3BeezTechnologies Pvt. Ltd.

In JavaScript, promises are a mechanism for handling asynchronous operations. They provide a way to work with asynchronous code in a more structured and readable manner compared to using callbacks. Promises represent the eventual completion or failure of an asynchronous operation and allow you to attach callbacks to handle the results.

A promise is in one of three states:

Pending: The initial state; the promise is neither fulfilled nor rejected.

Fulfilled: The operation completed successfully, and the promise has a resulting value.

Rejected: The operation failed, and the promise has a reason for the failure.

Example

```
// Creating a promise
const myPromise = new Promise((resolve, reject) => {
 // Simulating an asynchronous operation (e.g., fetching data)
 setTimeout(() => {
  const success = true;

  if (success) {
   // Resolve the promise with a value
   resolve("Operation successful!");
  } else {
   // Reject the promise with a reason
```

```
    reject("Operation failed!");
  }
 }, 2000); // Simulating a 2-second delay
});


// Handling the promise
myPromise
 .then((result) => {
  // This callback is executed if the promise is fulfilled
  console.log("Fulfilled:", result);
 })
 .catch((error) => {
  // This callback is executed if the promise is rejected
  console.error("Rejected:", error);
 });
```

In the example above:

The Promise constructor takes a function as its argument, which has two parameters: resolve and reject. These are functions provided by the promise system.

Inside the function, you perform an asynchronous operation. When the operation is successful, you call resolve with the result; if it fails, you call reject with an error.

The then method is used to attach a callback that will be executed when the promise is fulfilled. The catch method is used to attach a callback for handling rejection.

Promises can be chained using the then method, making it easier to handle multiple asynchronous operations sequentially. Additionally, with the introduction of async/await syntax in modern JavaScript, working with promises has become even more convenient.

Async/Await:

With the introduction of the async and await keywords in ECMAScript 2017 (ES8), working with promises has become more readable and synchronous-like. The async keyword is used to define an asynchronous function, and await is used inside such a function to wait for a promise to settle.

```
const fetchData = async () => {
  try {
    const result1 = await firstAsyncOperation();
    console.log(result1);

    const result2 = await secondAsyncOperation(result1);
    console.log(result2);
  } catch (error) {
    console.error("Error:", error);
  }
};
```

```
fetchData();
```