id>**

Create a new commit that undoes the specified commit.

Stashing Changes

- **git stash**

Temporarily save changes.

- **git stash list**

View stashed changes.

- **git stash pop**

Reapply stashed changes and remove them from the stash list.

- **git stash apply**

Reapply stashed changes without removing them.

- **git stash clear**

Remove all stashed entries.

Collaborating & Pull Requests

- **git branch -a**

List all branches, including remote.

- **git push origin :<branch-name>**

Delete a remote branch.

Creating a Pull Request: Go to your GitHub repository, select your branch, and click "New Pull Request."

Remote Repositories

- **git remote add origin <url>**

Link your local repository to a remote one.

- **git remote -v**

List the remote repository URLs.

- **git remote set-url origin <new-url>**

Update the remote URL for the repository.

- **git remote rename <old-name> <new-name>**

Rename a remote.

- **git push -u origin <branch-name>**

Push changes to the remote repository.

- **git pull origin <branch-name>**

Pull changes from the remote branch.

- **git fetch**

Download updates from the remote without merging.

- **git fetch <remote>**

Fetch updates from a specific remote.

Advanced Operations

- **git cherry-pick <commit-id>**

Apply a specific commit from another branch.

- **git cherry-pick <start-commit-id>^..<end-commit-id>**

Cherry-pick a range of commits.

- **git tag <tag-name>**

Add a tag to a commit.

- **git tag -d <tag-name>**

Remove a local tag.

- **git reflog**

View history of all changes (even uncommitted).

- **git reflog show <branch-name>**

Show reflog for a specific branch.

- **git show <commit-id>**

Show detailed info for a specific commit.

- **git bisect start**

Start bisecting to locate a bug.

Spoorti Shetty

Reviewing Changes

- **git show <file>**
Display changes made to a specific file.

- **git diff <commit-id1> <commit-id2>**
Compare changes between two commits.

Help Command

- **git help <command>**
Get detailed help for a specific command.

GitHub Commands (Optional with GitHub CLI)

- **gh repo create**
Create a new GitHub repo from the command line.

- **gh repo clone <repo-url>**
Clone a GitHub repository.

- **gh pr create**
Create a pull request from the command line.

- **gh pr list**
List open pull requests in the repository.

- **gh issue create**
Create a GitHub issue from the command line.

GitHub API (using curl)

- **curl -H "Authorization: token YOUR_TOKEN" https://api.github.com/repos/USERNAME/REPO_NAME/issues**
List issues in a repository.

Submodules & Worktrees

- **git submodule add <repo-url> <path>**
Add a submodule.

- **git submodule init**
Initialize submodules.

- **git submodule update**
Update submodules.

- **git worktree add <path> <branch>**
Create a new working tree for a branch.

Repository Management and Information

- **git shortlog -s -n**
Summarize commits by author.

- **git describe --tags**
Get a readable name for a commit.

- **git blame <file>**
Show who last modified each line of a file.

- **git grep "search-term"**
Search for a term in the repository.

- **git revert <commit-id1>..<commit-id2>**
Revert a range of commits.

- **git archive --format=zip HEAD -o latest.zip**
Archive the latest commit as a ZIP file.

- **git fsck**
Check the object database for integrity.

Cleaning Up

- **git clean -f**
Remove untracked files.

- **git clean -fd**
Remove untracked files and directories.

- **git gc --prune=now**
Clean up unnecessary files and optimize the local repository.

Spoorti Shetty

Best Practices and Common Workflows

- **Commit Often:** Make frequent commits with descriptive messages to maintain a clear project history.
- **Branch for Features:** Create a new branch for each feature or bug fix to keep changes organized and separate from the main codebase.
- **Use Meaningful Commit Messages:** Write clear and concise commit messages that explain the purpose of the changes.
- **Pull Regularly:** Regularly pull changes from the remote repository to stay updated with the latest changes and minimize merge conflicts.
- **Resolve Conflicts Promptly:** Address merge conflicts as soon as they arise to avoid complicating the integration process.
- **Review Pull Requests Thoroughly:** Ensure thorough review of pull requests to maintain code quality and facilitate knowledge sharing.
- **Tag Releases:** Use tags to mark important milestones or releases in the project for easy reference in the future.
- **Keep Your Branches Clean:** Delete branches that are no longer needed after merging them into the main branch to keep the repository organized.
- **Use Git Hooks for Automation:** Utilize Git hooks to automate tasks, like running tests before committing (pre-commit) or checking commit message formats. Hooks can help ensure code quality and consistency.
- **Squash Commits Before Merging:** Squash commits to combine related work into a single commit before merging, especially for feature branches. This keeps the project history clean and manageable.
- **Avoid Large Commits:** Try to keep commits small and focused on a single change or fix. This makes it easier to understand the history and isolate issues if something goes wrong.
- **Create Descriptive Branch Names:** Use branch naming conventions that describe the purpose, such as feature/login-form or fix/user-authentication-bug. This improves readability and collaboration.
- **Keep the Main Branch Deployable:** Always ensure that the main or production branch is stable and deployable. This allows the project to be released or updated at any time.

📚HTML Notes

follow @techie_programmer for more notes

🌟 What is HTML?

HTML (HyperText Markup Language) is the standard markup language for creating web pages. It describes the structure and content of web documents using elements and tags.

🏗️ Basic Structure

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.
0">
  <title>Page Title</title>
</head>
<body>
  <!-- Content goes here -->
</body>
</html>
```

🔍 Key Components:

- `<!DOCTYPE html>` - Declares HTML5 document type
- `<html>` - Root element of the page
- `<head>` - Contains metadata (not visible on page)
- `<body>` - Contains visible content

📝 HTML Elements and Tags

🏷️ Basic Syntax

- **Opening tag:** `<tagname>`

- **Closing tag:** `</tagname>`
- **Self-closing tag:** `<tagname />`
- **Element with content:** `<tagname>content</tagname>`

🎯 Attributes

```
<tagname attribute="value">content</tagname>
<img src="image.jpg" alt="Description" width="300">
```

📖 Text Elements

🔤 Headings

```
<h1>Main Heading</h1>     <!-- Largest -->
<h2>Sub Heading</h2>
<h3>Section Heading</h3>
<h4>Subsection</h4>
<h5>Minor Heading</h5>
<h6>Smallest Heading</h6>  <!-- Smallest -->
```

📄 Text Formatting

```
<p>Paragraph text</p>
<br>                  <!-- Line break -->
<hr>                  <!-- Horizontal rule -->

<!-- Text styling -->
<strong>Bold text</strong>
<b>Bold text (visual only)</b>
<em>Emphasized text</em>
<i>Italic text</i>
<u>Underlined text</u>
<mark>Highlighted text</mark>
<del>Deleted text</del>
<ins>Inserted text</ins>
<sub>Subscript</sub>
```

```
<sup>Superscript</sup>
<small>Small text</small>
<code>Code snippet</code>
<kbd>Keyboard input</kbd>
<var>Variable</var>
<samp>Sample output</samp>
```

📋 Preformatted Text

```
<pre>
  Preformatted text
  preserves   spaces
  and line breaks
</pre>

<blockquote cite="source">
  This is a quote from another source
</blockquote>
```

🔗 Links and Navigation

🌐 Anchor Links

```
<!-- External link -->
<a href="https://www.example.com">Visit Example</a>

<!-- Internal link -->
<a href="page.html">Go to Page</a>

<!-- Email link -->
<a href="mailto:someone@example.com">Send Email</a>

<!-- Phone link -->
<a href="tel:+1234567890">Call Us</a>

<!-- Link to section -->
<a href="#section-id">Go to Section</a>
```

```
<!-- Download link -->
<a href="file.pdf" download>Download PDF</a>

<!-- Open in new tab -->
<a href="https://example.com" target="_blank">Open in New Tab</a>
```

🖼️ Images and Media

📸 Images

```
<img src="image.jpg" alt="Description" width="300" height="200">

<!-- Responsive image -->
<img src="image.jpg" alt="Description" style="max-width: 100%; height: a
uto;">

<!-- Image with different sources -->
<picture>
  <source media="(min-width: 800px)" srcset="large.jpg">
  <source media="(min-width: 400px)" srcset="medium.jpg">
  <img src="small.jpg" alt="Description">
</picture>
```

🎵 Audio and Video

```
<!-- Audio -->
<audio controls>
  <source src="audio.mp3" type="audio/mpeg">
  <source src="audio.ogg" type="audio/ogg">
  Your browser does not support audio.
</audio>

<!-- Video -->
<video controls width="400">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
```

```
    Your browser does not support video.
</video>
```

📊 Lists

📝 Unordered Lists

```
<ul>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
</ul>
```

🔢 Ordered Lists

```
<ol>
    <li>First step</li>
    <li>Second step</li>
    <li>Third step</li>
</ol>

<!-- Different numbering styles -->
<ol type="A">  <!-- A, B, C -->
<ol type="a">  <!-- a, b, c -->
<ol type="I">  <!-- I, II, III -->
<ol type="i">  <!-- i, ii, iii -->
<ol start="5"> <!-- Start from 5 -->
```

📖 Definition Lists

```
<dl>
    <dt>HTML</dt>
    <dd>HyperText Markup Language</dd>
    <dt>CSS</dt>
    <dd>Cascading Style Sheets</dd>
</dl>
```

📋 Tables

🗂️ Basic Table Structure

```
<table>
  <thead>
    <tr>
      <th>Header 1</th>
      <th>Header 2</th>
      <th>Header 3</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Data 1</td>
      <td>Data 2</td>
      <td>Data 3</td>
    </tr>
    <tr>
      <td>Data 4</td>
      <td>Data 5</td>
      <td>Data 6</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td>Footer 1</td>
      <td>Footer 2</td>
      <td>Footer 3</td>
    </tr>
  </tfoot>
</table>
```

🔗 Table Attributes

```
<!-- Spanning cells -->
<td colspan="2">Spans 2 columns</td>
<td rowspan="3">Spans 3 rows</td>
```

```
<!-- Table caption -->
<table>
  <caption>Table Title</caption>
  <!-- table content -->
</table>
```

📝 Forms

📩 Form Structure

```
<form action="/submit" method="POST">
  <fieldset>
    <legend>Personal Information</legend>

    <!-- Text input -->
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>

    <!-- Email input -->
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>

    <!-- Password input -->
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>

    <!-- Number input -->
    <label for="age">Age:</label>
    <input type="number" id="age" name="age" min="1" max="120">

    <!-- Date input -->
    <label for="birthdate">Birthdate:</label>
    <input type="date" id="birthdate" name="birthdate">

    <!-- Radio buttons -->
    <fieldset>
      <legend>Gender:</legend>
```

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

        <input type="radio" id="female" name="gender" value="female">
        <label for="female">Female</label>
    </fieldset>

    <!-- Checkboxes -->
    <input type="checkbox" id="newsletter" name="newsletter" value="y
es">
    <label for="newsletter">Subscribe to newsletter</label>

    <!-- Dropdown select -->
    <label for="country">Country:</label>
    <select id="country" name="country">
        <option value="">Select a country</option>
        <option value="us">United States</option>
        <option value="uk">United Kingdom</option>
        <option value="ca">Canada</option>
    </select>

    <!-- Textarea -->
    <label for="comments">Comments:</label>
    <textarea id="comments" name="comments" rows="4" cols="50"></t
extarea>

    <!-- File upload -->
    <label for="file">Upload file:</label>
    <input type="file" id="file" name="file" accept=".pdf,.doc,.docx">

    <!-- Submit button -->
    <input type="submit" value="Submit">
    <button type="submit">Submit Form</button>
    <button type="reset">Reset Form</button>
  </fieldset>
</form>
```

🔧 Input Types

```
<input type="text">       <!-- Text input -->
<input type="email">      <!-- Email validation -->
<input type="password">   <!-- Hidden text -->
<input type="number">     <!-- Numeric input -->
<input type="tel">        <!-- Telephone -->
<input type="url">        <!-- URL validation -->
<input type="search">     <!-- Search box -->
<input type="date">       <!-- Date picker -->
<input type="time">       <!-- Time picker -->
<input type="datetime-local"> <!-- Date and time -->
<input type="month">      <!-- Month picker -->
<input type="week">       <!-- Week picker -->
<input type="color">      <!-- Color picker -->
<input type="range">      <!-- Slider -->
<input type="file">       <!-- File upload -->
<input type="hidden">     <!-- Hidden field -->
<input type="checkbox">   <!-- Checkbox -->
<input type="radio">      <!-- Radio button -->
<input type="submit">     <!-- Submit button -->
<input type="reset">      <!-- Reset button -->
<input type="button">     <!-- Generic button -->
```

🎨 Semantic HTML5 Elements

🏗️ Layout Elements

```
<header>
  <nav>
    <ul>
      <li><a href="#home">Home</a></li>
      <li><a href="#about">About</a></li>
      <li><a href="#contact">Contact</a></li>
    </ul>
  </nav>
</header>

<main>
```

```
    <article>
      <header>
        <h1>Article Title</h1>
        <time datetime="2024-01-01">January 1, 2024</time>
      </header>

      <section>
        <h2>Section Heading</h2>
        <p>Section content...</p>
      </section>

      <aside>
        <h3>Related Links</h3>
        <ul>
          <li><a href="#">Link 1</a></li>
          <li><a href="#">Link 2</a></li>
        </ul>
      </aside>
    </article>
</main>

<footer>
    <p>&copy; 2024 Your Website. All rights reserved.</p>
</footer>
```

📄 Content Elements

```
<figure>
    <img src="image.jpg" alt="Description">
    <figcaption>Image caption</figcaption>
</figure>

<details>
    <summary>Click to expand</summary>
    <p>Hidden content that can be toggled</p>
</details>

<progress value="70" max="100">70%</progress>
```

```
<meter value="6" min="0" max="10">6 out of 10</meter>

<address>
  Contact: <a href="mailto:email@example.com">email@example.com</a
>
</address>
```

🏷️ Meta Tags and Head Elements

📊 Essential Meta Tags

```
<head>
  <!-- Character encoding -->
  <meta charset="UTF-8">

  <!-- Viewport for responsive design -->
  <meta name="viewport" content="width=device-width, initial-scale=1.
0">

  <!-- Page description -->
  <meta name="description" content="Page description for search engine
s">

  <!-- Keywords -->
  <meta name="keywords" content="html, web development, tutorial">

  <!-- Author -->
  <meta name="author" content="Your Name">

  <!-- Refresh page -->
  <meta http-equiv="refresh" content="30">

  <!-- CSS link -->
  <link rel="stylesheet" href="styles.css">

  <!-- Favicon -->
  <link rel="icon" type="image/x-icon" href="favicon.ico">
```

```
    <!-- External fonts -->
    <link rel="preconnect" href="https://fonts.googleapis.com">

    <!-- JavaScript -->
    <script src="script.js"></script>
</head>
```

🎯 HTML Entities

🔤 Common Entities

```
&lt;      <!-- < (less than) -->
&gt;      <!-- > (greater than) -->
&amp;     <!-- & (ampersand) -->
&quot;    <!-- " (quotation mark) -->
&apos;    <!-- ' (apostrophe) -->
     <!-- Non-breaking space -->
&copy;    <!-- © (copyright) -->
&reg;     <!-- ® (registered) -->
&trade;   <!-- ™ (trademark) -->
&euro;    <!-- € (euro) -->
&pound;   <!-- £ (pound) -->
&yen;     <!-- ¥ (yen) -->
&sect;    <!-- § (section) -->
&para;    <!-- ¶ (paragraph) -->
```

🔧 Best Practices

✅ Do's

- Always use proper DOCTYPE declaration
- Use semantic HTML5 elements
- Include alt attributes for images
- Use proper heading hierarchy (h1 → h2 → h3)
- Validate your HTML code

- Use lowercase for element names and attributes
- Quote attribute values
- Close all tags properly
- Use meaningful class and ID names
- Optimize for accessibility

❌ Don'ts

- Don't use deprecated elements (`<font>` , `<center>` , etc.)
- Don't skip heading levels
- Don't use tables for layout
- Don't use inline styles (use CSS instead)
- Don't forget to escape special characters
- Don't use non-semantic elements when semantic ones exist
- Don't ignore validation errors

🌐 HTML5 APIs and Features

📍 New Input Attributes

```
<input type="text" placeholder="Enter your name" required>
<input type="email" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$">
<input type="number" min="1" max="100" step="5">
<input type="text" autocomplete="name">
<input type="text" autofocus>
<input type="text" readonly>
<input type="text" disabled>
```

🎯 Data Attributes

```
<div data-user-id="123" data-role="admin">User Info</div>
<button data-action="delete" data-confirm="Are you sure?">Delete</butto
n>
```

```
<template id="my-template">
  <style>
    /* Scoped styles */
  </style>
  <div>Template content</div>
</template>
```

🌐 Microdata

```
<div itemscope itemtype="http://schema.org/Person">
  <h1 itemprop="name">John Doe</h1>
  <p itemprop="jobTitle">Software Developer</p>
  <a itemprop="url" href="https://johndoe.com">Website</a>
</div>
```

📚 Quick Reference

🏷️ Essential Tags Checklist

- [ ] `<!DOCTYPE html>`
- [ ] `<html lang="en">`
- [ ] `<head>` with meta tags
- [ ] `<title>`
- [ ] `<body>`
- [ ] Semantic elements (`<header>`, `<main>`, `<footer>`)
- [ ] Proper heading structure
- [ ] Alt text for images
- [ ] Form labels and validation
- [ ] Valid and accessible markup

🔗 Useful Resources

- **W3C HTML Validator**: https://validator.w3.org/

- **MDN HTML Reference**: https://developer.mozilla.org/en-US/docs/Web/HTML
- **HTML5 Specification**: https://html.spec.whatwg.org/
- **Accessibility Guidelines**: https://www.w3.org/WAI/WCAG21/quickref/

Happy coding! 🎉 Remember to always write semantic, accessible, and valid HTML!

#_ Important [JavaScript Built-in Methods] {CheatSheet}

1. Array Methods

- forEach(): `array.forEach(element => console.log(element));`
- map(): `const squares = array.map(x => x * x);`
- filter(): `const evens = array.filter(x => x % 2 === 0);`
- reduce(): `const sum = array.reduce((acc, curr) => acc + curr, 0);`
- find(): `const found = array.find(x => x > 10);`
- indexOf(): `const index = array.indexOf(item);`
- push(): `array.push(item);`
- pop(): `const lastItem = array.pop();`
- shift(): `const firstItem = array.shift();`
- unshift(): `array.unshift(item);`
- splice(): `array.splice(startIndex, deleteCount, item);`
- slice(): `const sliced = array.slice(startIndex, endIndex);`
- concat(): `const mergedArray = array1.concat(array2);`
- join(): `const str = array.join(', ');`
- includes(): `const hasItem = array.includes(item);`
- some(): `const hasNegative = array.some(x => x < 0);`
- every(): `const allPositive = array.every(x => x > 0);`
- sort(): `array.sort((a, b) => a - b);`
- reverse(): `array.reverse();`

2. String Methods

- charAt(): `const char = str.charAt(index);`
- concat(): `const combinedStr = str1.concat(str2);`
- includes(): `const hasSubstring = str.includes(substring);`
- indexOf(): `const index = str.indexOf(substring);`
- lastIndexOf(): `const lastIndex = str.lastIndexOf(substring);`
- match(): `const matches = str.match(regex);`

By: Devendra Singh Rajput

-
-
- repeat(): `const repeatedStr = str.repeat(times);`
- replace(): `const replacedStr = str.replace(regex, newSubstr);`
- search(): `const index = str.search(regex);`
- slice(): `const slicedStr = str.slice(startIndex, endIndex);`

  split(): `const tokens = str.split(delimiter);`

    substring(): `const substring = str.substring(startIndex, endIndex);`
- toLowerCase(): `const lowerStr = str.toLowerCase();`
- toUpperCase(): `const upperStr = str.toUpperCase();`
- trim(): `const trimmedStr = str.trim();`

3. Object Methods

- Object.keys(): `const keys = Object.keys(obj);`
- Object.values(): `const values = Object.values(obj);`
- Object.entries(): `const entries = Object.entries(obj);`
- Object.assign(): `const newObj = Object.assign({}, obj1, obj2);`
- Object.freeze(): `const frozenObj = Object.freeze(obj);`
- Object.isFrozen(): `const isFrozen = Object.isFrozen(obj);`
- Object.seal(): `const sealedObj = Object.seal(obj);`
- Object.isSealed(): `const isSealed = Object.isSealed(obj);`
- Object.create(): `const newObj = Object.create(prototypeObj);`
- Object.hasOwnProperty(): `const hasProp = obj.hasOwnProperty(prop);`

4. Number Methods

- Number.parseInt(): `const intVal = Number.parseInt(string, radix);`
- Number.parseFloat(): `const floatVal = Number.parseFloat(string);`
- Number.isNaN(): `const isNaN = Number.isNaN(value);`
- Number.isFinite(): `const isFinite = Number.isFinite(value);`
- Number.isInteger(): `const isInteger = Number.isInteger(value);`
- Number.toFixed(): `const fixedStr = num.toFixed(digits);`

-
-

5. Math Functions

- Math.abs(): `const absolute = Math.abs(num);`
- Math.ceil(): `const roundedUp = Math.ceil(num);`
- Math.floor(): `const roundedDown = Math.floor(num);`
- Math.round(): `const rounded = Math.round(num);`
- Math.max(): `const max = Math.max(num1, num2, ...);`
- Math.min(): `const min = Math.min(num1, num2, ...);`
- Math.pow(): `const power = Math.pow(base, exponent);`

  Math.sqrt(): `const sqrt = Math.sqrt(num);`

  Math.random(): `const randomNum = Math.random();`

6. Global Functions

- parseInt(): `const intVal = parseInt(string, radix);`
- parseFloat(): `const floatVal = parseFloat(string);`
- isNaN(): `const isNaN = isNaN(value);`
- isFinite(): `const isFinite = isFinite(value);`

7. Date Methods

- Date.now(): `const now = Date.now();`
- Date.parse(): `const timestamp = Date.parse(dateString);`
- new Date(): `const date = new Date(year, month, day, ...);`

8. Regular Expression Methods

- RegExp.test(): `const matchesPattern = regex.test(string);`
- RegExp.exec(): `const match = regex.exec(string);`

9. JSON Methods

- JSON.stringify(): `const jsonString = JSON.stringify(obj);`
- JSON.parse(): `const obj = JSON.parse(jsonString);`

By: Devendra Singh Rajput

-
-

10. Web APIs

- setTimeout(): `const timeoutId = setTimeout(() => { /* code */ }, delay);`
- clearTimeout(): `clearTimeout(timeoutId);`
- setInterval(): `const intervalId = setInterval(() => { /* code */ }, delay);`
- clearInterval(): `clearInterval(intervalId);`

11. Promises and Asynchronous Programming

- Promise.then(): `promise.then(value => { /* code */ });`
  Promise.catch(): `promise.catch(error => { /* code */ });`
  Promise.finally(): `promise.finally(() => { /* code */ });`
- Promise.all(): `Promise.all([promise1, promise2]).then(values => { /* code */ });`
- Promise.race(): `Promise.race([promise1, promise2]).then(value => { /* code */ });`
- async function: `async function fetchData() { const data = await fetch(url); }`

12. Console Methods

- console.log(): `console.log('message', obj);`
- console.warn(): `console.warn('warning message');` ● console.error(): `console.error('error message');`
- console.info(): `console.info('info message');`
- console.clear(): `console.clear();`

13. Miscellaneous Methods

- alert(): `alert('Alert message');`
- confirm(): `const confirmed = confirm('Are you sure?');`

-
-
- prompt(): `const userInput = prompt('Enter your name:', 'Default Name');`
- encodeURIComponent(): `const encoded = encodeURIComponent(uriComponent);`
- decodeURIComponent(): `const decoded = decodeURIComponent(encodedUriComponent);`

14. DOM Manipulation (Web-Specific)

- document.getElementById(): `const element = document.getElementById('id');`
- document.querySelector(): `const element = document.querySelector('.class');`
- document.querySelectorAll(): `const elements = document.querySelectorAll('div');`
- document.createElement(): `const div = document.createElement('div');`
  element.appendChild(): `parentElement.appendChild(childElement);`
  element.innerHTML: `element.innerHTML = '<span>New content</span>';`
- element.addEventListener(): `element.addEventListener('click', eventHandler);`
- element.style: `element.style.color = 'blue';`
- element.setAttribute(): `element.setAttribute('data-custom', 'value');`
- element.removeAttribute(): `element.removeAttribute('data-custom');`

15. Event Handling

- element.onclick: `element.onclick = function() { /* code */ };`
- element.addEventListener(): `element.addEventListener('click', eventHandler);`
- element.removeEventListener(): `element.removeEventListener('click', eventHandler);`
- event.preventDefault(): `element.addEventListener('click', event => event.preventDefault());`

-
-

16. BOM (Browser Object Model) Methods

- window.open(): `const newWindow = window.open(url);`
- window.close(): `window.close();`
- window.moveTo(): `window.moveTo(x, y);`
- window.resizeTo(): `window.resizeTo(width, height);`

17. Web Storage API

- localStorage.setItem(): `localStorage.setItem('key', 'value');`
- localStorage.getItem(): `const value = localStorage.getItem('key');`
- sessionStorage.setItem(): `sessionStorage.setItem('key', 'value');`
- sessionStorage.getItem(): `const value = sessionStorage.getItem('key');`

18. Fetch API and XMLHttpRequest

- fetch(): `fetch(url).then(response => response.json());`
- XMLHttpRequest(): `const xhr = new XMLHttpRequest(); xhr.open('GET', url); xhr.send();`

19. History API

- history.pushState(): `history.pushState(stateObj, 'title', 'page2.html');`
- history.replaceState(): `history.replaceState(stateObj, 'title', 'page3.html');`
- history.back(): `history.back();`

20. WebSockets

- WebSocket: `const socket = new WebSocket('ws://example.com');`

21. File and FileReader API

- FileReader.readAsDataURL(): `const reader = new FileReader(); reader.readAsDataURL(file);`
- FileReader.onload: `reader.onload = function() { console.log(reader.result); };`

22. Geolocation API

- navigator.geolocation.getCurrentPosition(): `navigator.geolocation.getCurrentPosition(position => console.log(position));`
- navigator.geolocation.watchPosition(): `const watchID = navigator.geolocation.watchPosition(position => console.log(position));`

23. Performance API

- performance.now(): `const start = performance.now();`

24. Request Animation Frame

- requestAnimationFrame(): `requestAnimationFrame(timestamp => { /* animations */ });`

25. Internationalization

- Intl.DateTimeFormat(): `const formatter = new Intl.DateTimeFormat('en-US'); console.log(formatter.format(date));`
- Intl.NumberFormat(): `const numberFormatter = new Intl.NumberFormat('en-US'); console.log(numberFormatter.format(number));`

26. Canvas API

- getContext(): `const ctx = canvas.getContext('2d');`

27. Drag and Drop API

- ondragstart: `element.ondragstart = function(event) { /* code */ };`
- ondrop: `dropArea.ondrop = function(event) { /* code */ };`

28. Mutation Observer API

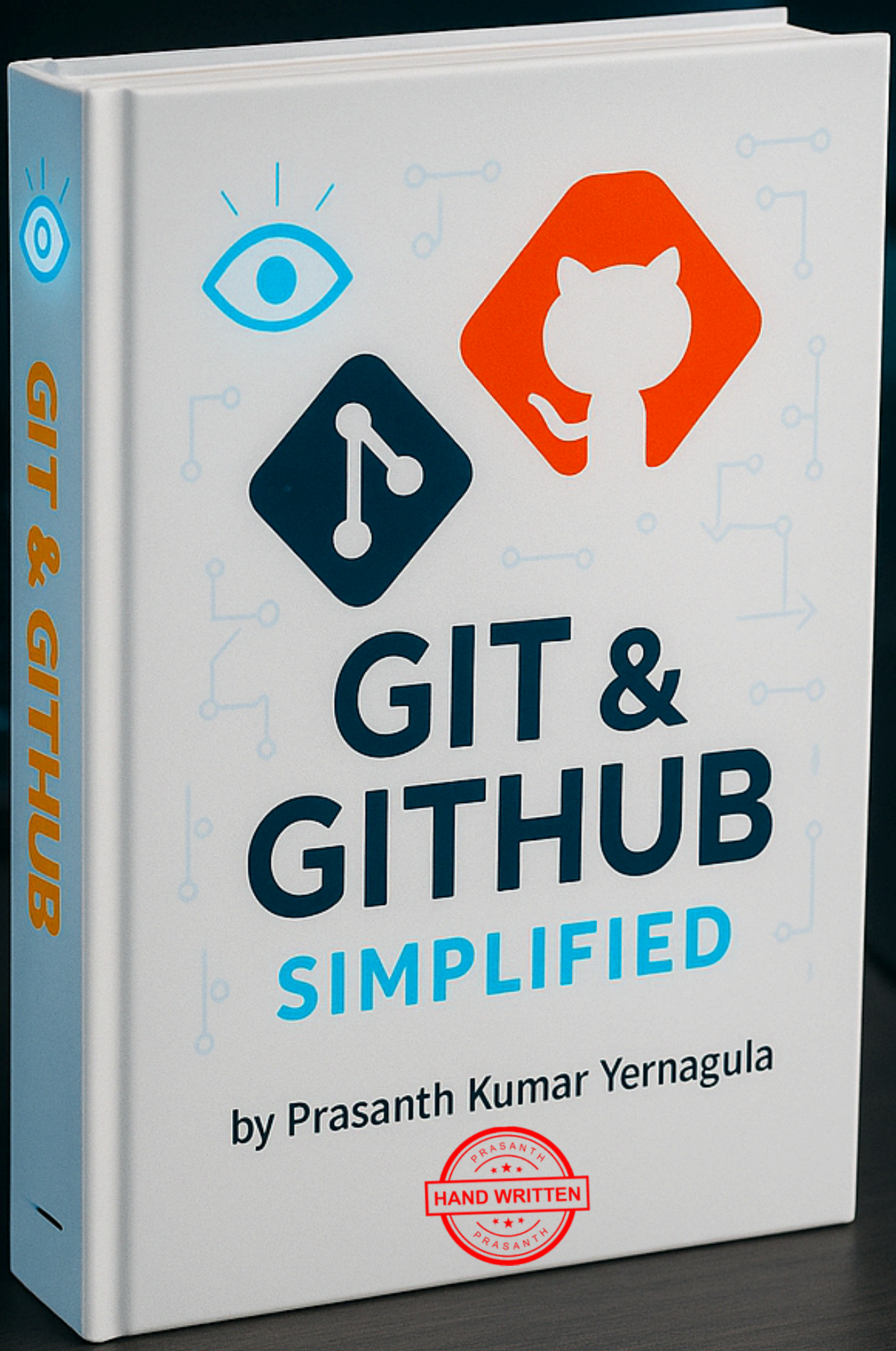
- MutationObserver(): `const observer = new MutationObserver(callback); observer.observe(targetNode, config);`

29. Screen API

- screen.width: `const screenWidth = screen.width;`
- screen.height: `const screenHeight = screen.height;`

30. Navigator API

- navigator.userAgent: `const userAgent = navigator.userAgent;`
- navigator.onLine: `const isOnline = navigator.onLine;`


GIT & GITHUB
GIT & GITHUB
SIMPLIFIED
by Prasanth Kumar Yernagula
PRASANTH
HAND WRITTEN
PRASANTH

"

Git & Git Hub "

Prasanth kumar Yernagula.

Prasanth kumar Yernagula

Git & Git Hub:-

* Introduction :-

→ these are version control systems
→ used to keep track of the different versions of your code

→ helps us in easily roll backing when mistakes happen

→ Version Control :-

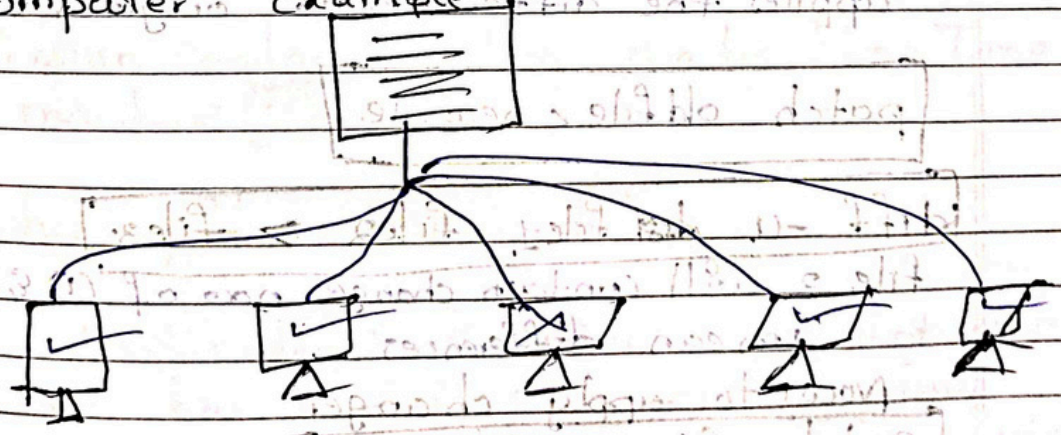
* when we try to manage changes in it its super important to have detailed history of information of organisation configuration file.

* this lets administrator see what was modified and when

* helps in troubleshooting.

* helps in future documentation.

computer example



Version control system allow us to keep track of when and who did what changes to our files. those can be code, configuration, images or whatever else we need to track.

2. keeping Historical Copies :-

"we occasionaly create copies of work in case we wanted to go back to an earlier version."

Diffing files :-

`diff file1 file2` shows difference between 2 file.

`dif -u file1 file2`
↳ "shows differences in a more unified manner."

diff is most popular one. Some more popular tools are.
codiff → highlights difference b/n words.

to help even more there are some graphical tools.
Some of them are : meld, kDiff3, vimdiff.

patch command :-

applies the differences to original file.

`patch oldfile < newfile.`

`diff -u file1 file2 > file3.`
file 3 will contain changes made of ① & ②
i.e mean differences
Now to apply changes
`patch file1 < file3.diff`
will apply changes

(till now we seen existing methods to compare differences & fixing them)

Version Control Systems

* keeps track of changes maker to file
* we can track who made and when made chags.
* colloboration made easier.

"regular file system keeps track of only recent data of file i.e most recent version."

*if we look vcs closer its will just a. stem that stores files, a vcs keeps track of all the different versions that we create. as we save our changes

* vcs also provide mechanish to author why change is committed, including bugs tickets or issues fixed by the change.

*In vcs files are organised to repositiries

* thousands of people can contribute to a single repository.

Version Control System can be like a Time machine

why git is powerful:-

*it is Distributed vcs which means that each developer has a full copy of repository Repositories can be used by as many developers needed

Linus Torvalds → Git.

Git :-

* free open source V.C.S.
* git can work as a standalone program as a server and as a client dont need server
* this means you can use git on a single machine without even having networ..
* git client can also communicate with servers
git website is called git-scm.com
scm stands for Source Control Management.
other vcm's are: "Subversion or Mercurial."

git --version → command to check version.

Repository: a central location in which data is stored and managed.

Commit: collection of edits we are making at one time is called commit.

Linus Torvalds developed Git in 2005 for better facilitating process of developing the Linux kernel with developers across the globe.

First Steps with Git :-
Vcs tracks who made changes for this we have to tell the git whoweare.
command:-
git config --global user.email "mee example.com"
git config --global user.name "My Name"
↳ for all repos
we will talk about remote repo later.

command:- initialising git in current directory
git init
.git file will be created in that repository
* git init is used to initialise new directories
* Area outside git directory is called the "Working Tree".
* it is the current version of your project. you can think of like a sandbox where we perform all operations or modifications we want.
* working tree contain all files currently tracked by git
If we have added a file in working tree
Now to make our git to track our file.
following command is used

command: adding file to staging area.
"git add filenam.extension"

Staging Area:- A file maintained by Git (index)
that contains all of the information about what files and changes are going to go in your next commit.

command: to get some information abaut the current working tree and pending changes
git status

we see that our new file is marked to be committed, this means that our change is currently in "staging area"
to get it commited we use command
git commit

*it opens a text editor where we can enter a commit message
*its the simple description of what we did to the file.

Sections of Git Project:-

Git Directory: contains history of all files and changes
Working Tree. current state of project.
Staging Area: changes that we have been marked to be included in next commit

when we operate on git files can be tracked or untracked
Tracked files are parts of Snapshot
Untracked files are not part of snapshot yet.

a file undergoes. → add, delete, update.

Modified → we made all changes to file not yet committed
staging → changes made to the file are ready to be committed
committed → changes committed

it means a file tracked by git, will first be modified when we change it in any way. Then it becomes staged when we mark those changes for tracking. finally it will get committed.

command:

```
git commit -m "message"
```

Summing up the work we did.
1. modified the file.
2. added to the staging Area.
3. committed to the changes.

Note: you cant commit with a empty commit message.

The Basic Git Workflow :-
1. git init
2. git add
3. git commit -m

Anatomy of a Git Commit Message:
a short description of the change, followed by one or more paragraph giving more details of the change (if needed)

→ Spend time in your documentation now it will definately pay off later.

Command: history of commits is found.

```
git log
```

output: "unique string used as identifier"

Command: to check curren user configuration

```
git config -l
```

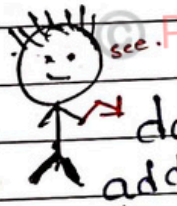
Skipping The Staging Area :-

* first we make changes and then we stage them and then we commit those changes.

but

* if we know that current changes are the ones that we want to commit, we can skip the staging step and go directly to commit.

we use -a flag to git commit command. this flag automatically stages every file thats tracked and modified before doing commit letting it skip the git add.



you might think that git commit dash a is a just a shortcut for git add followed by git commit but thats not exactly true

→ Git commit -a doesn't work on new files that is untracked files

command: shortcut for commit and message.

"git commit -a -m "message"

Getting More Information About Our Changes :-

by default git log shows us the list of

→ commits made in current repo

→ commit message

→ Author

→ Date of change

If we want to look what actual lines are changed in each commit To do with this git log we can use the -p flag. The p comes from patch, because using this flag gives the patch that created

C format is equivalent to the diff -u output that we saw on earlier video)

`git log -P`

Since it shows all commits we can get a particular commit by using show method.

(q exit) — com

show : takes commit id as parameter.

command : stats of repo.

`git log --stat`

To review changes before staging

1. git diff (-shows only unstaged changes)
   -staged (staged info)
2. git add -p

Removing A file :-

command : git rm filename.
git commit -m " "

Renaming A file.

command : git mv oldname NewName.

.gitignore → inside this we specify which files to be ignored for current repo.
(Example automatically generated files may noise while checking / tracking.)

command

ls -la → to see all file.

echo .DS_STORE >> .gitignore

git add .gitignore

git commit -m "added .gitignore file"

Head: is used to indicate what the current checked out snapshot is.

Undoing Changes made before Commit:-

~~staged~~ changed/modified → last committed

command: git checkout filename

change file back to early committed state by using git checkout before it gets Staged.

To reset ~~committed~~ Staged changes

command: git reset HEAD filename

Amending Commits :-

command: git --amend (take back from commit to staging Area)

↓

opens previous commit editor.

git commit --amend allows us to modify and add changes to most recent commit.

(we cant use it in public Repos.)

Avoid amending commits that have already been made to public

Roll Backs:-

* git revert:- a new commit is created with inverse changes.

* This cancels previous changes instead of making it as though the original commit never happened

Scenario:

→ fixing your work before commit is good. what if its already committed.

→ Example company updated code ? result in rare of ticket !

→ its time to roll back.

There are three ways to Roll back the commit in git. Now focus on git revert:-

→ it does not mean that undo it means it creates a commit that contains the inverse of all the changes made in bad commit inorder to cancel them out.

→ so git revert will create a new commit opposite of everything in the given commit.

→ we can revert last commit using head alias that we mentioned before. (or)
→ think of head as the pointer to the snapshot of current commit.

command: `git revert HEAD`

Identifying a Commit :-

"we can target a specific commit by using its commit ID"

commit id is → 40 character Long
→ Calculated by using "SHA1" Algorithm.
→

"what this algorithm does is takes a bunch of data as input and produce a 40 character string"

* Cryptographic algorithms like SHA1 are really complex so we dont go deep into what it means
* Git uses these hash functions to guarantee the consistancy of data repo
* computing hash keeps data consistant because its calculated from all the information that makes up a commit.
the commit message, date, author, author, taken from snapshot of the working tree

If Someone intentionally corrupt some data git can use the hash to spot that corruption.

 → Detective git.

command : git revert identifier.
identifier with atleast 3.

Advanced Git Interactions.

undoing things.

Branching and Merging :—

* Branch: a branch at most basic level is "a pointer to particular commit"

* it represents independent line of development in a project.

* The default branch that git creates for you when a new repository initialized is called master.

* master branch is commonly known to represent the known good state of a project.

* when you want to develop a feature or try something new in your project you can create a seperate branch to do your work... you can merge there.

Creating New Branches :-

we can use git branch command to create, list, delete and manipulate branches.

command: git branch branch_name -
creates new branch.

Switching to new branch :-
command: git checkout branch_name.

"we use git checkout to checkout the latest snapshot for both files and for branches."

command: git branch.
lists all branches, master branch is indicated with *.

command: git checkout -b new branch.
to create and switch branch at a time.

Note: 1) when we switch from branches head pointer also changes.

2) when we switch branches in git, the working directory and commit history will be changed to reflect the snapshot of our project in that branch (logs of that branch)

command:- git branch -d new-feature is used to delete the branch.

Merging :- The term Git uses for combining branched data and history together.

git merge command is used to merge branches

Command git merge branch-name

when we merge 2 branches it means we combine branched data and history together.

git uses 2 algorithms to to perform merge.

- fast forward
- 3-way merge.

Fast forward branch:- the merge performed was fast forward branch. this kind of merge occurs when all the commits in the checked branch are also in the branch that being merged.
if this is the case we can say that both branch history doesn't not "direrge"
So instead of merging we can just change head pointer.

3-way merge:- a 3 way merge is performd when the history of merging branches has diverged in some way, there is not a nice linear path to combine.
{ This happens when a commit is made in one branch after the point when both branches split }
when this occurs, Git will tie the branch histories together with new commit"

git tries to figure out how to merge those snapshots.
* if changes made on same part of same file git conflict will occur.

Merge Conflicts:-
if there is a conflict then git tells so follow instruction to resolve the commit and then finally commit.
command:
Example: git merge --abort. to abort merging

command: git log --graph. --oneline.
(better understand history of commits
So all commits will be one line.
* In Real word merge conflicts are always suspicious

Introduction to GitHub:-
* git is a "distributed" version control system.
Distributed:-
Each developer has a copy of the whole repository on their local machine.
Git hub:-
*Git hub is a "web based" Git repository hosting service
* On top of the version control functionalities of Git, GitHub includes extra features like bug tracking, wikis, and task management.
* it lets us share and access repositories on the other alternative: git lab. Bit Bucket.

web and copy or clone them to our local computer so we can work on them.

"Wiki" :- a seperate documentation section present in git repo
→ Section for hosting documentation in a repo

Cloning Repo :

command git clone url

(it will ask credentials.)

* Readme file of github is in markdown format .md

command git push
* it is used to push our modified repository.
* it is used to update the remote repository to reflect our changes

ways of avoiding entering multiple time password in git.

→ 1. one way is to create an SSH key pair and store the public key in our github profile so that git recognizes our comp
→ 2. other way is to use credential helper which caches credentials for a time window of 15 mins

command git config --global credential.helper cache

Command: git pull
Retrieve new changes in repo

* git pull is used to fetch the newest updates from a remote repository

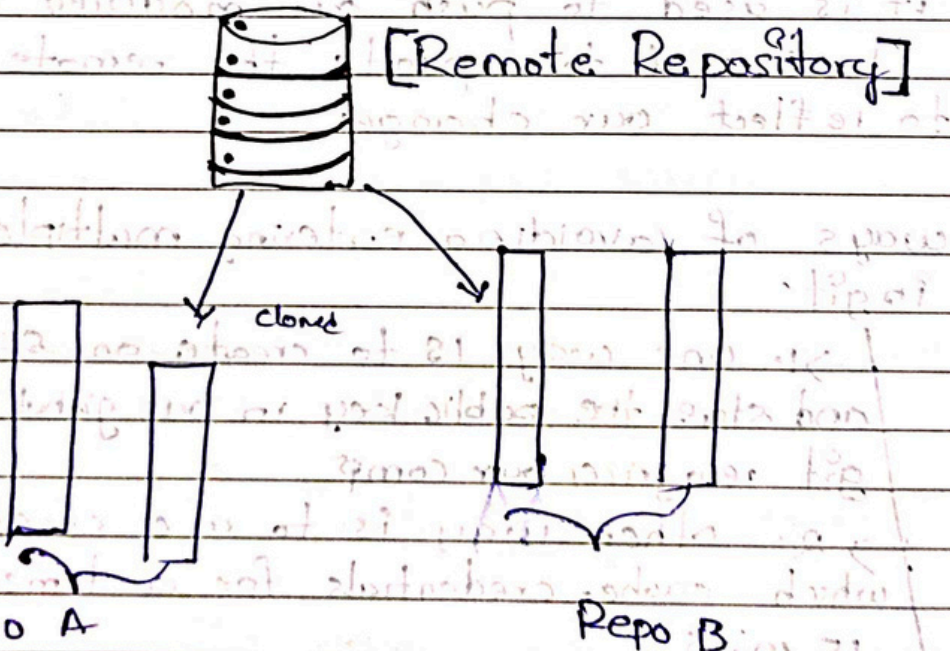
* it also updates the local repository by applying changes made in the remote repository

Using A Remote Repository.

*Remote repositories are big part of distributed nature of git collaboration

* Lets lot of developers to contribute the project from same work-station "making changes to local copy of the project independently of one another."

* when they need to share share changes they use git commands



before pushing they need to be merged by solving merge conflicts

Note:- after commiting, we will fetch any new changes from the remote repo manually merge if necessary and only push our changes to remote repo.

git supports various ways to connect remote Repository:

- HTTP → allows only read only access to repo
- HTTPS → provide authenticate method of user both
- ssh →

its good idea to have control over repo.

* when we call a git clone to get a local copy of a Remote repository

* git setups that remote repository with the default origin name.

command: git remote -v
to check configuration of Remote.
Note:
o/p: ~~url of~~ origin & destn
2 urls → 1. fetch data
→ 2. push data
1. → Http, 2. Https

Here same in some case,

command: git branch -r
used to look at the remote remote branches that our Git repo is currently tracking by.
running git branch -r.

→ its read only
→ its we can look at the commit history like we did with local branch, but
→ we can't make changes directly
→ to do so we hae to undergo the workflow
1. pull from nasly
2. merge with our chan
3. push.

Fetching New changes:-

Command:- git pull → instantly merges
command: ~~git push~~.
git fetch → only retrives remote updates

Updating the Local Repositorys:-

* Earlier we have seen basic work flow for working with remotes
* when we want it to fetch the changes manually merge, if necessary
* But git pull command that does both for us.
* Running git pull will fetch the remote copy of the current branch and automatically try to merge with current branch

↓

performs fast forward merge.

Note: git checkout command can also be used to download remote repository

Note:-
1. git remote update.
will fetch the contents of all remote branches and allow us to merge the contents ourselves

2. git remote show origin
if you want to see more information about a particular remote branch, you can use the git remote show command
Don't forget the commit ID

3. New merge commit creates explicit merge.

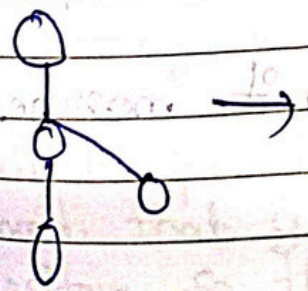
(if remote repo is changed)
↓ git pull → solve conflicts
⇒ git push

pull merge push workflow:-
command: git log -p origin/master
(view patch of remote repo)

fix, add,

command:- the command git checkout -b <branchname>
will first create a branch and switch to it

if changes applied on diffed files git rebase will help to fast forward



pulls our new bar on top of master branch

command :- git rebase <branchname>

Rebasing instead of merging rewrite history and maintains linearity, making for cleaner code.

Best practices for collaboration

1. Always Synchronize your branches before starting any work on your own. (Get most recent version)

2. Avoid having very large changes

3. when working on big change have feature branch.

4. Regularly merge changes made on the master branch back onto feature branch.

5. Have latest version of the project in the master branch and the stable version of the project on a seperate branch

6. you should not rebase changes that have been pushed to remote repos

7. Having good commit messages is very important.

Lab:- use SSH key in Lab
personal access token

module → 4

Collaboration:-

*in recent years github has become a super platform for collaboration across many projects
*it is used heavily in open source project

A simple pull Request on GitHub:-

Forking:-

*A way of creating a copy of the given repository so that it belongs to our user.

*in other words our user will be able to push changes to the forked copy, even when we cant push changes to other repo.

when collaborating on projects hosted on github typical work flow is
1. create the fork of that repo
2. then work on that local ~~Repo~~ forks
(a forked repo is normal repo except github knows which repo is forked from)
3. Eventually can place a merge request by creating a pull request

Pull Request: A commit or series of commits that you send to the owner of the repository so that they incorporate into their repo

(only few people of the project will have commit rights)

Typical Pull Request Workflow:-

Cmd → mark down format
contain simple text followed by some Rules

→ for working on big changes we have to work on our local computer
→ finally push it to remote fork repo using
git push -u origin add -readme.
→ After placing pull request id will be generated

Updating an Existing Pull Request:-

* project maintain might ask some improvements when we place pull request

* make changes in local repo and push it again.

Squashing Changes:-

(git rebase -i master)
(modify or merging & commit)
"git push -f → forcing git"

Note:- fixup operation will keep the original message and discard the message from the fixup commit while squashing combine them

What are code reviews ?

Code Review:-

→ going through someone else code, documentation or configuration and checking that it all makes sense and follows the expected pattern.

→ increases no of eyes, bugs will be fixed, code improves.

→ let us comment on other code.

→ improve code overall project.

→ its about making code better.

→ At Google, we believe deeply in value of reviewing everything we do . even content or course.

Code Review Work flow

code
reveew
merge.

{ python uses PEPs documentation style guide to write code }

How to Code Review on GitHub:-

using ai tools

Managing Collaboration:-

* its important that you understand any changes you accept.

* if you are project maintainer, its important that you reply promptly to pull requests and don't let them stagnate.

when it comes to coordinating who does what and when a common strategy for active software projects is to use an issue tracker.

Tracking issues:-
* Deciding who is going to do what is critical when collaborating with other

* with no coordination 2 or more people spending on same piece of code leaving the critical region

issue tracker → use for bug tracker → to assign some bugs so that people can work on.

Bug Zilla → popular bug tracker.

{pull req & issue reqs have unique no:}

Continious Integration: (e.g. Travis along with github)
* we build and test code every time there is a change.

Continous Deployment: Once our code is automatically built and tested, the next automated step is to continous deployment, which sometimes called continuous delivery (CD)

pipelines:-
Steps to run to get result you want.

Artifacts:
The names used to describe any file that are generated as part of pipeline.

git push -u origin branch_name.

Safe way to merge a git branch into master.

1. git fetch — git fetch
2. git rebasing — git rebase.
3. switching to master
4. pulling changes
5. merging changes
6. pushing changes

will bring lastest commits to

React

The Official Roadmap

Swipe

The Prerequisites

Learning **React** is a great choice for **Frontend Developers** due to it's **large community** and **popularity** in the industry, however, you need to understand that **React** is not a technology for absolute beginners, so here is what you will **need to know**:

✔ Basic HTML

- Common Tags
- Attributes
- File Structure

✔ Basic CSS

- Applying Styles
- Flexbox Layouts
- Media Queries

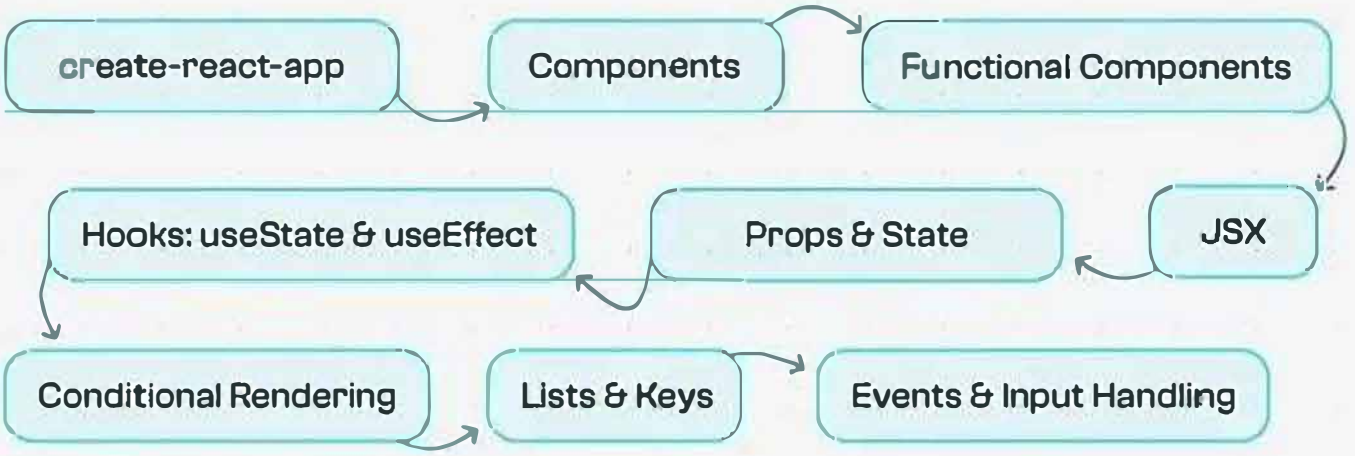
✔ Good Knowledge of Javascript (ES6)

- Variables (let, const)
- Loops & Conditionals
- Arrow Functions
- Arrays, Objects
- Imports and Exports
- How to use NPM

Next

The First Steps

Spend time to **understand the base concepts** of **React** apps, learn how to create a simple application, study the **folder structure** and the concept of components, **your path:**

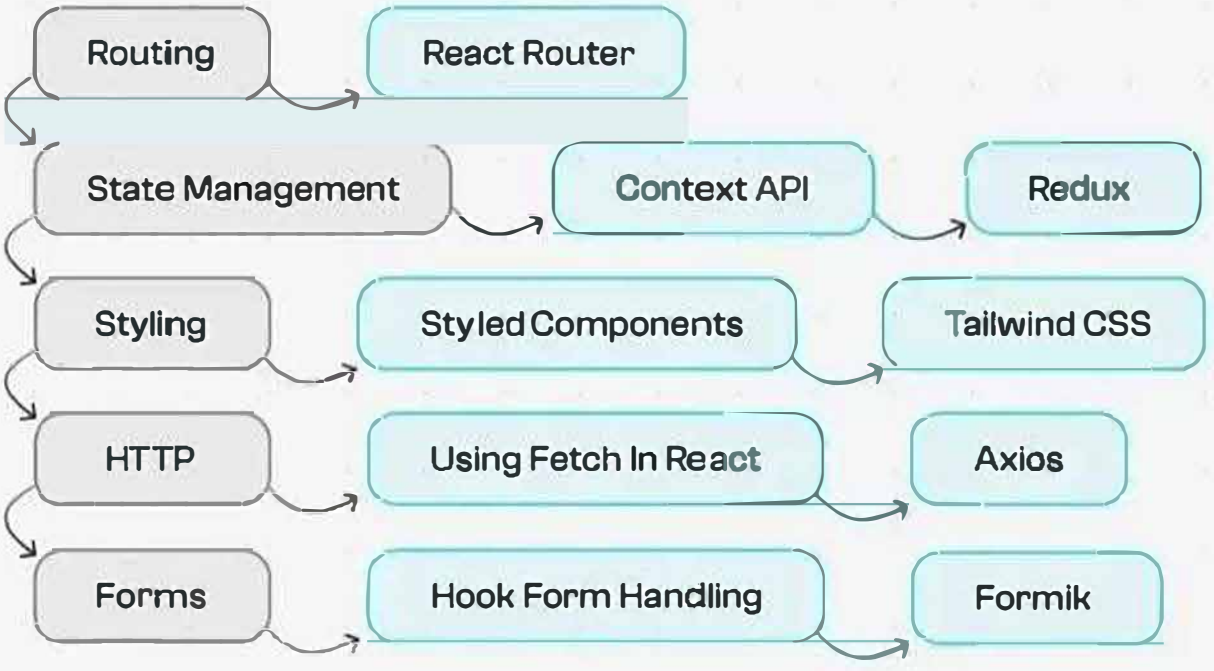


Follow the path and make sure to **practice** what you see, not just watch. **Next**, let's learn about the **React Ecosystem.**

Next

The React Ecosystem

Since **React** is only a **library** and **not a framework**, you will need to work with various other libraries to build **complete apps**, here is the **path to take**:

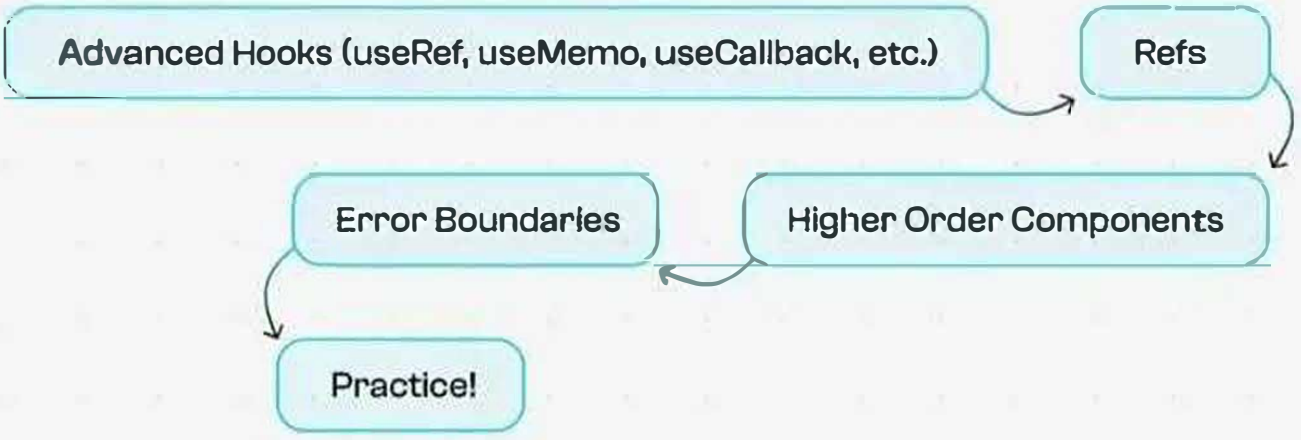


There are plenty of options, but integrating libraries with the core **React** package is something that you will need to master in order to build **advanced applications.**

Next

The Advanced Topics

If you think you mastered the basics, you should move on to **learning these topics:**



Now it's time to practice and build some amazing projects and add them to your portfolio to impress!

Was this useful?

Let me know with a comment

Share it with others

Save it for later




JavaScript Roadmap (Detailed)

---

1. Foundations of Web Development

Before diving into JS deeply, get comfortable with:

- **HTML**
  - Elements, attributes, semantic tags
  - Forms and input types
  - Accessibility basics
- **CSS**
  - Selectors, box model, positioning
  - Flexbox, Grid
  - Transitions & animations
  - Responsive design (media queries, fluid layouts)

👉 *Why?* JavaScript manipulates the DOM, so HTML + CSS knowledge is crucial.

---

2. Core JavaScript (Beginner)

Start with the language fundamentals:

- **Basics**
  - Variables (var, let, const)
  - Data types (string, number, boolean, null, undefined, symbol, bigint)
  - Operators (arithmetic, comparison, logical, assignment, ternary)
- **Control Flow**

- Conditionals (if, switch)
- Loops (for, while, for...of, for...in)

- **Functions**
  - Function declaration vs expression
  - Arrow functions
  - Parameters, default values, rest & spread operators
- **Objects & Arrays**
  - Creating & manipulating objects
  - Array methods (map, filter, reduce, forEach, find, some, every)
- **Scope & Hoisting**
  - Global vs local scope
  - Block scope (let, const)
  - Hoisting rules
- **The DOM**
  - Selecting elements (getElementById, querySelector)
  - Manipulating elements (text, attributes, styles, classes)
  - Event listeners (addEventListener)
  - Forms and input handling
- **Basic Debugging**
  - console.log()
  - Browser DevTools

👉 *Milestone:* Be able to make a simple interactive webpage (like a todo list or counter).

---

3. Core JavaScript (Intermediate)

Go deeper into the language:

- **Functions**
  - Higher-order functions
  - Closures
  - Callback functions
- **ES6+ Features**
  - Template literals
  - Destructuring (objects, arrays)
  - Modules (import/export)
  - Optional chaining
  - Nullish coalescing (??)
- **Objects**
  - Object destructuring
  - Object shorthand
  - Prototypes & inheritance
- **Async JavaScript**
  - Event loop & call stack
  - setTimeout, setInterval
  - Promises
  - async/await
  - Fetch API (making HTTP requests)
- **Error Handling**
  - try/catch/finally
  - Custom errors
- **DOM Advanced**
  - Delegated events
  - Creating & removing nodes

- Browser storage (localStorage, sessionStorage, cookies)

👉 *Milestone:* Build apps like a weather app (using API), a quiz app, or a notes app.

---

4. Advanced JavaScript

Master deeper concepts:

- **JavaScript Engine**
  - How JS runs in browsers (V8, SpiderMonkey)
  - Just-in-time (JIT) compilation
- **Memory Management**
  - Garbage collection
  - Stack vs heap
- **Advanced Objects**
  - this keyword (global, function, object, class contexts)
  - call, apply, bind
  - Object.defineProperty
  - Getters & setters
- **Prototypes & Classes**
  - Prototype chain
  - ES6 class syntax
  - Inheritance
- **Modules & Bundlers**
  - ES modules
  - CommonJS
  - Bundlers: Webpack, Parcel, Vite
- **Asynchronous JS Deep Dive**

    - Event loop phases (microtasks vs macrotasks)
    - Async patterns (callbacks → promises → async/await)
    - Generators & iterators
- **Design Patterns in JS**
    - Singleton, Factory, Module, Observer
- **Functional Programming Concepts**
    - Pure functions
    - Immutability
    - Currying, composition

👉 *Milestone:* Be able to explain the event loop and build large projects without spaghetti code.

---

5. Browser APIs

Learn to interact with the environment:

- DOM APIs (covered earlier, now in depth)
- **Storage APIs**
    - IndexedDB
- **Multimedia APIs**
    - Audio & video
    - Canvas & SVG
- **Network APIs**
    - Fetch, WebSockets
- **Geolocation**
- **Service Workers**
    - Offline apps, caching
- **Web Components**

- Custom elements
- Shadow DOM

---

6. Tooling & Ecosystem

- **Package Managers**
  - npm, yarn, pnpm
- **Transpilers**
  - Babel
- **Linters & Formatters**
  - ESLint, Prettier
- **Testing**
  - Unit testing (Jest, Mocha, Jasmine)
  - End-to-end testing (Cypress, Playwright)
- **Version Control**
  - Git & GitHub basics

---

7. Modern Frameworks & Libraries

Once confident in vanilla JS:

- **React** (most popular choice)
  - JSX, components, hooks, context API
  - State management (Redux, Zustand, Recoil)
- **Vue**
  - Options API, Composition API
- **Angular**
  - Components, modules, services
- **Other Libraries**

- Svelte
- Solid.js

👉 Pick *one* framework and go deep.

---

8. Backend with JavaScript

If you want full-stack:

- **Node.js**
  - Modules, npm
  - File system
  - Events & streams
- **Express.js**
  - REST APIs
  - Middleware
  - Authentication (JWT, sessions)
- **Databases**
  - MongoDB (NoSQL)
  - PostgreSQL/MySQL (SQL)
  - ORMs (Prisma, Sequelize, Mongoose)

---

9. Advanced Topics for Professional Work

- **TypeScript**
  - Typing, interfaces, generics
  - Better tooling & scalability
- **Testing at Scale**
  - Unit, integration, E2E
  - CI/CD pipelines

- **Performance Optimization**
  - Debouncing, throttling
  - Lazy loading
  - Webpack optimization
- **Security**
  - XSS, CSRF, CORS
- **Architecture**
  - MVC, MVVM
  - Monorepos, microservices

---

10. Building Projects

Practical application:

- Beginner: Calculator, Todo app, Quiz app
- Intermediate: Weather app (API), Chat app (WebSockets), Notes app
- Advanced: Full-stack app (React + Node + DB), E-commerce site, Real-time collaboration tool

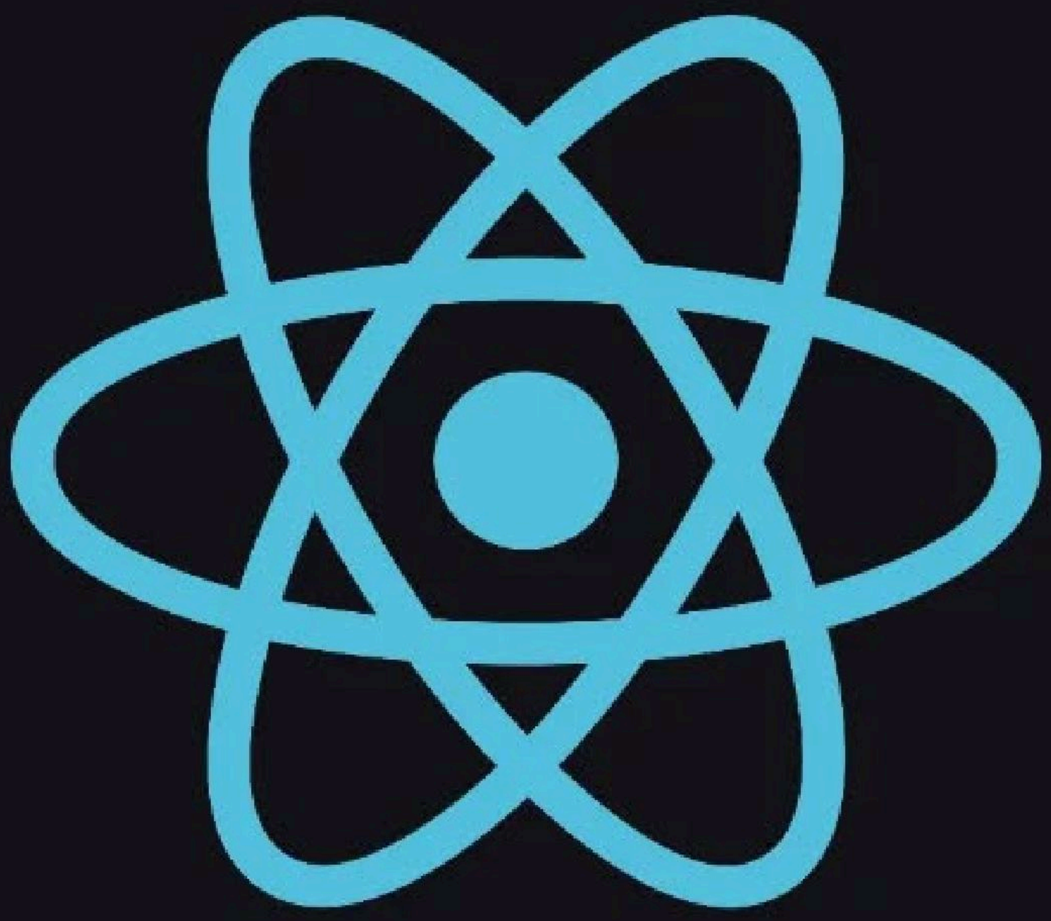
---

11. Career Roadmap

- **Junior Dev:** Strong with HTML/CSS/JS, knows one framework, builds small projects.
- **Mid-level Dev:** Deeper knowledge of JS internals, ecosystem tooling, testing, state management.
- **Senior Dev:** Architecture decisions, performance optimization, TypeScript, backend experience, mentoring.

Save this post

ReactJS CheatSheet



@imperio_coders

1. Setup & Basics

```
// Create App
npx create-react-app my-app

// Functional Component
function Welcome() {
  return <h1>Hello React</h1>;
}

// JSX Rules
const name = "Nikhil";
return <h2>Hello, {name}</h2>;
```

2. Props & State

```
// Props
function Greet({ name }) {
  return <p>Hello, {name}</p>;
}

// State
import { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count + 1)}>{count}</button>;
}
```

Comment “React” for Full PDF

3. useEffect Hook

```jsx
import { useEffect } from 'react';

useEffect(() => {
  console.log("Component mounted");

  return () => {
    console.log("Cleanup");
  };
}, []);
```

4. Event Handling

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

5. Conditional Rendering

```jsx
{isLoggedIn ? <Logout /> : <Login />}

// && operator
{loading && <p>Loading...</p>}
```

Join telegram for free notes (Link in bio)

6. Lists & Keys

```
const items = ["A", "B", "C"];
return (
  <ul>
    {items.map((item, i) => (
      <li key={i}>{item}</li>
    ))}
  </ul>
);
```

7. Forms

```
function Form() {
  const [input, setInput] = useState("");

  return (
    <input
      value={input}
      onChange={(e) => setInput(e.target.value)}
    />
  );
}
```

Comment “React” for Full PDF

8. useContext

```
const ThemeContext = createContext();

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Child />
    </ThemeContext.Provider>
  );
}

function Child() {
  const theme = useContext(ThemeContext);
  return <div>{theme}</div>;
}
```

9. useReducer

```
const reducer = (state, action) => {
  switch (action.type) {
    case "INC": return { count: state.count + 1 };
    default: return state;
  }
};

const [state, dispatch] = useReducer(reducer, { count: 0 });
dispatch({ type: "INC" });
```

10. Custom Hooks

```
function useToggle(initial) {
  const [state, setState] = useState(initial);
  const toggle = () => setState(!state);
  return [state, toggle];
}

// Usage
const [show, toggleShow] = useToggle(false);
```

11. React Router (v6)

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

<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
  </Routes>
</BrowserRouter>
```

12. Memoization

```
const expensive = useMemo(() => compute(data), [data]);
const stableFunc = useCallback(() => doSomething(), []);
```

13. Lifting State Up

```jsx
// Parent Component
function App() {
  const [data, setData] = useState("");
  return <Child update={setData} />;
}

// Child Component
function Child({ update }) {
  return <button onClick={() => update("New Data")}>Send</button>;
}
```

14. Refs & useRef

```jsx
const inputRef = useRef();

<input ref={inputRef} />
<button onClick={() => inputRef.current.focus()}>Focus</button>
```

15. Component Optimization

```jsx
import React, { memo } from "react";

const MyComp = memo(function MyComp({ name }) {
  return <p>{name}</p>;
});
```

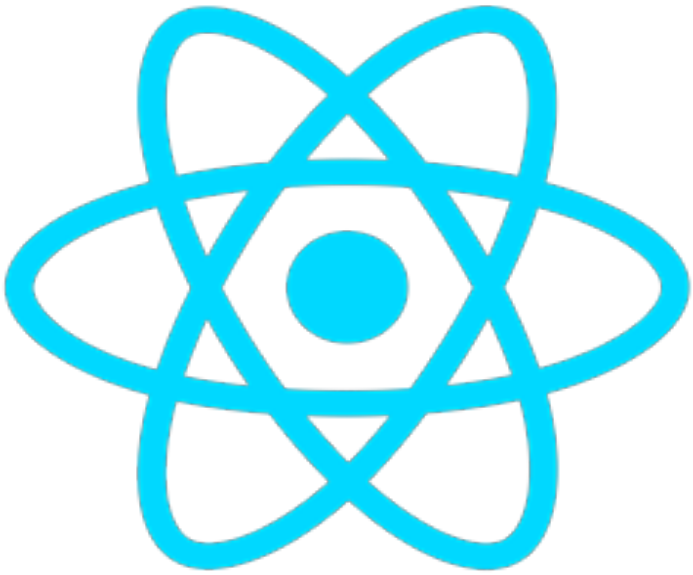
Save It & Follow For More !

Join our Whatsapp Community for free Resources & More!

Link in Bio !

Top 50 React Interview Questions and Answers

Created by: Mohit Decodes



Following are frequently asked React interview questions for fresher's as well as experienced React developers.

1) What is Reactjs?

React is a JavaScript library that makes building user interfaces easy. It was developed by Facebook.

2) Does React use HTML?

No, It uses JSX, which is similar to HTML.

3) When was React first released?

React was first released on March 2013.

4) Give me two most significant drawbacks of React

Integrating React with the MVC framework like Rails requires complex configuration.
React require the users to have knowledge about the integration of user interface into MVC framework.

5) State the difference between Real DOM and Virtual DOM

Real DOM	Virtual DOM
It is updated slowly.	It updates faster.
It allows a direct update from HTML.	It cannot be used to update HTML directly.
It wastes too much memory.	Memory consumption is less

6) What is Flux Concept In React?

Facebook widely uses flux architecture concept for developing client-side web applications. It is ***not*** a framework or a library. It is simply a new kind of architecture that complements React and the concept of Unidirectional Data Flow.

7) Define the term Redux in React

Redux is a library used for front end development. It is a state container for JavaScript applications which should be used for the applications state management. You can test and run an application developed with Redux in different environments.

8) What is the 'Store' feature in Redux?

Redux has a feature called 'Store' which allows you to save the application's entire State at one place. Therefore all it's component's State are stored in the Store so that you will get regular updates directly from the Store. The single state tree helps you to keep track of changes over time and debug or inspect the application.

9) What is an action in Redux?

It is a function which returns an action object. The action-type and the action data are always stored in the action object. Actions can send data between the Store and the software application. All information retrieved by the Store is produced by the actions.

10) Name the important features of React

Here, are important features of React.

- Allows you to use 3rd party libraries
- Time-Saving
- Faster Development
- Simplicity and Composable
- Fully supported by
- Facebook.
- Code Stability with One-directional data binding

React Components

11) Explain the term stateless components

Stateless components are pure functions that render DOM-based solely on the properties provided to them.

12) Explain React Router

React Router is a routing library which allows you to add new screen flows to your application, and it also keeps URL in sync with what’s being shown on the page.

13) What is dispatcher?

A dispatcher is a central hub of app where you will receive actions and broadcast payload to registered callbacks.

14) What is meant by callback function? What is its purpose?

A callback function should be called when setState has finished, and the component is retendered. As the setState is asynchronous, which is why it takes in a second callback function.

15) Explain the term high order component

A higher-order component also shortly known as HOC is an advanced technique for reusing component logic. It is not a part of the React API, but they are a pattern which emerges from React's compositional nature.

16) Explain the Presentational segment

A presentational part is a segment which allows you to renders HTML. The segment's capacity is presentational in markup.

17) What are Props in react js?

Props mean properties, which is a way of passing data from parent to child. We can say that props are just a communication channel between components. It is always moving from parent to child component.

18) Explain yield catchphrase in JavaScript

The yield catchphrase is utilized to delay and resume a generator work, which is known as yield catchphrase.

19) Name two types of React component

Two types of react Components are:

- Function component
- Class component

20) Explain synthetic event in React js

Synthetic event is a kind of object which acts as a cross-browser wrapper around the browser's native event. It also helps us to combine the behaviors of various browser into signal API.

21) What is React State?

It is an object which decides how a specific component renders and how it behaves. The state stores the information which can be changed over the lifetime of a React component.

22) How can you update state in react js?

A state can be updated on the component directly or indirectly.

23) Explain the use of the arrow function in React

The arrow function helps you to predict the behavior of bugs when passed as a callback. Therefore, it prevents bug caused by this all together.

24) State the main difference between Pros and State

The main difference the two is that the State is mutable and Pros are immutable.

25) Explain pure components in React js

Pure components are the fastest components which can replace any component with only a render(). It helps you to enhance the simplicity of the code and performance of the application.

26) What kind of information controls a segment in React?

There are mainly two sorts of information that control a segment: State and Props

- State: State information that will change, we need to utilize State.
- P rops: Props are set by the parent and which are settled all through the lifetime of a part.

27) What is 'create-react-app'?

'create-react-app' is a command-line tool which allows you to create one basic react application.

28) Explain the use of 'key' in react list

Keys allow you to provide each list element with a stable identity. The keys should be unique.

29) What are children prop?

Children props are used to pass component to other components as properties.

30) Explain error boundaries?

Error boundaries help you to catch Javascript error anywhere in the child components. They are most used to log the error and show a fallback UI.

31) What is the use of empty tags ?

Empty tags are used in React for declaring fragments.

32) Explain strict mode

StrictMode allows you to run checks and warnings for react components. It runs only on development build. It helps you to highlight the issues without rendering any visible UI.

33) What are reacted portals?

Portal allows you to render children into a DOM node. ***CreatePortalmethod*** is used for it.

34) What is Context?

React context helps you to pass data using the tree of react components. It helps you to share data globally between various react components.

35) What is the use of Webpack?

Webpack in basically is a module builder. It is mainly runs during the development process.

36) What is Babel in React js?

Babel, is a JavaScript compiler that converts latest JavaScript like ES6, ES7 into plain old ES5 JavaScript that most browsers understand.

37) How can a browser read JSX file?

If you want the browser to read JSX, then that JSX file should be replaced using a JSX transformer like Babel and then send back to the browser.

38) What are the major issues of using MVC architecture in React?

Here are the major challenges you will face while handling MVC architecture:

DOM handling is quite expensive

Most of the time applications were slow and inefficient

Because of circular functions, a complex model has been created around models and ideas

39) What can be done when there is more than one line of expression?

At that time a multi-line JSX expression is the only option left for you.

40) What is the reduction?

The reduction is an application method of handling State.

41) Explain the term synthetic events

It is actually a cross-browser wrapper around the browser's native event. These events have interface stopPropagation() and preventDefault().

42) When should you use the top-class elements for the function element?

If your element does a stage or lifetime cycle, we should use top-class elements.

43) How can you share an element in the parsing?

Using the State, we can share the data.

44) Explain the term reconciliation

When a component's state or props change then rest will compare the rendered element with previously rendered DOM and will update the actual DOM if it is needed. This process is known as reconciliation.

45) How can you re-render a component without using setState() function?

You can use forceUpdate() function for re-rending any component.

46) Can you update props in react?

You can’t update props in react js because props are read-only. Moreover, you can not modify props received from parent to child.

47) Explain the term 'Restructuring.'

Restructuring is extraction process of array objects. Once the process is completed, you can separate each object in a separate variable.

48) Can you update the values of props?

It is not possible to update the value of props as it is immutable.

49) Explain the meaning of Mounting and Demounting

The process of attaching the element tothe DCOMis called mounting.
The process of detaching the element from the DCOM is called the demounting process.

50) What is the use of ‘props-types’ liberary?

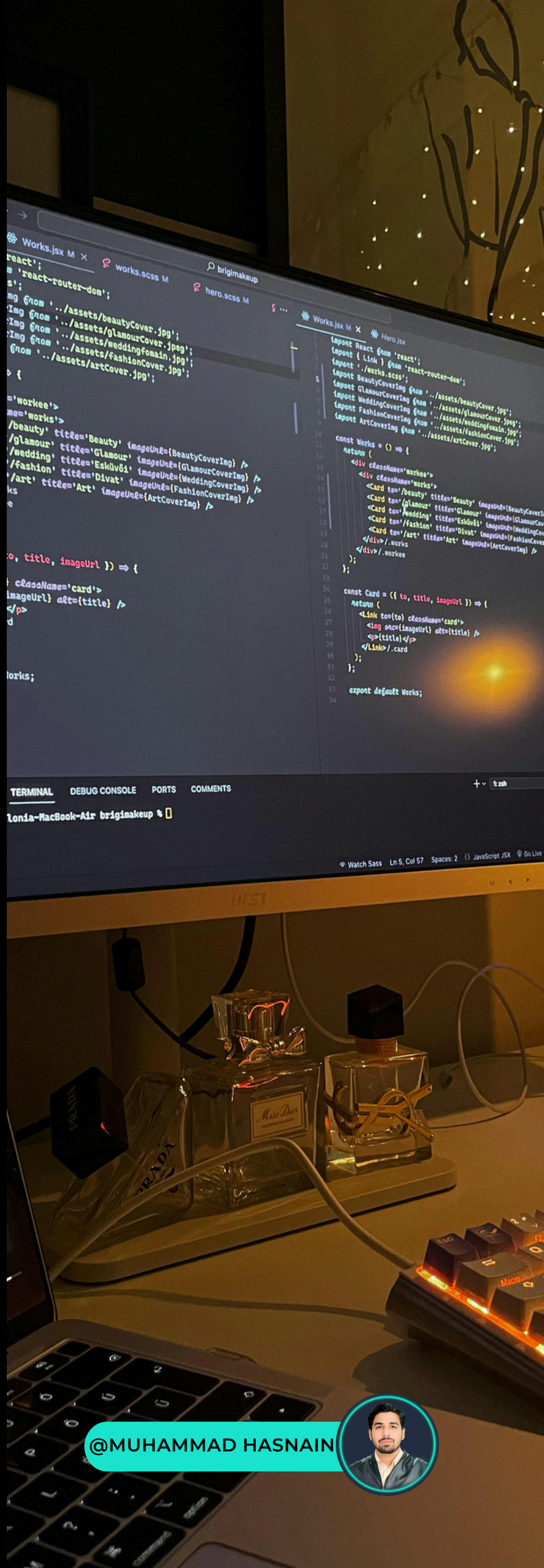
'Prop-types' library allows you to perform runtime type checking for props and similar object in a recent application.

****

BEGINNER DEVELOPER JOURNEY

5 FRONT-END CONCEPTS I WISH I HAD MASTERED EARLIER

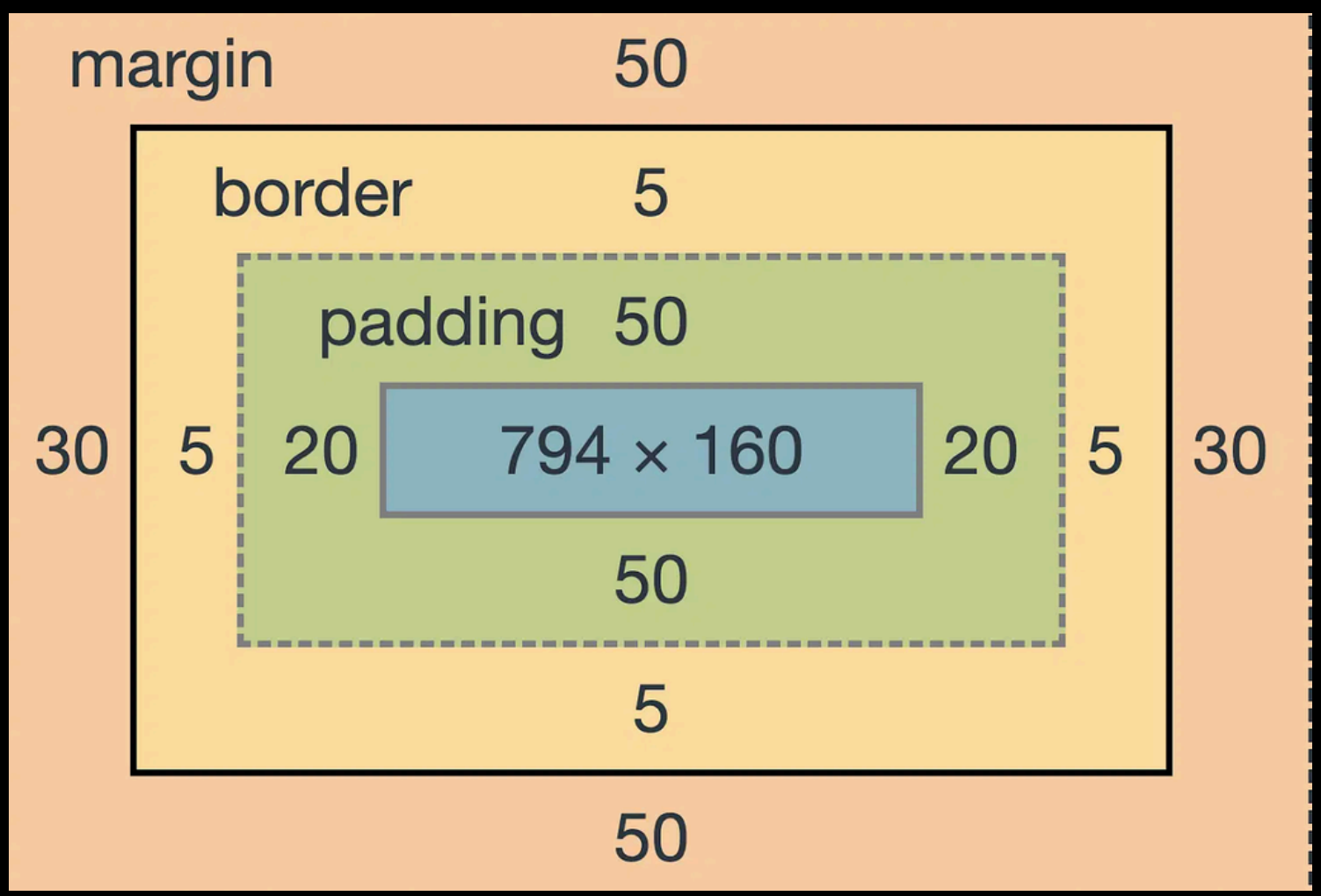
AVOID THESE BEGINNER MISTAKES



BEGINNER DEVELOPER JOURNEY

CONCEPT #1

BOX MODEL



Understanding the box model helps in precise layout control.

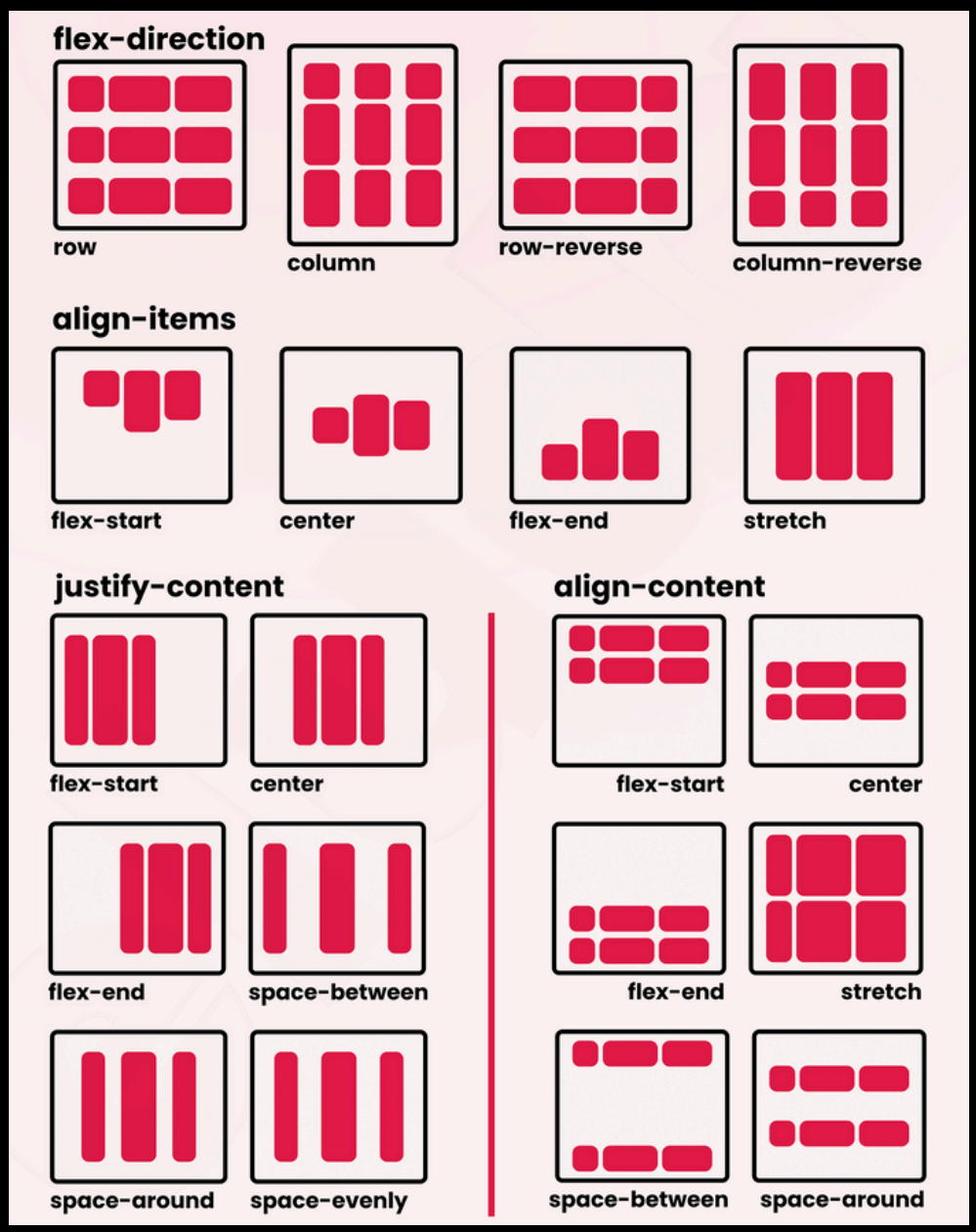
Content → Padding → Border → Margin.

@MUHAMMAD HASNAIN

BEGINNER DEVELOPER JOURNEY

CONCEPT #2

FLEXBOX



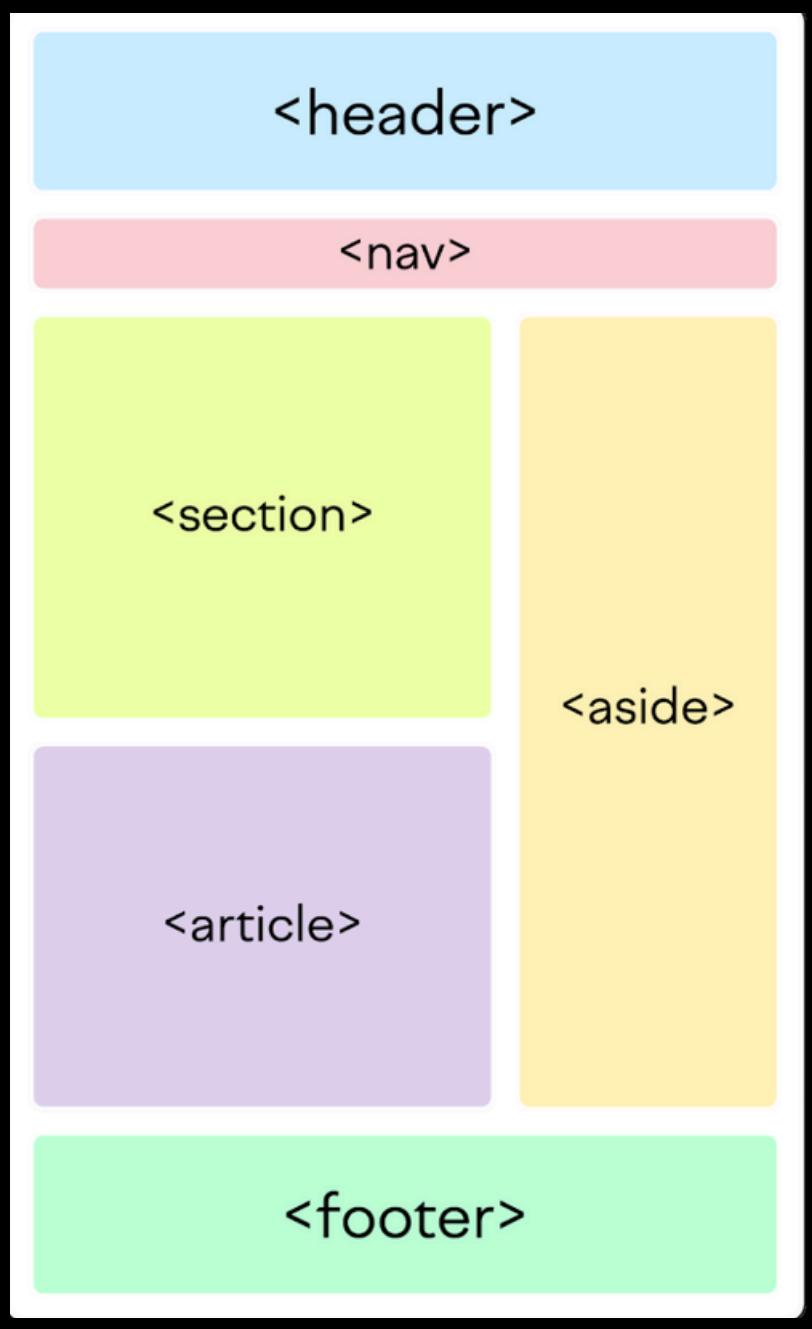
Flexbox makes layout alignment effortless.
Master justify-content and align-items.

@MUHAMMAD HASNAIN

BEGINNER DEVELOPER JOURNEY

CONCEPT #3

SEMANTIC HTML



Using proper tags like <header>, <section>, and <main> improves accessibility & SEO.

@MUHAMMAD HASNAIN

BEGINNER DEVELOPER JOURNEY

CONCEPT #4

CSS SPECIFICITY

Understand how styles are applied.

Order: Inline > ID > Class > Element

@MUHAMMAD HASNAIN

BEGINNER DEVELOPER JOURNEY

CONCEPT #5

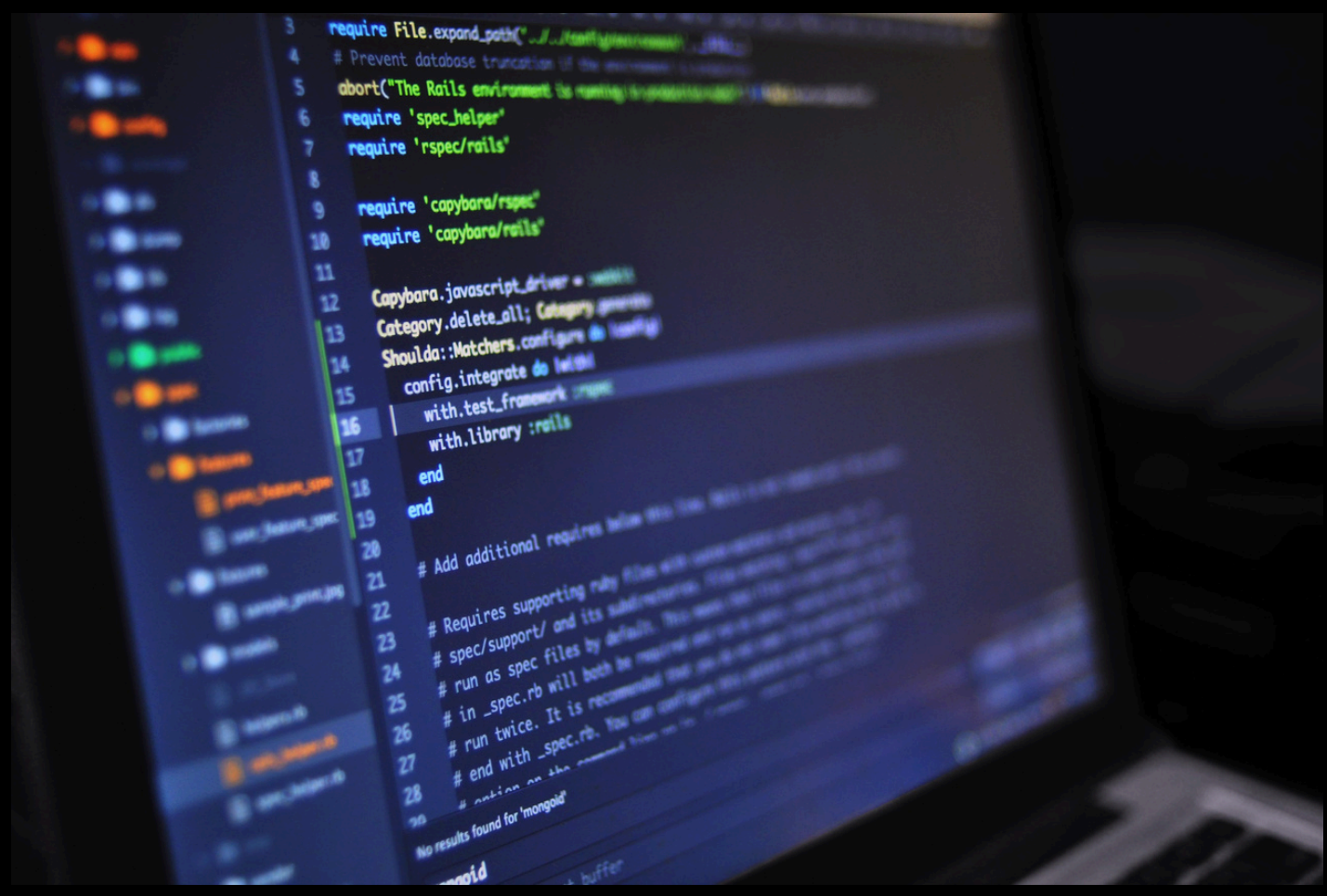
MEDIA QUERIES

Responsive design starts here.
Use @media to make designs mobile-friendly.

@MUHAMMAD HASNAIN

BEGINNER DEVELOPER JOURNEY

Ready to Level Up Your Front-End Skills?



Start today—write cleaner code, build faster, and grow with confidence.

CONNECT WITH ME ON LINKEDIN

@MUHAMMAD HASNAIN

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

LeetCode Cheatsheet

A Curated List of Must Solve Problems

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Preparing for coding interviews used to feel like an endless grind. I spent countless hours searching for the right LeetCode problems, wondering if I was even solving the right ones.

If I had a structured cheat sheet back then, it would have saved me so much time and effort.

That’s why I’m sharing this well curated list of must-solve problems for online assessments and coding interviews.

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Two Pointers

Essential for optimizing array and string problems, reducing time complexity from O(n²) to O(n).

2Pointers - Nov

KSI · 30 questions · 430 Saved

Practice

Updated: a few seconds ago

Progress

0/30 Solved

0 Attempting

Easy 0/12

Med. 0/16

Hard 0/2

763. Partition Labels	Med.	
653. Two Sum IV - Input is a BST	Easy	
1089. Duplicate Zeros	Easy	
658. Find K Closest Elements	Med.	
881. Boats to Save People	Med.	
777. Swap Adjacent in LR String	Med.	
1093. Statistics from a Large Sample	Med.	
680. Valid Palindrome II	Easy	
795. Number of Subarrays with Bounded Maximum	Med.	
905. Sort Array By Parity	Easy	
696. Count Binary Substrings	Easy	
917. Reverse Only Letters	Easy	
1023. Camelcase Matching	Med.	
809. Expressive Words	Med.	
922. Sort Array By Parity II	Easy	
923. 3Sum With Multiplicity	Med.	

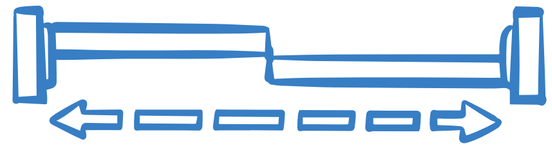
https://leetcode.com/problem-list/9ns2k47r/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Sliding Window



Great for subarray problems, helps in handling streaming data efficiently.

Sliding Window Problems

Pradipta · 12 questions · 457 Saved

Practice

Updated: a few seconds ago

Progress

0/12 Solved

0 Attempting

Easy 0

Med. 0/10

Hard 0/2

930. Binary Subarrays With Sum	Med.	
1234. Replace the Substring for Balanced String	Med.	
424. Longest Repeating Character Replacement	Med.	
1208. Get Equal Substrings Within Budget	Med.	
992. Subarrays with K Different Integers	Hard	
904. Fruit Into Baskets	Med.	
1438. Longest Continuous Subarray With Absolute Dif...	Med.	
209. Minimum Size Subarray Sum	Med.	
1358. Number of Substrings Containing All Three...	Med.	
1004. Max Consecutive Ones III	Med.	
1248. Count Number of Nice Subarrays	Med.	
862. Shortest Subarray with Sum at Least K	Hard	

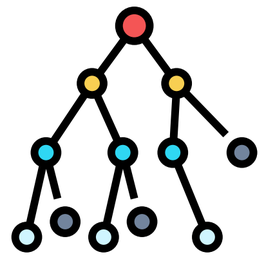
https://leetcode.com/problem-list/x1lbzfk3/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Binary Search



A must-know for efficient searching in sorted datasets, reducing complexity to O(log n).

Binary Search - Beginners

Joseph Cristiano · 41 questions · 482 Saved

Practice

Updated: a few seconds ago

Progress

0/41 Solved

0 Attempting

Easy 0/5

Med. 0/29

Hard 0/7

1283. Find the Smallest Divisor Given a Threshold	Med.	
4. Median of Two Sorted Arrays	Hard	
1292. Maximum Side Length of a Square with Sum Les...	Med.	
1802. Maximum Value at a Given Index in a Bounded...	Med.	
875. Koko Eating Bananas	Med.	
528. Random Pick with Weight	Med.	
658. Find K Closest Elements	Med.	
275. H-Index II	Med.	
1562. Find Latest Group of Size M	Med.	
278. First Bad Version	Easy	
704. Binary Search	Easy	
153. Find Minimum in Rotated Sorted Array	Med.	
410. Split Array Largest Sum	Hard	
154. Find Minimum in Rotated Sorted Array II	Hard	
668. Kth Smallest Number in Multiplication Table	Hard	
1201. Ugly Number III	Med.	

https://leetcode.com/problem-list/504wrexe/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Hash Table & Map

Fundamental for handling large datasets, fast lookup operations, and avoiding nested loops.

Hash Table and Map

Ars · 46 questions · 526 Saved

Practice

Updated: a few seconds ago

Progress

0/46
Solved
0 Attempting

Easy 0/27

Med. 0/19

Hard 0

1. Two Sum	Easy	
3. Longest Substring Without Repeating Characters	Med.	
387. First Unique Character in a String	Easy	
389. Find the Difference	Easy	
645. Set Mismatch	Easy	
1679. Max Number of K-Sum Pairs	Med.	
136. Single Number	Easy	
138. Copy List with Random Pointer	Med.	
525. Contiguous Array	Med.	
771. Jewels and Stones	Easy	
1189. Maximum Number of Balloons	Easy	
18. 4Sum	Med.	
274. H-Index	Med.	
1002. Find Common Characters	Easy	
884. Uncommon Words from Two Sentences	Easy	
409. Longest Palindrome	Easy	

https://leetcode.com/problem-list/9ixr4vaj/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Linked List

Commonly tested in FAANG interviews, important for understanding memory management.

Linked List

Mahima Arora · 13 questions · 465 Saved

Practice

Updated: a few seconds ago

Progress

0/13 Solved

0 Attempting

Easy 0/1

Med. 0/11

Hard 0/1

2. Add Two Numbers	Med.	
328. Odd Even Linked List	Med.	
138. Copy List with Random Pointer	Med.	
109. Convert Sorted List to Binary Search Tree	Med.	
142. Linked List Cycle II	Med.	
143. Reorder List	Med.	
1019. Next Greater Node In Linked List	Med.	
19. Remove Nth Node From End of List	Med.	
21. Merge Two Sorted Lists	Easy	
86. Partition List	Med.	
23. Merge k Sorted Lists	Hard	
24. Swap Nodes in Pairs	Med.	
92. Reverse Linked List II	Med.	

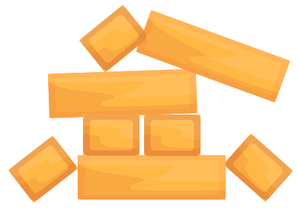
https://leetcode.com/problem-list/9rizphpj/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Heaps



Crucial for priority-based problems like scheduling and finding the k-largest elements.

heap

Joseph Cristiano · 22 questions · 384 Saved

Practice

Updated: a few seconds ago

Progress

0/22 Solved

0 Attempting

Easy 0/1

Med. 0/12

Hard 0/9

871. Minimum Number of Refueling Stops	Hard	
264. Ugly Number II	Med.	
767. Reorganize String	Med.	
1439. Find the Kth Smallest Sum of a Matrix With Sort...	Hard	
703. Kth Largest Element in a Stream	Easy	
23. Merge k Sorted Lists	Hard	
295. Find Median from Data Stream	Hard	
692. Top K Frequent Words	Med.	
451. Sort Characters By Frequency	Med.	
215. Kth Largest Element in an Array	Med.	
347. Top K Frequent Elements	Med.	
480. Sliding Window Median	Hard	
358. Rearrange String k Distance Apart	Hard	
1167. Minimum Cost to Connect Sticks	Med.	
621. Task Scheduler	Med.	
373. Find K Pairs with Smallest Sums	Med.	

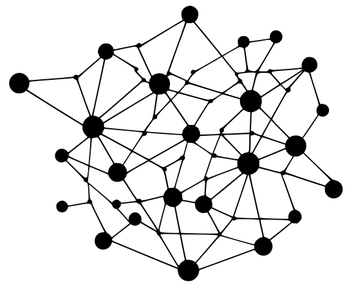
https://leetcode.com/problem-list/93chilpi/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Graphs



Key for solving network and connectivity problems in real-world applications.

Graph Related Problems

Joseph Cristiano · 46 questions · 533 Saved

Practice

Updated: a few seconds ago

Progress

0/46 Solved

0 Attempting

Easy 0/2

Med. 0/42

Hard 0/2

130. Surrounded Regions	Med.	
133. Clone Graph	Med.	
990. Satisfiability of Equality Equations	Med.	
994. Rotting Oranges	Med.	
399. Evaluate Division	Med.	
997. Find the Town Judge	Easy	
1091. Shortest Path in Binary Matrix	Med.	
882. Reachable Nodes In Subdivided Graph	Hard	
886. Possible Bipartition	Med.	
1311. Get Watched Videos by Your Friends	Med.	
542. 01 Matrix	Med.	
417. Pacific Atlantic Water Flow	Med.	
785. Is Graph Bipartite?	Med.	
547. Number of Provinces	Med.	
1319. Number of Operations to Make Network...	Med.	
1466. Reorder Routes to Make All Paths Lead to the Ci...	Med.	

https://leetcode.com/problem-list/9x1uea1h/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Trees

Important for hierarchical data structures, recursion, and divide & conquer techniques.

Tree

Joseph Cristiano · 25 questions · 486 Saved

Practice

Updated: a few seconds ago

Progress

0/25 Solved

0 Attempting

Easy 0/7

Med. 0/16

Hard 0/2

654. Maximum Binary Tree	Med.	
662. Maximum Width of Binary Tree	Med.	
543. Diameter of Binary Tree	Easy	
297. Serialize and Deserialize Binary Tree	Hard	
437. Path Sum III	Med.	
572. Subtree of Another Tree	Easy	
199. Binary Tree Right Side View	Med.	
208. Implement Trie (Prefix Tree)	Med.	
98. Validate Binary Search Tree	Med.	
226. Invert Binary Tree	Easy	
100. Same Tree	Easy	
102. Binary Tree Level Order Traversal	Med.	
103. Binary Tree Zigzag Level Order Traversal	Med.	
104. Maximum Depth of Binary Tree	Easy	
105. Construct Binary Tree from Preorder and Inorder...	Med.	
230. Kth Smallest Element in a BST	Med.	

https://leetcode.com/problem-list/9ak7i9wv/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Trees

More in-depth coverage of tree traversal, balancing, and BST optimizations.

Tree

Joseph Cristiano · 25 questions · 486 Saved

Practice

Updated: a few seconds ago

Progress

0/25 Solved

0 Attempting

Easy 0/7

Med. 0/16

Hard 0/2

654. Maximum Binary Tree	Med.	
662. Maximum Width of Binary Tree	Med.	
543. Diameter of Binary Tree	Easy	
297. Serialize and Deserialize Binary Tree	Hard	
437. Path Sum III	Med.	
572. Subtree of Another Tree	Easy	
199. Binary Tree Right Side View	Med.	
208. Implement Trie (Prefix Tree)	Med.	
98. Validate Binary Search Tree	Med.	
226. Invert Binary Tree	Easy	
100. Same Tree	Easy	
102. Binary Tree Level Order Traversal	Med.	
103. Binary Tree Zigzag Level Order Traversal	Med.	
104. Maximum Depth of Binary Tree	Easy	
105. Construct Binary Tree from Preorder and Inorder...	Med.	
230. Kth Smallest Element in a BST	Med.	

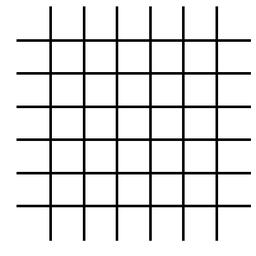
https://leetcode.com/problem-list/9ak7i9wv/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Dynamic Programming



The toughest yet most valuable technique, used in optimal substructure and overlapping subproblems.

DP for Beginners

Joseph Cristiano · 49 questions · 707 Saved

Practice

Updated: a few seconds ago

Progress

0/49 Solved

0 Attempting

Easy 0/1

Med. 0/41

Hard 0/7

983. Minimum Cost For Tickets	Med.	
132. Palindrome Partitioning II	Hard	
518. Coin Change II	Med.	
646. Maximum Length of Pair Chain	Med.	
647. Palindromic Substrings	Med.	
877. Stone Game	Med.	
279. Perfect Squares	Med.	
416. Partition Equal Subset Sum	Med.	
673. Number of Longest Increasing Subsequence	Med.	
300. Longest Increasing Subsequence	Med.	
174. Dungeon Game	Hard	
304. Range Sum Query 2D - Immutable	Med.	
688. Knight Probability in Chessboard	Med.	
1218. Longest Arithmetic Subsequence of Given...	Med.	
309. Best Time to Buy and Sell Stock with Cooldown	Med.	
312. Burst Balloons	Hard	

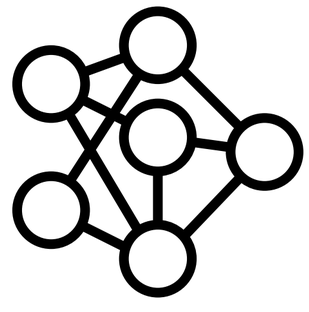
https://leetcode.com/problem-list/9x5spweh/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Greedy Algorithms



Great for optimization problems, making locally optimal choices at each step.

Greedy Related Problems copy 1

Joseph Cristiano · 34 questions · 538 Saved

Practice

Updated: a few seconds ago

Progress

0/34 Solved

0 Attempting

Easy 0/5

Med. 0/28

Hard 0/1

763. Partition Labels	Med.	
984. String Without AAA or BBB	Med.	
870. Advantage Shuffle	Med.	
134. Gas Station	Med.	
135. Candy	Hard	
767. Reorganize String	Med.	
874. Walking Robot Simulation	Med.	
1296. Divide Array in Sets of K Consecutive Numbers	Med.	
402. Remove K Digits	Med.	
1433. Check If a String Can Break Another String	Med.	
881. Boats to Save People	Med.	
406. Queue Reconstruction by Height	Med.	
1094. Car Pooling	Med.	
45. Jump Game II	Med.	
1217. Minimum Cost to Move Chips to The Same...	Easy	
55. Jump Game	Med.	

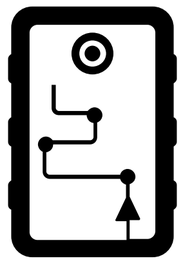
https://leetcode.com/problem-list/925p7hr1/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Backtracking



Powerful for solving permutations, combinations, and constraint-satisfaction problems.

backtrack

Joseph Cristiano · 17 questions · 457 Saved

Practice

Updated: a few seconds ago

Progress

0/17 Solved

0 Attempting

Easy 0

Med. 0/17

Hard 0

784. Letter Case Permutation	Med.
131. Palindrome Partitioning	Med.
39. Combination Sum	Med.
40. Combination Sum II	Med.
1079. Letter Tile Possibilities	Med.
1415. The k-th Lexicographical String of All Happy...	Med.
77. Combinations	Med.
46. Permutations	Med.
47. Permutations II	Med.
78. Subsets	Med.
17. Letter Combinations of a Phone Number	Med.
526. Beautiful Arrangement	Med.
797. All Paths From Source to Target	Med.
373. Find K Pairs with Smallest Sums	Med.
22. Generate Parentheses	Med.
216. Combination Sum III	Med.

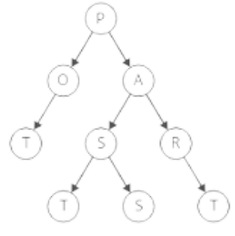
https://leetcode.com/problem-list/9x9qz3md/

Tauseef Fayyaz

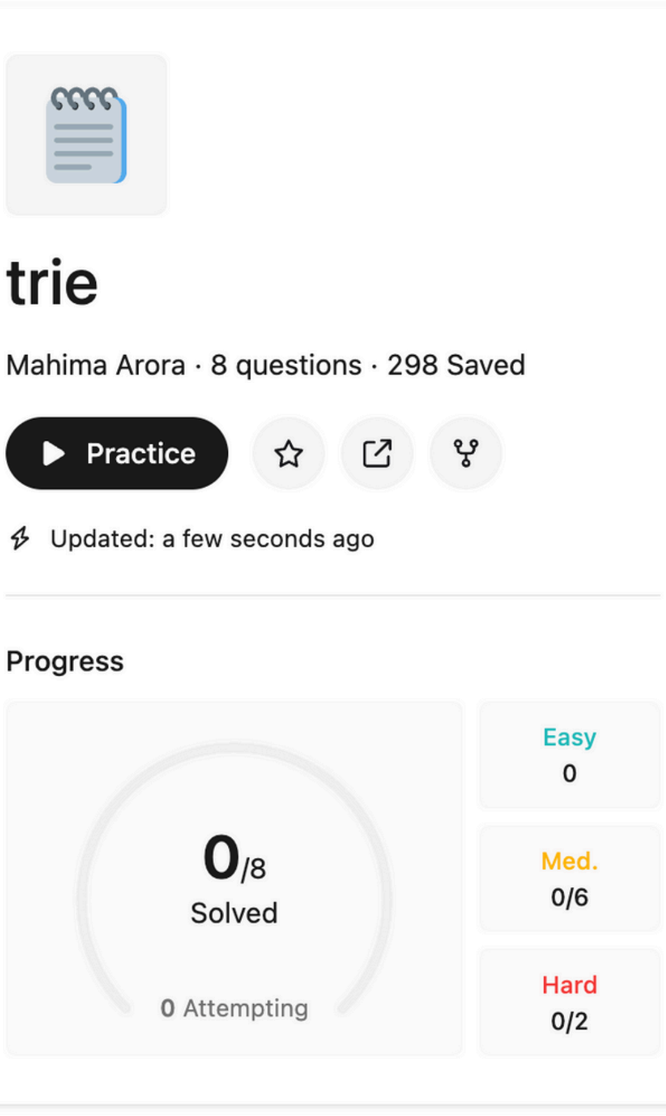
Follow for coding, software and career tips

@tauseeffayyaz

Trie



Best for efficient word searching, autocomplete, and prefix-based problems.



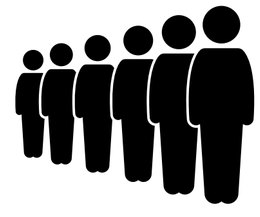
https://leetcode.com/problem-list/5uyupjcr/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Monotonic & Priority Queue



Useful for maintaining ordered sequences and optimizing real-time decision-making.

queue

Mahima Arora · 13 questions · 308 Saved

Practice

Updated: a few seconds ago

Progress

0/13 Solved

0 Attempting

Easy 0

Med. 0/4

Hard 0/9

480. Sliding Window Median	Hard	
739. Daily Temperatures	Med.	
1425. Constrained Subsequence Sum	Hard	
295. Find Median from Data Stream	Hard	
901. Online Stock Span	Med.	
239. Sliding Window Maximum	Hard	
857. Minimum Cost to Hire K Workers	Hard	
907. Sum of Subarray Minimums	Med.	
84. Largest Rectangle in Histogram	Hard	
85. Maximal Rectangle	Hard	
375. Guess Number Higher or Lower II	Med.	
975. Odd Even Jump	Hard	
862. Shortest Subarray with Sum at Least K	Hard	

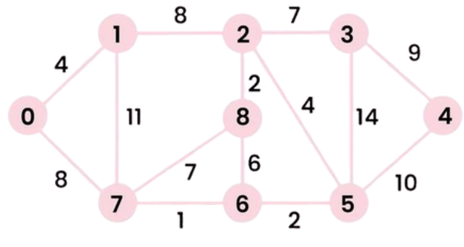
https://leetcode.com/problem-list/9rt1jt27/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Dijkstra’s Algorithm



Key for shortest path problems in weighted graphs, widely used in maps and routing.

Graph - Dijkstra's

Joseph Cristiano · 14 questions · 391 Saved

Practice

Updated: a few seconds ago

Progress

0/14 Solved

0 Attempting

Easy 0

Med. 0/9

Hard 0/5

787. Cheapest Flights Within K Stops	Med.	
1293. Shortest Path in a Grid with Obstacles Elimination	Hard	
743. Network Delay Time	Med.	
1976. Number of Ways to Arrive at Destination	Med.	
1514. Path with Maximum Probability	Med.	
1368. Minimum Cost to Make at Least One Valid Path i...	Hard	
1334. Find the City With the Smallest Number of...	Med.	
499. The Maze III	Hard	
1091. Shortest Path in Binary Matrix	Med.	
1631. Path With Minimum Effort	Med.	
882. Reachable Nodes In Subdivided Graph	Hard	
407. Trapping Rain Water II	Hard	
1786. Number of Restricted Paths From First to Last...	Med.	
505. The Maze II	Med.	

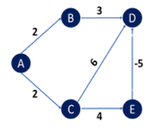
https://leetcode.com/problem-list/9id5lube/

Tauseef Fayyaz

Follow for coding, software and career tips

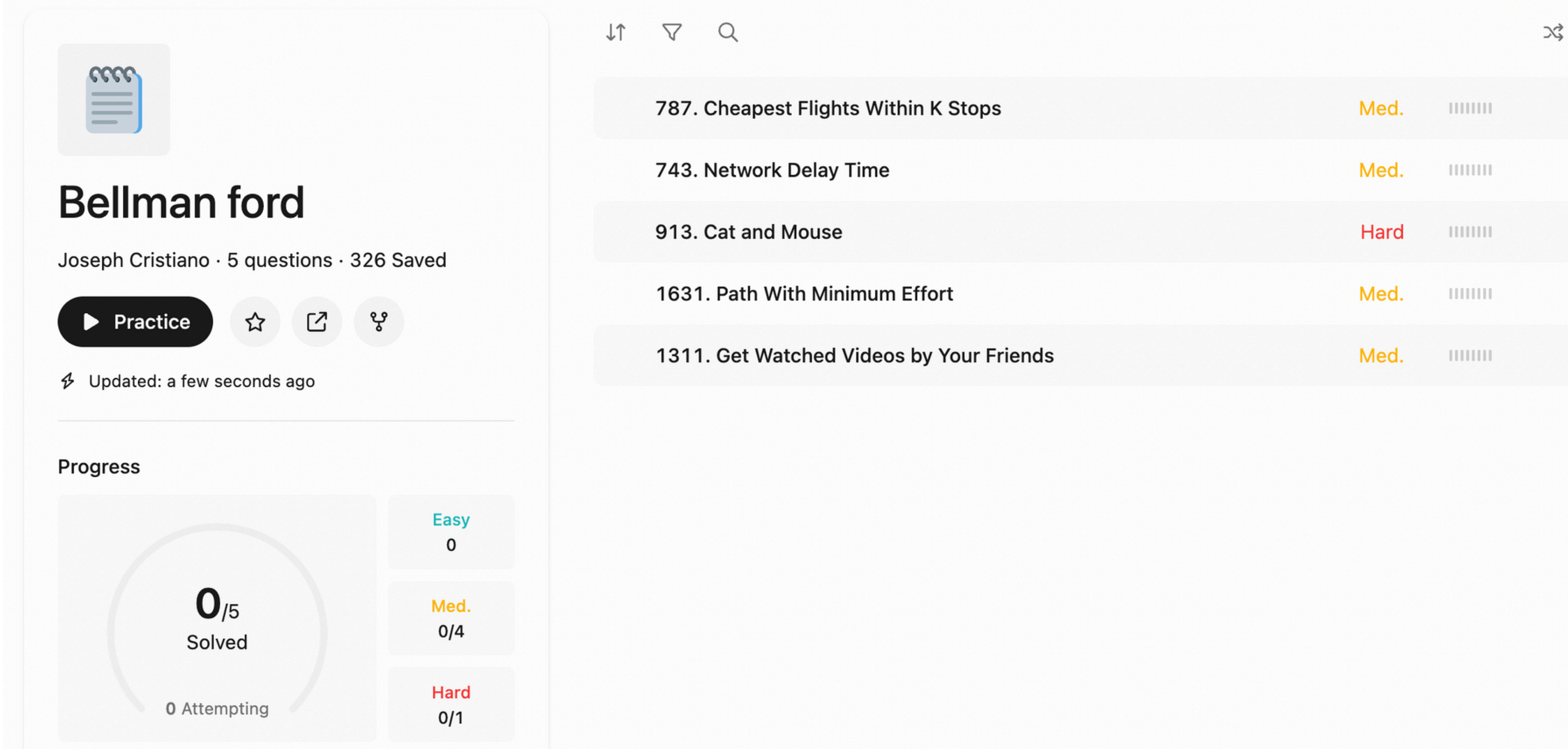
@tauseeffayyaz

Bellman-Ford Algorithm



	B	C	D	E
0	∞	∞	∞	∞
0	2	2	∞	∞
0	2	2	2	6
0	2	2	3	6
0	2	2	3	6

Similar to Dijkstra but works with negative weights, useful in finance and network routing.



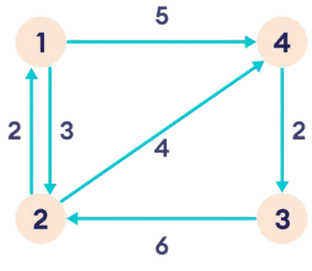
https://leetcode.com/problem-list/9id9smj2/

Tauseef Fayyaz

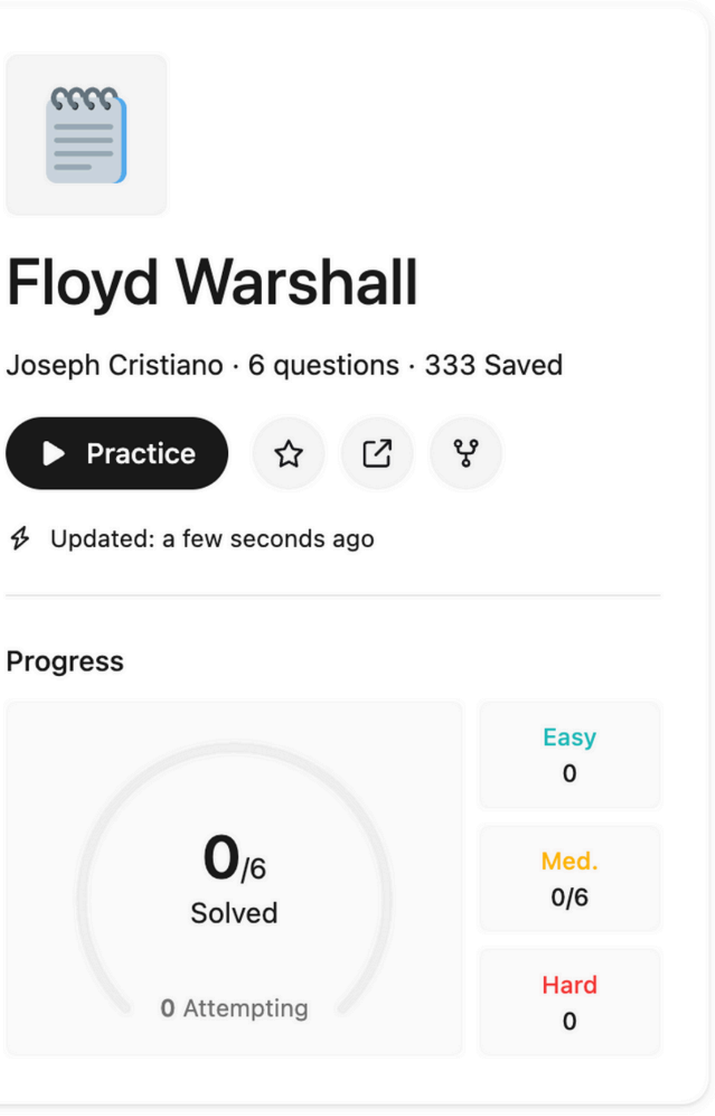
Follow for coding, software and career tips

@tauseeffayyaz

Floyd-Warshall Algorithm



Used for finding shortest paths between all pairs of nodes in a graph.



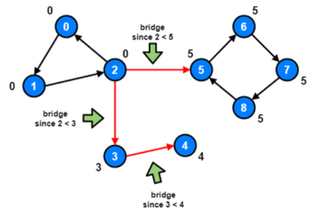
https://leetcode.com/problem-list/9idenloe/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Bridges & Articulation Points



Advanced graph concepts used for network reliability and vulnerability analysis.

Bridges&articulation pts

Joseph Cristiano · 6 questions · 282 Saved

Practice

Updated: a few seconds ago

Progress

0/6 Solved

0 Attempting

Easy 0

Med. 0/1

Hard 0/5

924. Minimize Malware Spread — Hard

928. Minimize Malware Spread II — Hard

1584. Min Cost to Connect All Points — Med.

1489. Find Critical and Pseudo-Critical Edges in... — Hard

1192. Critical Connections in a Network — Hard

1568. Minimum Number of Days to Disconnect Island — Hard

https://leetcode.com/problem-list/9id9ahz7/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Disjoint Set Union

Vital for connectivity problems, cycle detection, and Kruskal's algorithm.

DSU Problems @rowe 1227

kimiquokka · 25 questions · 629 Saved

Practice

Updated: a few seconds ago

Progress

0/25 Solved

0 Attempting

Easy 0

Med. 0/12

Hard 0/13

128. Longest Consecutive Sequence	Med.	
261. Graph Valid Tree	Med.	
990. Satisfiability of Equality Equations	Med.	
1697. Checking Existence of Edge Length Limited Paths	Hard	
1202. Smallest String With Swaps	Med.	
1061. Lexicographically Smallest Equivalent String	Med.	
547. Number of Provinces	Med.	
1319. Number of Operations to Make Network...	Med.	
1579. Remove Max Number of Edges to Keep Graph...	Hard	
1258. Synonymous Sentences	Med.	
305. Number of Islands II	Hard	
1101. The Earliest Moment When Everyone Become...	Med.	
924. Minimize Malware Spread	Hard	
928. Minimize Malware Spread II	Hard	
711. Number of Distinct Islands II	Hard	
1627. Graph Connectivity With Threshold	Hard	

https://leetcode.com/problem-list/5lhmb4mj/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Bit Manipulation

```
10110
10110
01100
```

Optimizes mathematical operations, reduces memory usage, and speeds up calculations.

Bit Manipulation

Joseph Cristiano · 30 questions · 429 Saved

Practice

Updated: a few seconds ago

Progress

0/30 Solved

0 Attempting

Easy 0/16

Med. 0/12

Hard 0/2

1178. Number of Valid Words for Each Puzzle	Hard	
1290. Convert Binary Number in a Linked List to Integer	Easy	
260. Single Number III	Med.	
389. Find the Difference	Easy	
645. Set Mismatch	Easy	
136. Single Number	Easy	
137. Single Number II	Med.	
393. UTF-8 Validation	Med.	
268. Missing Number	Easy	
405. Convert a Number to Hexadecimal	Easy	
1009. Complement of Base 10 Integer	Easy	
287. Find the Duplicate Number	Med.	
1318. Minimum Flips to Make a OR b Equal to c	Med.	
421. Maximum XOR of Two Numbers in an Array	Med.	
169. Majority Element	Easy	
1734. Decode XORed Permutation	Med.	

https://leetcode.com/problem-list/92qvw6c6/

Tauseef Fayyaz

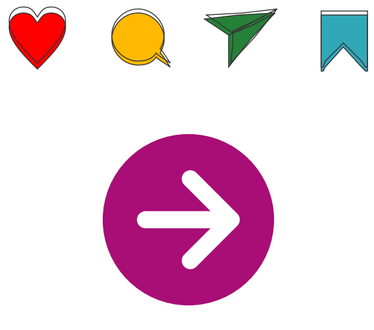
Follow for coding, software and career tips

@tauseeffayyaz

What’s Next?

Let me know what you’re struggling with, and I’ll bring the most helpful tips and resources to you.

It takes time and effort to prepare these valuable resources, but I’m committed to helping you.



Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

Your support keeps me motivated

https://www.linkedin.com/in/tauseeffayyaz/

https://x.com/tauseeffayyaz0

https://www.instagram.com/tauseeffayyaz/

Tauseef Fayyaz

Follow for coding, software and career tips

@tauseeffayyaz

THANK YOU

LIKE & REPOST