Questions:

What is http vs https.

```
HTTP (HyperText Transfer Protocol) and HTTPS (HyperText Transfer Protocol
Secure) are protocols used for transmitting data over the internet. The
primary distinction between them lies in security:

HTTP:

  - Transmits data in plaintext, making it susceptible to interception and
tampering.
  - Operates over port 80.
  - Lacks encryption, posing risks when handling sensitive information like
passwords or credit card details.

HTTPS:

  - Enhances HTTP by incorporating SSL/TLS encryption, ensuring data
confidentiality and integrity during transmission.
  - Operates over port 443.
  - Requires a digital certificate from a Certificate Authority (CA) to
authenticate the server's identity.
  - Protects against eavesdropping and man-in-the-middle attacks, making it
essential for secure communications.

In summary, while HTTP is suitable for general information transfer, HTTPS
is crucial for secure data exchange, especially when dealing with sensitive
information.
```

Guess the Output of JS code.

```
const a = { foo: 123 };
const b = Object.create(a);
b.foo = 444; // A
delete b.foo; // B

Inheritance & prototype chain
# b.foo => 123
# a.fo
o => 123
```

Code: Resolve array of async operations and handle error if any in between. (even add, odd reject)

```
function checkEvenOdd(number) {
  return new Promise((resolve, reject) => {
    if (typeof number !== 'number') {
      reject(new Error('Input must be a number'));
    } else if (number % 2 === 0) {
      resolve(number);
    } else {
      reject(new Error(`${number} is an odd number`));
    }
  });
}
```

HTML: Create square and place it in centre.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Centered Square with Flexbox</title>
  <style>
    .container {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh; /* Full viewport height */
      background-color: #f0f0f0;
    }
    .square {
      width: 100px;
      height: 100px;
      background-color: #007bff;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="square"></div>
  </div>
</body>
</html>
```

HTML: Create 4 equal quadrants and place numbers 1,2,3,4 in each quadrant in centre.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Four Equal Quadrants</title>
  <style>
    .container {
      display: grid;
      grid-template-columns: 1fr 1fr;
      grid-template-rows: 1fr 1fr;
      width: 80vmin;
      height: 80vmin;
      gap: 10px;
    }
    .quadrant {
      display: flex;              // Important to show the text in center
      justify-content: center;    //
      align-items: center;        //
      background-color: #007bff;
      color: white;
      font-size: 2rem;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="quadrant">1</div>
    <div class="quadrant">2</div>
    <div class="quadrant">3</div>
    <div class="quadrant">4</div>
  </div>
</body>
</html>
```

Code: Write a program to get nth Fibonacci series.

```
function nthFibonacci(n){
    // Base case: if n is 0 or 1, return n
    if (n <= 1) {
        return n;
    }
    // Recursive case: sum of the two preceding Fibonacci
    // numbers
    return nthFibonacci(n - 1) + nthFibonacci(n - 2);
}
```

The code part started with some question about the script html tag.

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

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

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

<script type="module">
  // JavaScript module code here
</script>

<script src="legacy-script.js" nomodule></script>

Scripts can be placed in the <head> or <body> sections of an HTML document.
```

Last part was javascript, it was to write a deepEquality function.

```
function deepEqual(obj1, obj2) {
  // Check for strict equality first
  if (obj1 === obj2) return true;

  // Check if both arguments are objects
  if (typeof obj1 !== 'object' || typeof obj2 !== 'object' || obj1 === null
|| obj2 === null) {
    return false;
  }

  // Get the keys of both objects
  const keys1 = Object.keys(obj1);
  const keys2 = Object.keys(obj2);

  // Check if both objects have the same number of keys
  if (keys1.length !== keys2.length) return false;

  // Check if all keys and their values are equal
  for (let key of keys1) {
    if (!keys2.includes(key) || !deepEqual(obj1[key], obj2[key])) {
      return false;
    }
  }

  return true;
}
```

What's this

```
In JavaScript, the `this` keyword refers to the context in which a function
is executed. Its value is determined by how the function is called, not
where it's defined.
```

```
apply, bind, call
```

What's event loop, then explain Promise

```
In JavaScript, the event loop is a fundamental mechanism that manages the
execution of code, handling of events, and execution of queued tasks. It
enables asynchronous operations, allowing JavaScript to perform non-
blocking tasks despite being single-threaded.

How the Event Loop Works:

  Call Stack: This is where the JavaScript engine keeps track of function
calls. When a function is invoked, it's added to the stack, and when it
returns, it's removed.

  Web APIs: Functions like setTimeout, DOM events, and AJAX calls are
handled by the browser's Web APIs, which operate outside the call stack.

  Callback Queue: Once Web APIs complete their tasks, they push their
callbacks to the callback queue.

  Event Loop: The event loop continuously checks the call stack and the
callback queue. If the call stack is empty, it pushes the first callback
from the queue to the stack for execution.

A Promise in JavaScript represents the eventual completion (or failure) of
an asynchronous operation and its resulting value.
Promises provide a cleaner and more manageable way to handle asynchronous
code compared to traditional callbacks.

States of a Promise:

  Pending: Initial state; neither fulfilled nor rejected.

  Fulfilled: Operation completed successfully.

  Rejected: Operation failed.

Promises interact with the event loop through microtasks
```

what's the issue/problem been solved by Promise

```
By addressing these issues, Promises provide a more robust and maintainable
approach to handling asynchronous operations in JavaScript

1. Callback Hell (Pyramid of Doom)
  When multiple asynchronous operations are nested within callbacks, the
```

```
code structure can become deeply indented and difficult to read and
maintain.

  doSomething(function(result) {
    doSomethingElse(result, function(newResult) {
      doThirdThing(newResult, function(finalResult) {
        console.log(finalResult);
      });
    });
  });

  =>
  doSomething()
    .then(result => doSomethingElse(result))
    .then(newResult => doThirdThing(newResult))
    .then(finalResult => console.log(finalResult))
    .catch(error => console.error(error));

2. Error Handling
  asyncOperation()
    .then(result => processResult(result))
    .catch(error => handleError(error));
```

What's SPA (Single Page Application)

pros and cons

```
Pros:
  Improved User Experience
  Reduced Server Load
  Enhanced Performance
  Simplified Development

Cons:
  SEO Challenges
  Initial Load Time
  Browser Compatibility: SPAs rely heavily on JavaScript; users with
disabled JavaScript or older browsers may experience issues
  Complexity in State Management
```

what's the difference between normal webpages?

```
Navigation: SPAs provide dynamic content updates without full page reloads,
while MPAs require a full reload for each page navigation.

Performance: SPAs may have a slower initial load but offer faster
subsequent interactions. MPAs may have consistent load times but can be
slower overall due to repeated server requests.
```

```
Development Approach: SPAs often require a more complex setup with
JavaScript frameworks, whereas MPAs can be developed using traditional
server-side rendering techniques.

SEO and Accessibility: MPAs are typically more accessible to search engines
and users with disabilities, while SPAs may require additional
considerations to achieve similar levels of accessibility and SEO
performance.
```

how to implement FE-side routing

```
1. Hash-Based Routing:
Utilizes the URL fragment identifier (the part after # hashtag) to manage
routes. Changes to the hash don't trigger full page reloads, making it
suitable for SPAs.

<body>
  <nav>
    <a href="#home">Home</a>
    <a href="#about">About</a>
    <a href="#contact">Contact</a>
  </nav>
  <div id="content"></div>

  <script>
    function renderPage() {
      const route = window.location.hash.substring(1);
      const content = document.getElementById('content');
      switch (route) {
        case 'home':
          content.innerHTML = '<h1>Home Page</h1>';
          break;
        case 'about':
          content.innerHTML = '<h1>About Page</h1>';
          break;
        case 'contact':
          content.innerHTML = '<h1>Contact Page</h1>';
          break;
        default:
          content.innerHTML = '<h1>Welcome!</h1>';
      }
    }

    window.addEventListener('hashchange', renderPage);
    window.addEventListener('load', renderPage);
  </script>
</body>

2. History API-Based Routing:
Leverages the HTML5 History API (pushState and replaceState) to manipulate
```

```
the browser's session history, enabling clean URLs without the hash symbol.
<body>
  <nav>
    <a href="/home" data-link>Home</a>
    <a href="/about" data-link>About</a>
    <a href="/contact" data-link>Contact</a>
  </nav>
  <div id="content"></div>

  <script>
    function navigate(event) {
      event.preventDefault();
      const path = event.target.getAttribute('href');
      window.history.pushState({}, '', path);
      renderPage();
    }

    function renderPage() {
      const path = window.location.pathname;
      const content = document.getElementById('content');
      switch (path) {
        case '/home':
          content.innerHTML = '<h1>Home Page</h1>';
          break;
        case '/about':
          content.innerHTML = '<h1>About Page</h1>';
          break;
        case '/contact':
          content.innerHTML = '<h1>Contact Page</h1>';
          break;
        default:
          content.innerHTML = '<h1>Welcome!</h1>';
      }
    }

    window.addEventListener('popstate', renderPage);
    document.querySelectorAll('a[data-link]').forEach(link => {
      link.addEventListener('click', navigate);
    });
    window.addEventListener('load', renderPage);
  </script>
</body>

3. vue router
```

How they implement Vue reactivity system?

```
In Vue2, Utilized Object.defineProperty to define getters and setters on
each property of an object, enabling the detection of changes.
```

```
Vue 3: Transitioned to using JavaScript's Proxy object to create reactive
proxies for entire objects.
```

How to do performace tunning? share some experience

```
1. Minimize HTTP Requests
2. Implement Caching: browser and server-side caching
3. Optimize Images: Lazy load
4. Use Content Delivery Networks (CDNs)
5. Minify and Compress Resources: Minify CSS, JavaScript, and HTML files to
reduce their size.
6. Monitor Performance Metrics
7. Utilize Asynchronous Operations
```

key factors in web performance

```
Web Vitals is an initiative by Google that provides unified guidance for
quality signals
essential to delivering a great user experience on the web.

1. Largest Contentful Paint (LCP):
   Measures loading performance. To provide a good user experience, LCP
should occur within 2.5 seconds of when the page first starts loading.

2. Interaction to Next Paint (INP):
  Measures responsiveness. To provide a good user experience, pages should
have an INP of 200 milliseconds or less.

3. Cumulative Layout Shift (CLS):
  Measures visual stability. To provide a good user experience, pages
should maintain a CLS of 0.1 or less.


To measure and report these metrics, developers can use various tools and
libraries.
For instance, the `web-vitals` JavaScript library is a small, production-
ready wrapper around the underlying web APIs that measures each metric
in a way that accurately matches how they're reported by all the Google
tools.
```

explain repaint/reflow

```
Repaint:
  A repaint occurs when changes are made to an element's appearance that do
```

```
not affect its layout.
  指网页渲染引擎根据显示属性（如颜色、文字大小等）重新绘制页面元素，不影响元素的位置和尺寸。例如：改变  outline、visibility、color、background-color等。

Reflow:
  A reflow, also known as layout, happens when changes to the DOM affect the document's structure, requiring the browser to recalculate the positions and sizes of elements.
  指网页渲染引擎根据元素的尺寸、位置和显示属性来重新计算页面的排版和布局，是网页渲染过程中的一个重要步骤。例如：改变  width、height、font-size  等
```

how to reduce webpack bundle size?

```
1. 启用生产模式（Production Mode）： 确保在生产环境中使用 Webpack 的生产模式，它会自动进行代码压缩和优化。
webpack --mode production

2. 代码分割（Code Splitting）： 将代码拆分为更小的块，实现按需加载，减少初始加载时间。可以在 Webpack 配置中使用 SplitChunksPlugin
module.exports = {
  // ...
  optimization: {
    splitChunks: {
      chunks: 'all',
    },
  },
};

3. 分析打包内容： 使用如 Webpack Bundle Analyzer 等工具，可视化并了解打包内容，识别需要优化或移除的大型模块或依赖。
const { BundleAnalyzerPlugin } = require('webpack-bundle-analyzer');

module.exports = {
  // ...
  plugins: [
    new BundleAnalyzerPlugin(),
  ],
};
```

Do you know about dynamic import?

```
async function loadModule() {
  try {
    const module = await import('./module.js');
    module.exportedFunction();
  } catch (error) {
    console.error('Error loading module:', error);
```

```
  }
}

for better:
Lazy Loading
Code splitting
Asyns Loading
Error handling
```

What's the difference of async and defer in