No.

Date . .

JavaScript- Promises

In JavaScript, promise is an object that represents a value that might not be available yet. but will be resolved in the future either successfully or with an error.

It allows you to write asynchronous code that can handle the results when they become available.

Real life explanation

@Samarthya Pandey

Imagine...
(You ask your friend to give you a toy but your friend is not here right now.
Your friend promises to give you the toy when they came back) ← This is like promise in JavaScript.

In JavaScript, you can use Promises to ask the computer to do something like

No.

Date . .

load a picture or some text but it might take some time to get the information.

So the Promise is like a promise from the computer that it will give you the information when it is ready.

Just like your friend who promised to give you the toy, the Promise in JavaScript can either keep the promise and give you the information or it can break the promise and give you an error message instead.

@Samarthya Pandey

So you can add some code to handle what happens if the promise fails.

Fun Facts About Promises.

1. Promises were introduced in ES6 in 2015.
2. Promises can be in one of three states.
i) Pending ii) Resolved iii) Rejected
3. Promises can be chained together using .then() method.

No.

Date . .

4. Promises help simplify asynchronous code by allowing you to write code that depends on the result of asynchronous code

5. Promises are ~~t~~ not the only way to handle asynchronous code in JavaScript, but they have been widely used and popular approach.

@Samarthya Pandey

When to use Promises

1. When you need to make a small number of asynchronous operation that are simple to manage.

2. When you need to perform a sequence of asynchronous operation in certain order. Promises can help manage the flow of operations.

3. When you need to execute multiple asynchronous operations in paraell. Promises can help to monoge it efficently.

No.
Date . .

4. When you want to simply make the codebase more clearner and readable by avoiding nested call back.

* When to avoid → Promises.

1. When you are working on the project that doesn't require asynchronous operations.

@Samarthya Pandey

2. When you are working with a small + amount of ~~callback~~ asynchronous operation that can be easily managed with traditional callbacks.

3. When you are working on a project that has strict performance requirement and promises could introduce unnecessary overhead.

4. When you are not working with the libraries or frameworks that don't strictly recommend promises.

No.
Date . .

Code example:

~~fe~~ Here's a simple code example that uses Promises in JavaScript. to fetch data from API.

```
fetch('http://jsonplaceholder.com/parts/1')
.then(response => response.json())
.then(data => console.log (data))
.catch(error=> console.log (error));
```

@Samarthya Pandey

In this example, we use the fetch() method to make a request to an API to get the data about the post.
The fetch() method returns a Promise that resolves to the Response object, which we can then use to extract the JSON data.

We use '.then()' method to handle the successful resolution of the Promise
The first .then() method takes the Response object and calls the '.json()'

No.

Date . .

method on it to extract the JSON data.

We use the second '.then' method to handle the successful resolution of the Promise returned by '.json()'. This method takes the parsed JSON data and logs to the console.

@ Samarthya Pandey.

Finally,
We use the '.catch()' method to handle any errors that occurs during the Promise chain.

If there is error, it will be caught and logged to the console.