

1. What we're doing today

Week 5 how to write your own promises
writing promise class yourself (hard)
async await syntax
practise

2.

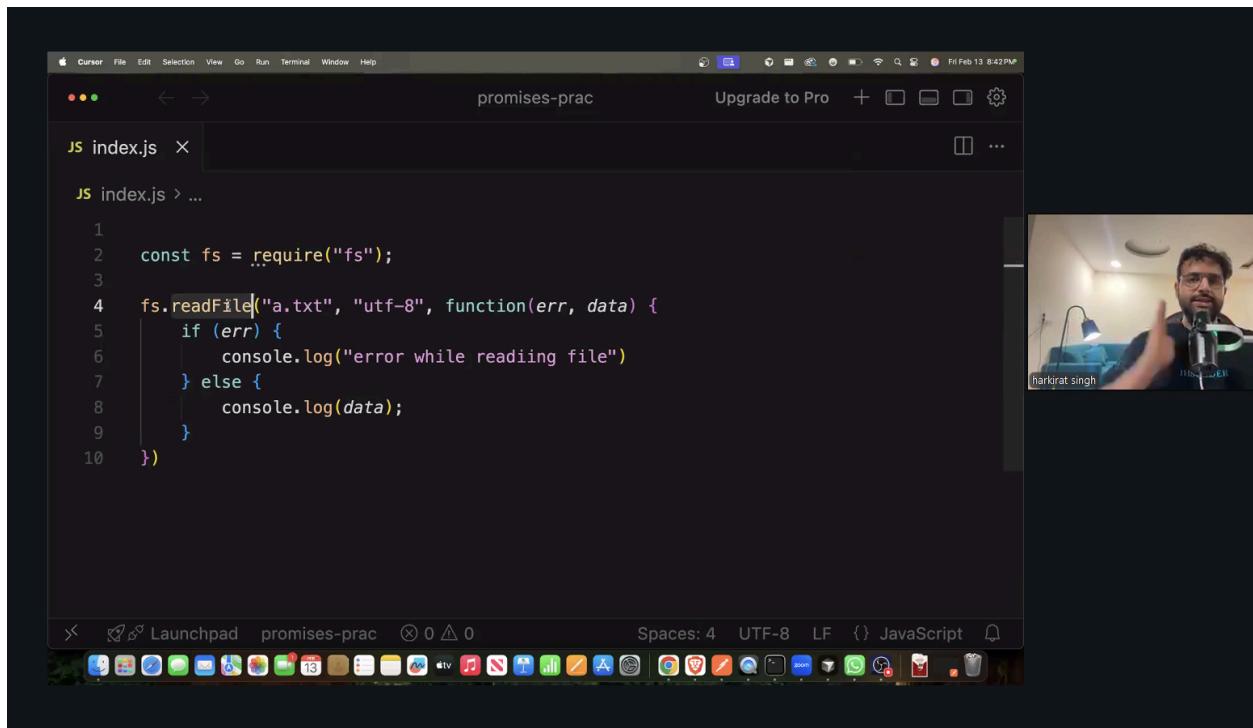
What is a promise

A **Promise** in JavaScript is an object that represents the eventual result of an asynchronous operation — either a success (**resolved**) or a failure (**rejected**).

It's commonly used for things like:

- Fetching data from an API
- Reading files
- Timers (`setTimeout`)
- Database requests

3. Callback function



The screenshot shows a Mac desktop with a terminal window open. The terminal window has tabs for 'index.js' and 'promises-prac'. The code in 'index.js' is:

```
1 const fs = require("fs");
2
3 fs.readFile("a.txt", "utf-8", function(err, data) {
4     if (err) {
5         console.log("error while reading file")
6     } else {
7         console.log(data);
8     }
9 })
10 }
```

On the right side of the screen, there is a video call interface. A person with a beard and glasses, identified as 'harkirat singh', is visible. The video call window has a small 'HIGHLIGHT' button in the bottom right corner. The desktop background shows a blurred view of a room with a blue sofa and a lamp. The system tray at the bottom shows various icons and the date/time: 'Fri Feb 13 8:42PM'.

4. Promise has 3 state

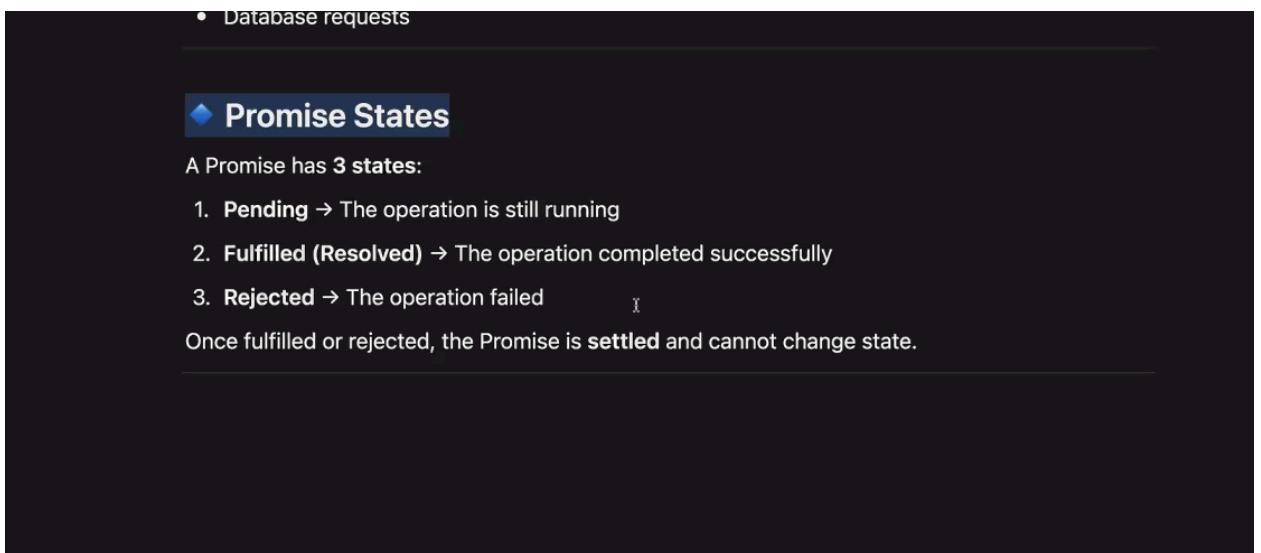
- Database requests

Promise States

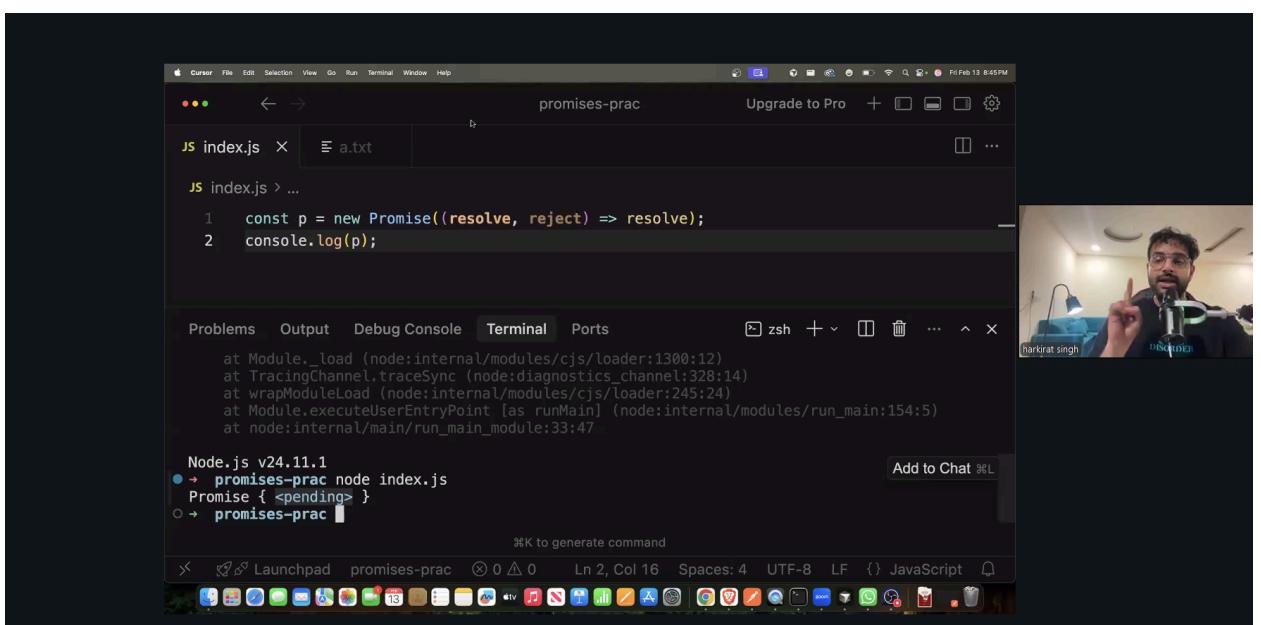
A Promise has 3 states:

1. Pending → The operation is still running
2. Fulfilled (Resolved) → The operation completed successfully
3. Rejected → The operation failed

Once fulfilled or rejected, the Promise is **settled** and cannot change state.



5.



The screenshot shows a developer's workspace in VS Code. In the top-left, there are tabs for 'index.js' and 'a.txt'. The 'Terminal' tab is active, displaying the following Node.js code and its execution:

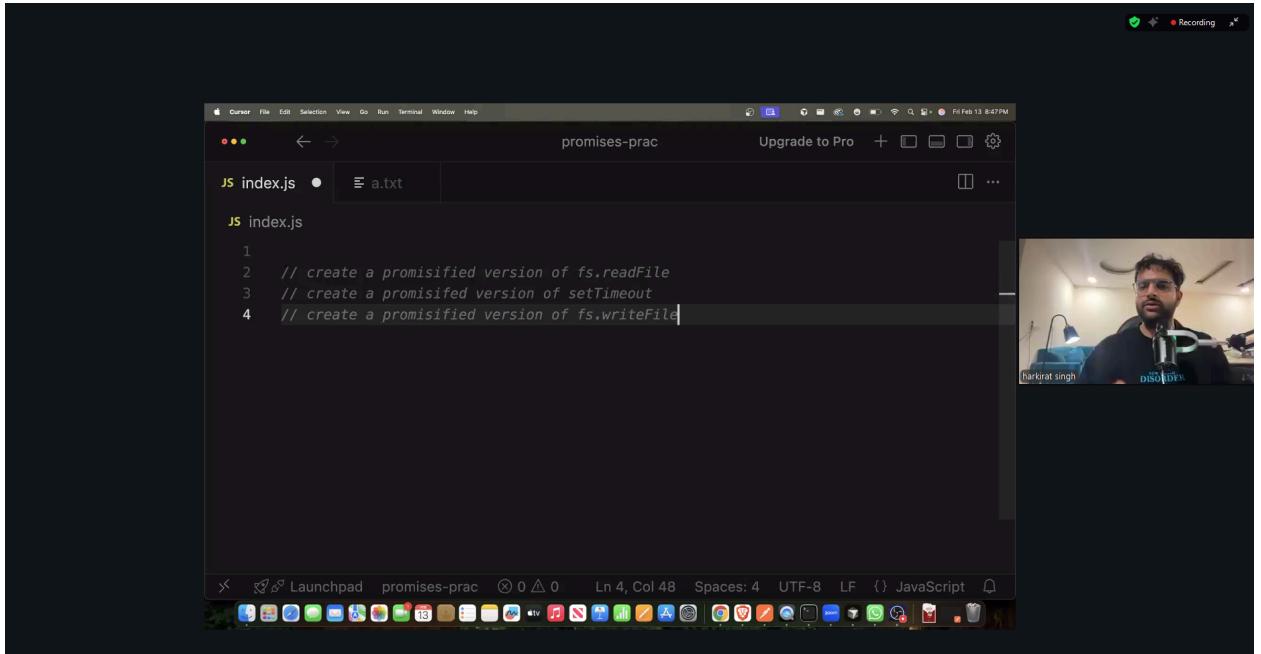
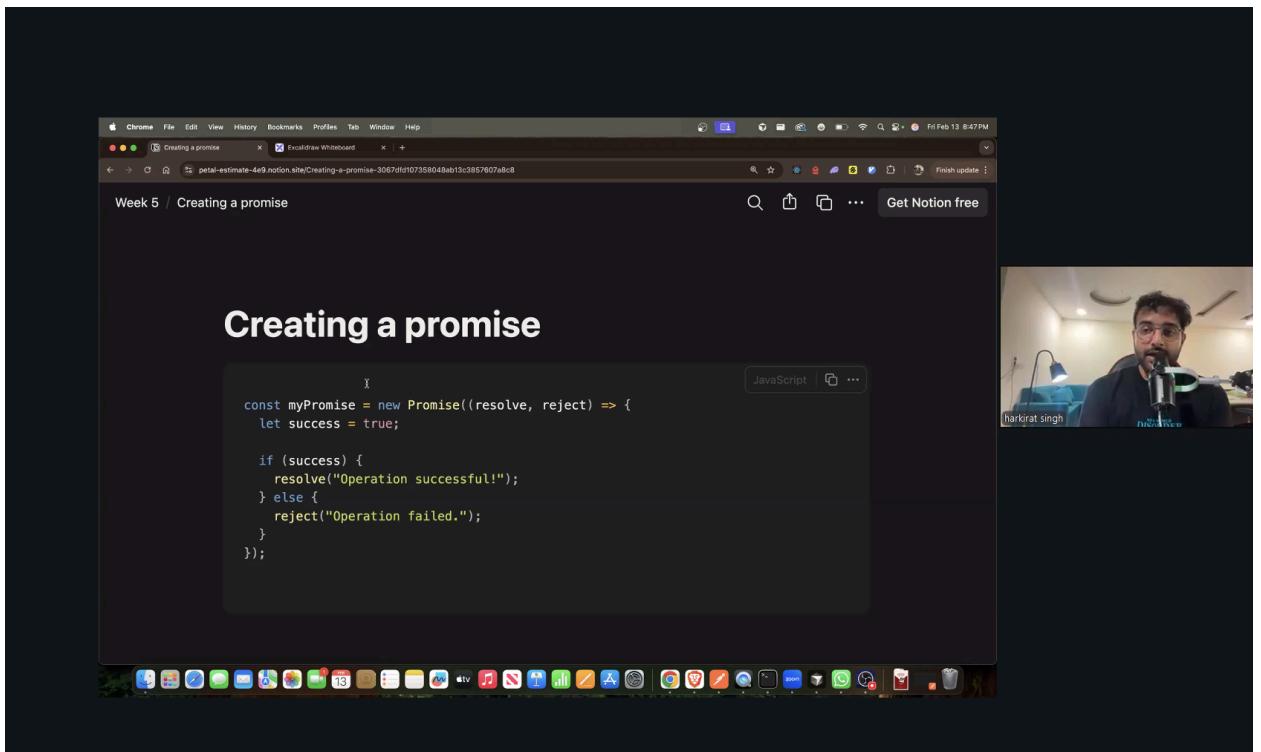
```
JS index.js > ...
1 const p = new Promise((resolve, reject) => resolve);
2 console.log(p);
```

Below the code, the terminal shows the output of the Node.js process:

```
Node.js v24.11.1
● → promises-prac node index.js
Promise { <pending> }
```

The developer's video feed is visible in the top-right corner, and they are pointing upwards while speaking.

6. Let's create promise



8. Create a promisified version of fs.readFile



A screenshot of a Mac OS X desktop showing a terminal window titled "index.js". The code in the terminal is:

```
JS index.js > ⚡ fsReadFilePromise
1 // create a promisified version of fs.readFile
2 // create a promisified version of setTimeout
3 // create a promisified version of fs.writeFile
4
5
6 // on top of fs.readFile
7 function fsReadFilePromise(fileName, encoding) {
8     return new Promise(function(resolve, reject) {
9         ...
10    });
11 }
12
13 fsReadFilePromise("a.txt", "utf-8")
14 .then(function(data) {
15     console.log(data);
16 })

```

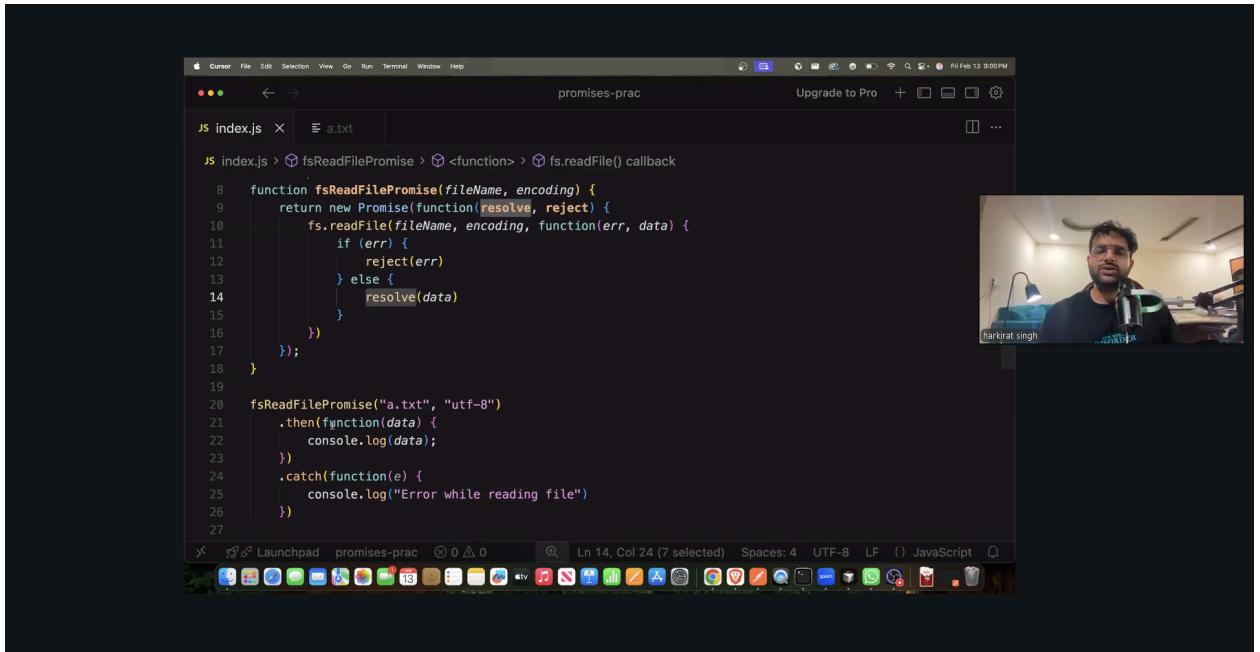
The status bar at the bottom shows "Spaces: 4" and "JavaScript". A video feed of a man with glasses and a black shirt is visible in the top right corner.



A screenshot of a Mac OS X desktop showing a terminal window titled "index.js". The code in the terminal is:

```
JS index.js > ⚡ fsReadFilePromise
7 function fsReadFilePromise(fileName, encoding) {
8     return new Promise(function(resolve, reject) {
9         ...
10    });
11 }
12
13 class Promise {
14     constructor(fn) {
15
16         fn(function() {
17
18         }, function() {
19
20         });
21     }
22 }
```

The status bar at the bottom shows "Spaces: 4" and "JavaScript". A video feed of a man with glasses and a black shirt is visible in the top right corner.



A screenshot of a Mac desktop. In the top right corner, there is a video call window showing a man with a beard and dark hair, wearing a black t-shirt. The video call has a timestamp of "Fri Feb 13 9:00PM" and a name "harkirat singh". In the center, there is a code editor window titled "index.js" with the file path "promises-prac/a.txt". The code is as follows:

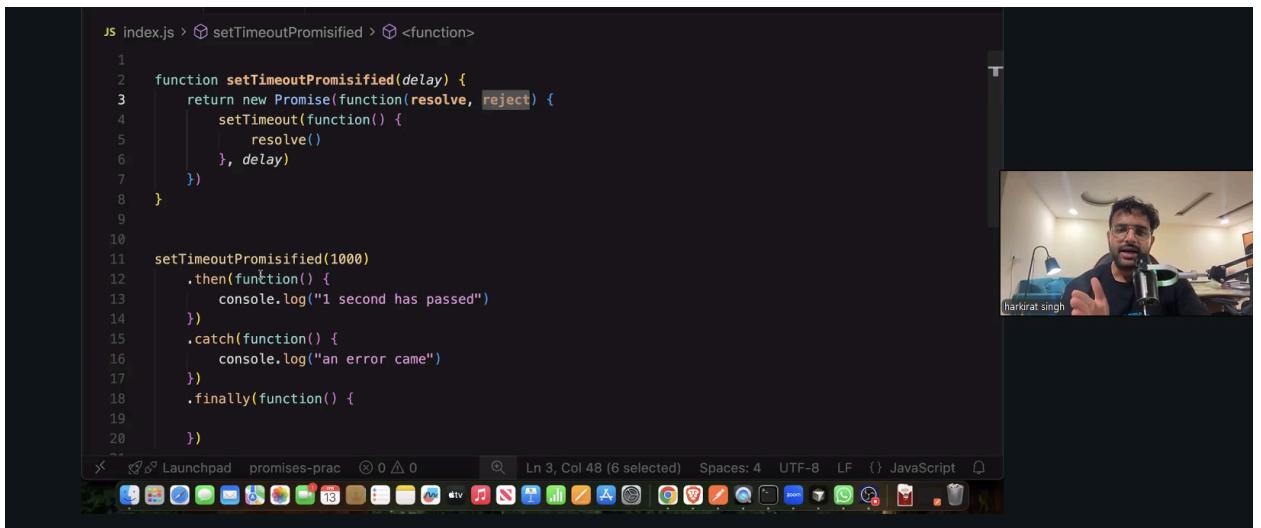
```
JS index.js > ⓘ fs.readFilePromise > ⓘ <function> > ⓘ fs.readFile() callback
  8   function fsReadFilePromise(fileName, encoding) {
  9     return new Promise(function(resolve, reject) {
10       fs.readFile(fileName, encoding, function(err, data) {
11         if (err) {
12           reject(err)
13         } else {
14           resolve(data)
15         }
16       })
17     });
18   }
19
20   fsReadFilePromise("a.txt", "utf-8")
21   .then(function(data) {
22     console.log(data);
23   })
24   .catch(function(e) {
25     console.log("Error while reading file")
26   })
27
```

The status bar at the bottom shows "Ln 14, Col 24 (7 selected)" and "Spaces: 4". The Mac dock is visible at the bottom with various application icons.

9. Create a promisified version of setTimeout

```
26
27
28
29   function setTimeoutPromisified(delay) {
30     return new Promise(function(resolve, reject) {
31       setTimeout(function() {
32         resolve()
33       }, delay)
34     })
35   }
36
37
38   setTimeoutPromisified(10000)
39   .then(function() {
40     console.log("1 second has passed")
41   })
42
```

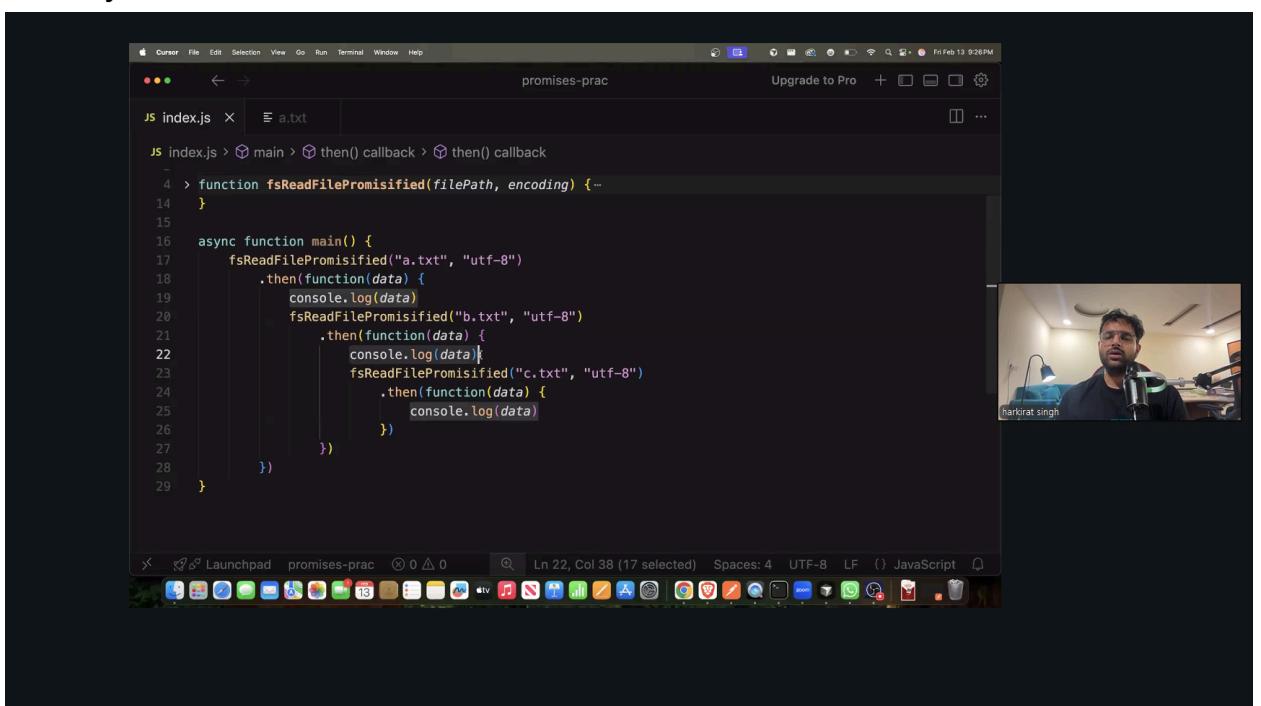
10. finally concept in Promise



```
JS index.js > ⚡ setTimeoutPromisified > ⚡ <function>
1
2   function setTimeoutPromisified(delay) {
3     return new Promise(function(resolve, reject) {
4       setTimeout(function() {
5         resolve()
6       }, delay)
7     })
8   }
9
10 setTimeoutPromisified(1000)
11   .then(function() {
12     console.log("1 second has passed")
13   })
14   .catch(function() {
15     console.log("an error came")
16   })
17   .finally(function() {
18
19   })
20 }
```

The screenshot shows a Mac desktop with a terminal window open. The terminal window displays a JavaScript file named 'index.js' containing code for a promise-based timeout. The code defines a function 'setTimeoutPromisified' that returns a promise. This promise is then chained through '.then()', '.catch()', and '.finally()' methods. A video call overlay of a man with glasses and a beard is visible in the top right corner.

11. Async/Await



```
JS index.js > ⚡ main > ⚡ then() callback > ⚡ then() callback
1
2   async function main() {
3     fsReadFilePromisified("a.txt", "utf-8")
4       .then(function(data) {
5         console.log(data)
6         fsReadFilePromisified("b.txt", "utf-8")
7           .then(function(data) {
8             console.log(data)
9             fsReadFilePromisified("c.txt", "utf-8")
10               .then(function(data) {
11                 console.log(data)
12               })
13             })
14           })
15         }
16       }
```

The screenshot shows a Mac desktop with a terminal window open. The terminal window displays a JavaScript file named 'index.js' containing code using the 'async/await' pattern. It defines an 'fsReadFilePromisified' function and then uses it within an 'async function main()'. Inside 'main()', three file reads are performed sequentially using 'await' and 'then()' callbacks. A video call overlay of a man with glasses and a beard is visible in the top right corner.



A screenshot of a Mac OS X desktop showing a terminal window and a video call overlay.

The terminal window title is "promises-prac". It contains the following code:

```
JS index.js > ⌘ main
1
2   const fs = require("fs");
3
4 > function fsReadFilePromisified(filePath, encoding) { ...
14 }
15
16   async function main() {
17     let file1Contents = await fsReadFilePromisified("a.txt", "utf-8");
18     let file2Contents = await fsReadFilePromisified("b.txt", "utf-8");
19     let file3Contents = await fsReadFilePromisified("c.txt", "utf-8");
20
21     console.log(file1Contents);
22     console.log(file2Contents);
23     console.log(file3Contents);
24   }
25
26   main();
```

The status bar at the bottom shows "Ln 24, Col 2" and "Spaces: 4".

12. Difference between Async and Sync



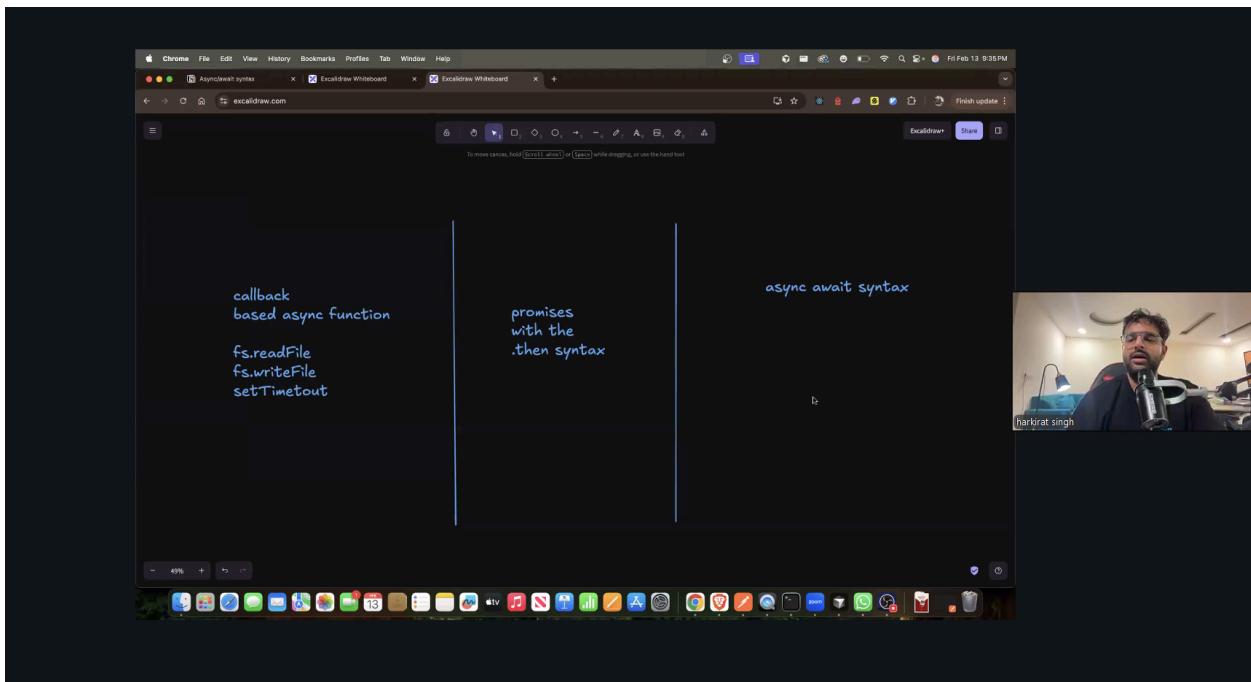
```
JS index.js > const fs = require('fs');
3
4 > function fsReadFilePromisified(filePath, encoding) {-
14   }
15
16   function main() {
17     let file1Contents = fs.readFileSync("a.txt", "utf-8");
18     let file2Contents = fs.readFileSync("b.txt", "utf-8")
19     let file3Contents = fs.readFileSync("c.txt", "utf-8")
20
21     console.log(file1Contents);
22     console.log(file2Contents);
23     console.log(file3Contents);
24   }
25
26   main();
27
```

JS index.js > ...

```
14   }
15
16   async function main() {
17     let file1Contents = await fsReadFilePromisified("a.txt", "utf-8");
18     let file2Contents = await fsReadFilePromisified("b.txt", "utf-8")
19     let file3Contents = await fsReadFilePromisified("c.txt", "utf-8")
20
21     console.log(file1Contents);
22     console.log(file2Contents);
23     console.log(file3Contents);
24   }
25
26   main();
27   console.log("hi")
28   console.log("help");
29
```



13. Difference between callback, promise, async/await



Defining your own async function

Q: Write a function that

1. Reads the contents of a file
2. Trims the extra space from the left and right
3. Writes it back to the file

Synchronous

A screenshot of a Mac desktop. On the left, a terminal window titled "promises-prac" shows a script named "index.js" with the following code:

```
JS index.js > ...
1 const fs = require("fs");
2
3 let contents = fs.readFileSync("a.txt", "utf-8");
4 const trimmedContents = contents.trim();
5 fs.writeFileSync("a.txt", trimmedContents);
```

The terminal status bar indicates "Ln 5, Col 44". On the right, a Slack message list shows several messages from different users:

- Anurag Bansal to Everyone: AB Y
- Dhiruv Dhuria to Everyone: DD yes
- Krishna Chandu to Everyone: KC S
- Akhchat Thakur to Everyone: AT nono
- Prathamesh Sutar to Everyone: PS no
- Kamlesh Mandloi to Everyone: KM yes
- Opendra Kumar to Everyone: OK Valentine ki raat or andheraa offoooo - male hi male
- Who can see your messages? Recording

At the bottom right, there is a video feed of a person with the name "harkirat singh" displayed.

Approach #1(callback based sync function calls)

Approach #2(callback based async function calls)

A screenshot of a Mac desktop showing a browser window with two tabs side-by-side. Both tabs are titled "Excalidraw Whiteboard".

The left tab is labeled "Approach #1 (callback based, sync fn calls)" and contains the following code:

```
JS index.js > ...
1 const fs = require("fs");
2
3 function cleanFileSync(filePath, cb) {
4     const contents = fs.readFileSync(filePath, "utf-8");
5     const trimmedContents = contents.trim();
6     fs.writeFileSync("a.txt", trimmedContents);
7 }
8
9 cleanFileSync("a.txt");
```

The right tab is labeled "Approach #2 (callback based, async function calls)" and contains the following code:

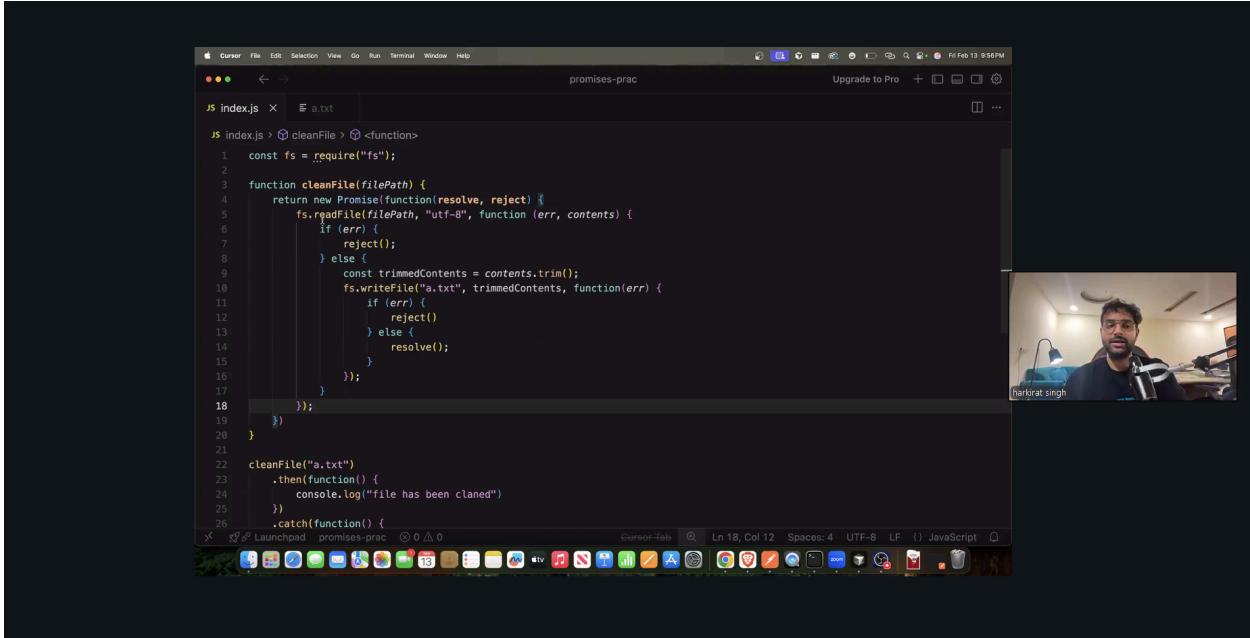
```
const fs = require("fs");

function cleanFile(filePath, cb) {
    fs.readFile(filePath, "utf-8", function (err, contents) {
        const trimmedContents = contents.trim();
        fs.writeFileSync("a.txt", trimmedContents, function() {
            cb();
        });
    });
}

cleanFile("a.txt", function() {
    console.log("done cleaning a.txt");
});
```

At the bottom right, there is a video feed of a person with the name "harkirat singh" displayed.

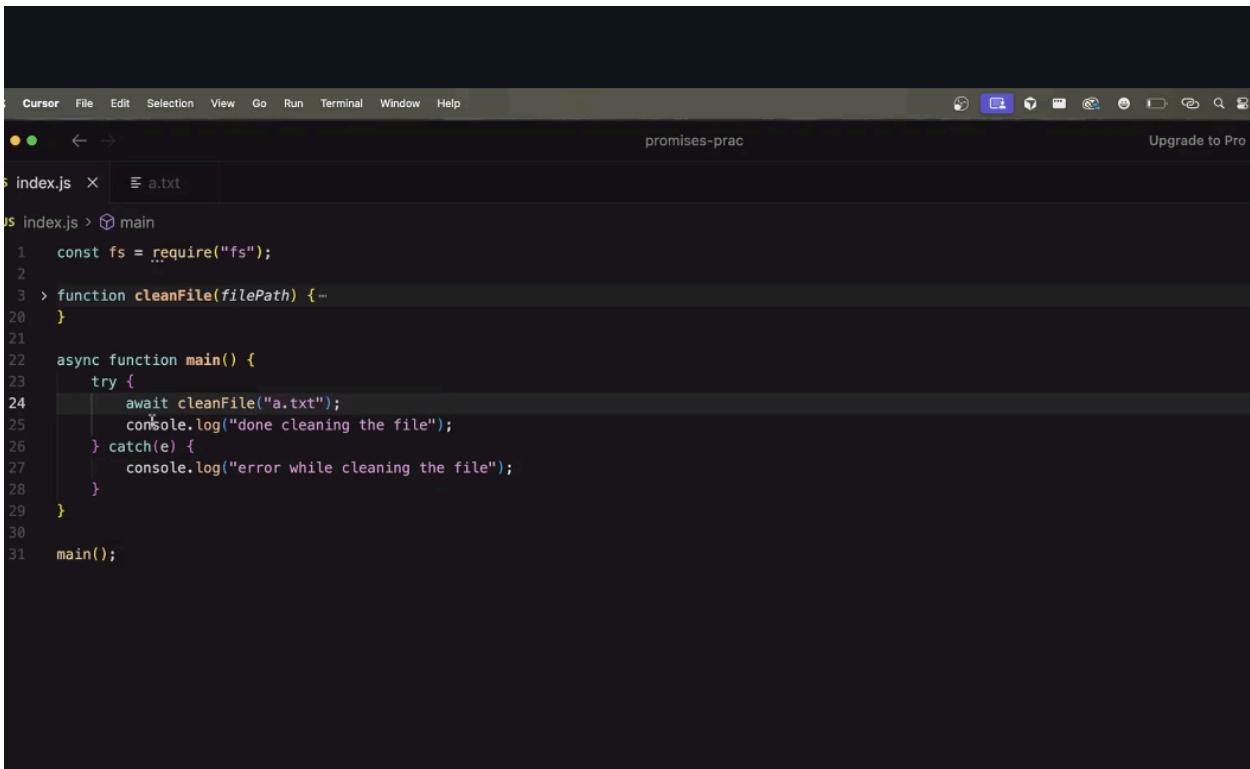
Approach #3(promise based, no async/await)



```
JS index.js > ⚡ cleanFile : ⚡ <function>
1  const fs = require("fs");
2
3  function cleanFile(filePath) {
4    return new Promise(function(resolve, reject) {
5      fs.readFile(filePath, "utf-8", function (err, contents) {
6        if (err) {
7          reject();
8        } else {
9          const trimmedContents = contents.trim();
10         fs.writeFile("a.txt", trimmedContents, function(err) {
11           if (err) {
12             reject();
13           } else {
14             resolve();
15           }
16         });
17       }
18     });
19   }
20
21   cleanFile("a.txt")
22     .then(function() {
23       console.log("file has been cleaned")
24     })
25     .catch(function() {
26

```

Approach #4(Async/Await)



```
JS index.js > ⚡ main
1  const fs = require("fs");
2
3  > function cleanFile(filePath) { ...
20  }
21
22  async function main() {
23    try {
24      await cleanFile("a.txt");
25      console.log("done cleaning the file");
26    } catch(e) {
27      console.log("error while cleaning the file");
28    }
29  }
30
31  main();
```

A screenshot of a Mac desktop environment. At the top, there's a dark-themed window for a code editor showing JavaScript code. The code defines a function `cleanManyFiles` that uses promises to clean multiple files. Below the code editor is a standard Mac OS X dock with various application icons. At the bottom, there's a dark-themed terminal window with a small video thumbnail of a person speaking.

```
JS index.js > a1.txt
JS index.js > cleanManyFiles > <function>
-- 
21
22
23     async function cleanManyFiles(prefix) {
24         await cleanFile(prefix + "1.txt");
25         await cleanFile(prefix + "2.txt");
26         await cleanFile(prefix + "3.txt");
27     }
28
29     function cleanManyFiles(prefix) {
30         return new Promise(function(resolve, reject) {
31             cleanFile(prefix + "1.txt")
32                 .then(function() {
33                     cleanFile(prefix + "2.txt")
34                         .then(function() {
35                             cleanFile(prefix + "3.txt")
36                                 .then(function() {
37                                     resolve()
38                                 })
39                         })
40                     })
41                 .catch(function() {
42                     reject()
43                 })
44             })
45     }

```

A screenshot of a Mac desktop environment, similar to the one above. This view shows a more complex version of the code. It includes imports for `fs` and `Promise`, and adds logging statements to the promise chain. The code editor interface is identical to the first screenshot, showing the same file structure and code content.

```
JS index.js > a1.txt
JS index.js > cleanManyFiles
1     const fs = require("fs");
2
3     > function cleanFile(filePath) { ...
20     }
21
22
23     async function cleanManyFiles(prefix) {
24         // ⌘L to chat, ⌘K to generate
25         await cleanFile(prefix + "1.txt");
26         await cleanFile(prefix + "2.txt");
27         await cleanFile(prefix + "3.txt");
28     }
29
30
31     // write a promisified function that takes a file prefix as an input (a)
32     // and cleans ({prefix}1.txt, {prefix}2.txt, {prefix}3.txt
33
34     cleanManyFiles("a")
35         .then(function() {
36             console.log("all 3 files have been cleaned");
37         })
38         .catch(function() {
39             console.log("error while cleaning these 3 files")
40         })

```