Java Script IN LWC

**Array Destructuring:**

const numbers = [1, 2, 3, 4, 5];

Explaination:-

const [a, b, c, d, e] = numbers;

console.log(a); // 1

console.log(b); // 2

### Object Destructuring:

const person = {

name: 'John Doe',

age: 30,

city: 'New York'

};

Explaination:-

const { name, age, city } = person;

console.log(name); // 'John Doe'

console.log(age); // 30

console.log(city); // 'New York'

OR  
const { name: personName, age: personAge } = person;

console.log(personName); // 'John Doe'

console.log(personAge); // 30  
  
Promises in JavaScript

1. **Pending:** Initial state, neither fulfilled nor rejected.
2. **Fulfilled:** The operation completed successfully.
3. **Rejected:** The operation failed.

// Simulating an asynchronous task with a timeout

const myAsyncTask = () => {

return new Promise((resolve, reject) => {

setTimeout(() => {

const success = true; // Simulate success

if (success) {

resolve('Operation succeeded'); // Resolve with a value

} else {

reject(new Error('Operation failed')); // Reject with an error

}

}, 2000);

});

};

// Consuming the Promise using .then() and .catch()

myAsyncTask()

.then(result => {

console.log('Success:', result); // Handling the resolved value

})

.catch(error => {

console.error('Error:', error.message); // Handling the rejected error

* });  
  **myAsyncTask** simulates an asynchronous task that will either resolve with a success message or reject with an error after a timeout of 2 seconds.
* **.then()** is used to handle the resolved value if the promise is fulfilled.
* **.catch()** is used to handle any errors if the promise is rejected.

Promises are commonly used in modern JavaScript, especially with APIs like **fetch** for making network requests, **setTimeout**, **setInterval**, and various other asynchronous operations, allowing cleaner and more organized handling of asynchronous code compared to callback-based approaches.

Top of Form