Promise in js

Differences between promise and fetch:

- **Purpose**:
  - **Promise**: Ek general concept hai jo kisi bhi async operation ke liye use hota hai (timers, file operations, API calls, etc.).
  - **Fetch**: Specifically network requests (HTTP/HTTPS) ke liye bana hai, jaise API se data lena.

promise in JavaScript ek tareeka hai **asynchronous operations** ko handle karne ka.

1. Promise Kya Hai?

Promise ek object hota hai jo future mein koi value ya error dega. Ye teen states mein hota hai:

- **Pending**: Jab operation chal raha hota hai.
- **Fulfilled**: Jab operation successfully complete ho jata hai aur value milti hai.
- **Rejected**: Jab koi error aata hai aur operation fail ho jata hai.

Example:

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    let success = true;
    if (success) {
      resolve("Kaam ho gaya!");
    } else {
      reject("Kuch toh gadbad hai!");
    }
  }, 2000);
});

promise
  .then((result) => console.log(result)) // "Kaam ho gaya!" after 2 seconds
  .catch((error) => console.log(error));
```

2. Promise Ka Use Kahan Hota Hai?

Promise ka use wahan hota hai jahan **asynchronous tasks** ko handle karna hota hai, jaise:

- **API Calls**: Server se data fetch karna (e.g., `fetch` API).
- **File Operations**: File read/write karna (Node.js mein).
- **Timeouts/Intervals**: Time-based operations (e.g., `setTimeout`).
- **Database Queries**: Database se data lena ya save karna.
- **User Input**: Koi user action ka wait karna (e.g., file upload).

Basically, jab koi kaam turant complete nahi hota aur uska result baad mein aata hai, wahan promise use hota hai.

3. Promise Kyun Use Karte Hain?

- **Avoid Callback Hell**: Pehle callbacks use hote the, jo nested ho jate the aur code padhna mushkil ho jata tha (aka "callback hell"). Promise ke saath code cleaner aur readable hota hai.

  ```
  // Callback Hell Example
  getData(function(a) {
    getMoreData(a, function(b) {
      getEvenMoreData(b, function(c) {
        // ...aur aage
      });
    });
  });
  ```

  Promise ke saath:

  ```
  getData()
    .then(a => getMoreData(a))
    .then(b => getEvenMoreData(b))
    .then(c => console.log(c));
  ```

- **Error Handling**: `.catch` se errors ko easily handle kar sakte hain.
- **Chaining**: Multiple async operations ko `.then` se chain kar sakte hain.
- **Parallel Execution**: `Promise.all` se ek saath multiple promises ko run kar sakte hain.

  ```
  Promise.all([promise1, promise2]).then(results => console.log(results));
  ```

4. Fetch aur Promise Mein Same Ya Difference?

Fetch ek built-in JavaScript API hai jo HTTP requests (GET, POST, etc.) ke liye use hota hai, aur ye **promise-based** hai. Yani, fetch ek promise return karta hai. Let's break it down:

Similarities:

- Fetch promise ka use karta hai. Jab tum `fetch()` call karte ho, ye ek promise return karta hai jo ya toh resolve hota hai (response ke saath) ya reject hota hai (error ke saath).
- Dono asynchronous operations ke liye hain.

Differences:

- **Purpose**:
  - **Promise**: Ek general concept hai jo kisi bhi async operation ke liye use hota hai (timers, file operations, API calls, etc.).
  - **Fetch**: Specifically network requests (HTTP/HTTPS) ke liye bana hai, jaise API se data lena.
- **Implementation**:
  - Promise ko tum manually bana sakte ho (`new Promise`).
  - Fetch ek ready-made function hai jo browser provide karta hai, aur ye internally promise ka use karta hai.
- **Response Handling**:
  - Fetch ka promise resolve hone ke baad bhi response ko manually parse karna padta hai (e.g., `.json()`).

```
fetch('https://api.example.com/data')
  .then(response => response.json()) // Extra step
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

Fetch Example:

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    if (!response.ok) throw new Error('Network response was not ok');
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

5. Bonus: Async/Await

Promise ke saath ek modern tareeka hai **async/await**, jo promise-based code ko aur simple bana deta hai:

```
async function getData() {
  try {
    let response = await fetch('https://jsonplaceholder.typicode.com/posts/1');
    let data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('Error:', error);
  }
}
getData();
```

Yahan `await` promise ke resolve hone ka wait karta hai, aur code synchronous jaisa dikhta hai.

Summary:

- **Promise**: Async operations ke liye general tool, jo future mein result ya error deta hai.
- **Use Cases**: API calls, timeouts, file operations, etc.
- **Kyun Use**: Callback hell se bachta hai, error handling aur chaining ko easy karta hai.
- **Fetch**: Promise-based API specifically network requests ke liye. Ye promise ka ek practical implementation hai.
- **Async/Await**: Promise ka wrapper, jo code ko aur readable banata hai.