

JAVASCRIPT DAY 19

1) Callbak Hell

Cheet Sheat For Interview Preparation

1. What is Callback Hell?

Ans: Callback Hell occurs when multiple nested callbacks make code messy and hard to read.

2. Why is Callback Hell called "Pyramid of Doom"?

Ans: The deeply nested structure of callbacks forms a pyramid-like shape, making the code look cluttered.

3. What causes Callback Hell?

Ans: It happens when asynchronous tasks depend on each other and are handled using nested callbacks.

4. What are the problems with Callback Hell?

Ans: Difficult to read Hard to debug Poor error handling Tough to maintain

5. How can you avoid Callback Hell?

Ans: Use Promises to chain tasks. Use Async/Await for cleaner, readable code. Break code into smaller functions.

6. What is a callback?

Ans: A function passed as an argument to another function, executed after the first function completes.

7. How do Promises solve Callback Hell?

Ans: They allow chaining of tasks instead of nesting, making the code flatter and easier to read.

8. What does async do in JavaScript?

Ans: Marks a function to handle asynchronous tasks, allowing the use of await.

9. What does await do in JavaScript?

Ans: It pauses the execution of an async function until a Promise resolves or rejects.

10. What is the catch() method in Promises used for?

Ans: It handles errors that occur during the execution of a Promise chain

FAQ'S OF CALLBACK HELL

1. What is Callback Hell?

Ans: Callback Hell is a situation where too many nested callbacks make the code difficult to read, debug, and maintain.

2. Why does Callback Hell happen?

Ans: It happens when multiple asynchronous tasks depend on each other, and each task uses a callback function.

3. Why is Callback Hell bad?

Ans: It makes the code messy, hard to understand, and challenging to fix if errors occur.

4. How can Callback Hell be avoided?

Ans: By using alternatives like Promises or Async/Await.

5. What is a callback?

Ans: A callback is a function passed to another function to be executed later, often after an asynchronous operation completes.

6. What are the disadvantages of Callback Hell?

Ans: Poor readability, difficulty in debugging, and reduced maintainability.

7. What is a "Pyramid of Doom"?

Ans: It's another name for Callback Hell, describing the deeply indented structure of nested callbacks.

8. What is the role of Promises in solving Callback Hell?

Ans: Promises allow chaining asynchronous tasks in a flat and readable manner instead of nesting.

9. How does Async/Await help solve Callback Hell?

Ans: Async/Await makes asynchronous code look like synchronous code, improving readability and simplicity.

10. Is Callback Hell only a problem in JavaScript?

Ans: While it is most commonly associated with JavaScript, similar issues can occur in other programming languages that support asynchronous callbacks.

MCQ'S OF CALLBACK HELL

1. **What does Callback Hell refer to?**
 - b) Deeply nested callbacks that are hard to read
2. **What is another name for Callback Hell?**
 - b) Pyramid of Doom
3. **Which method is NOT a solution for Callback Hell?**
 - c) Nested Callbacks
4. **What does the async keyword do in JavaScript?**
 - b) Marks a function to handle asynchronous operations
5. **How do Promises help in avoiding Callback Hell?**
 - b) By flattening nested callbacks with chaining

6. What does the await keyword do in JavaScript?

b) Pauses execution until a Promise resolves

7. Which of the following is an example of Callback Hell?

```
task1(function() {  
  task2(function() {  
    task3(function() {  
      console.log("Done");  
    });  
  });  
});
```

a) Yes

8. What is a common issue in Callback Hell?

b) Errors are not handled properly

9. Which of these is the cleanest way to handle asynchronous operations?

a) Using Promises

10. What keyword is used to define a function that can use await?

c) async

PROMISES TOPIC ASSIGNMENT

CHEET SHEAT OF PROMISES

1. What is a Promise in JavaScript?

Ans: Promise is an object representing the eventual completion or failure of an asynchronous operation.

2. What are the three states of a Promise?

Ans: Pending o Fulfilled o Rejected

3. Why are Promises used?

Ans: To handle asynchronous operations in a more structured and readable way than callbacks.

4. What is the syntax to create a Promise?

Ans: let promise = new Promise((resolve, reject) => { // async code });

5. How do you consume a Promise?

Ans: Using .then() for success. o Using .catch() for errors. o Using .finally() for final actions.

6. How do you handle errors in Promises?

Ans: Using .catch() or try...catch with async/await.

7. What happens if .catch() is not used for a rejected Promise?

Ans: Unhandled promise rejections may throw an error or cause warnings in modern JavaScript runtimes.

8. What is the purpose of .finally()?

Ans: Executes a callback regardless of whether the promise is fulfilled or rejected. Promise Methods

9. What does Promise.all() do?

Ans: It resolves when all promises resolve, or rejects if any promise rejects.

10. What does Promise.race() do?

Ans: Resolves or rejects as soon as the first promise settles (fulfilled or rejected).

11. What is Promise.allSettled()?

Ans: Returns an array of objects with the status (fulfilled or rejected) and value/reason of all promises, after they all settle.

12. What does Promise.any() do?

Ans: Resolves with the first fulfilled promise or rejects if all promises are rejected.

13. What is the difference between Promise.all() and Promise.allSettled()?

Ans: Promise.all() fails fast, rejecting as soon as one promise rejects.

- Promise.allSettled() waits for all promises to settle and provides their statuses and results.

14. What is the output of Promise.any([])?

Ans: It resolves with an AggregateError because there are no promises to resolve.

15. How does Promise.race() differ from Promise.any()?

Ans: Promise.race() resolves/rejects with the first settled promise (fulfilled or rejected).

- Promise.any() resolves with the first fulfilled promise, ignoring rejections.

16. How do you chain multiple .then() methods?

Ans: let promise = new Promise((resolve) => resolve(10));

promise .then((value) => value * 2) .then((value) => value + 5) .then(console.log);

// Output: 25

17. What does Promise.resolve(value) do?

Ans: Creates a promise that is immediately resolved with the provided value.

18. What does Promise.reject(reason) do?

Ans: Creates a promise that is immediately rejected with the provided reason.

19. How does async/await work with Promises?

- `async` makes a function return a promise.
- `await` pauses the function execution until the promise resolves.

20. How do you convert a callback function into a Promise?

Ans: function `asyncTask(callback)`

```
{ setTimeout(() => callback(null, "Done!"), 1000); }
```

```
const promisifiedTask = () => new Promise((resolve, reject) => { asyncTask((err, result) => {  
  if (err) reject(err);  
  else resolve(result);  
}); });
```

```
promisifiedTask().then(console.log);
```

21. When should you use `Promise.all()`?

Ans: When you need all promises to resolve before continuing execution.

22. When should you use `Promise.race()`?

Ans: When you care only about the first settled promise, regardless of its outcome.

23. What happens if all promises in `Promise.all()` are fulfilled?

Ans: It resolves with an array of all resolved values.

24. What happens if one promise in `Promise.all()` rejects?

Ans: It immediately rejects with the rejection reason.

25. What is the use of `Promise.allSettled()` in APIs?

Ans: Useful when you want to gather results of multiple API calls without failing for partial errors.

26. How can you retry a failed Promise?

Ans: By chaining `.catch()` with another call to the function.

```
function fetchData() { return fetch("https://example.com/api"); }  
fetchData().catch(() => fetchData());
```

27. What is the output of this code?

```
Promise.any([ Promise.reject("Error 1"),  
Promise.reject("Error 2"), ]).catch((err) => console.log(err));
```

Answer: AggregateError: All promises were rejected

28. What does new `Promise()` require?

Ans: A callback function with two parameters: `resolve` and `reject`.

29. Why is `finally()` useful?

Ans: To perform cleanup tasks like closing a loader, regardless of the promise outcome.

30. What is the main advantage of Promises?

Ans: They simplify handling asynchronous operations and improve code readability compared to callbacks.

FAQ'S OF PROMISES TOPIC

1. What are Promises in JavaScript?

Promises provide a cleaner, more robust way to handle asynchronous operations compared to callbacks.

2. What are the three states of a Promise?

- **Pending:** The initial state, neither fulfilled nor rejected.
- **Fulfilled:** Operation completed successfully.
- **Rejected:** Operation failed.

3. How does the .then() method work in Promises?

It handles the result of a fulfilled promise and can return another promise for chaining.

4. What is the purpose of .catch() in Promises?

It handles errors or promise rejections.

5. What does .finally() do in Promises?

It executes a callback after the promise settles (fulfilled or rejected), for cleanup purposes.

6. How do resolve and reject work in a Promise?

- **resolve:** Marks a promise as successful and passes the result.
- **reject:** Marks a promise as failed and passes the error.

7. What is Promise.all() used for?

It runs multiple promises in parallel and resolves when all resolve or rejects if any fail.

8. How is Promise.race() different from Promise.all()?

- **Promise.race():** Resolves/rejects as soon as the first promise settles.
- **Promise.all():** Waits for all promises to settle.

9. What is Promise.allSettled()?

It waits for all promises to settle (fulfilled or rejected) and returns an array of their statuses and values.

10. What happens if all promises in Promise.any() reject?

It throws an **AggregateError** containing all rejection reasons.

11. How does async/await work with Promises?

- **async:** Declares a function that always returns a promise.
- **await:** Pauses execution until the awaited promise resolves.

12. Can you chain .then() calls in Promises?

Yes, for sequential execution or transforming results across multiple steps.

13. What happens if you omit a .catch() block?

Unhandled promise rejections may throw an error, depending on the environment.

14. How is error handling done with Promise.all()?

If any promise rejects, Promise.all() immediately rejects with the first error encountered.

15. What is the difference between Promise.any() and Promise.race()?

- **Promise.any()**: Resolves with the first successful promise (ignores rejections).
- **Promise.race()**: Resolves or rejects with the first settled promise (fulfilled or rejected).

MCQ'S OF PROMISES

Basic Questions

- 1. What does the then() method of a Promise handle?**
A) Fulfilled promises
- 2. What is the primary benefit of using Promises over callbacks?**
B) Avoiding callback hell
- 3. Which of the following methods can be used to handle errors in Promises?**
C) .catch()
- 4. What is the default state of a Promise?**
C) Pending
- 5. What does the reject function in a Promise signify?**
C) The promise has failed

Advanced Methods

- 6. What does Promise.all() return if one of the promises is rejected?**
B) The rejection reason of the first rejected promise
- 7. Which Promise method resolves with the fastest fulfilled or rejected promise?**
B) Promise.race()
- 8. What does Promise.allSettled() return?**
B) An array of objects with each promise's status and value/reason
- 9. What happens when all promises in Promise.any() are rejected?**
B) It throws an AggregateError.
- 10. Which method should be used when you want the result of only the first fulfilled promise?**
C) Promise.any()

Code-Based Questions**11. What will this code output?**

```
let p1 = Promise.resolve("Task 1 complete");  
let p2 = Promise.reject("Task 2 failed");  
Promise.all([p1, p2])  
  .then(results => console.log(results))  
  .catch(error => console.log(error));
```

B) "Task 2 failed"

12. What will Promise.race([p1, p2]) output if p1 resolves in 1 second and p2 rejects in 2 seconds?

B) The value of p1

13. How does Promise.allSettled() handle rejected promises?

C) Returns their reason in the output array

14. What is the difference between Promise.any() and Promise.allSettled()?

A) Promise.any() resolves with the first successful promise, while Promise.allSettled() provides the status of all promises.

15. What happens if no promises are provided to Promise.all()?

A) It resolves with an empty array.