A Promise in JavaScript is an object that represents the eventual completion or failure of an asynchronous operation and its resulting value

A promise has the following states:

1. Fulfilled - successful
2. Rejected - failed
3. Pending - ongoing

```
const promise = new Promise ((resolve, reject) => {
    const state = false
    if (state) resolve ('true')
    else reject('not true')
})

promise.then(result => console.log(result)).catch(() => console.log("error!"))
```

```
function promise() {
 new Promise ((resolve, reject) => {
    const data = { name: "John", age: 24}
    if (true) resolve(data)
    else reject("Error")
})

promise().then((result) => console.log(result)).catch((err) => console.log("Error", err))
```

Promise Methods:

1. then() : Used to handle the result of a successful promise.

```
myPromise.then((result) => {
  console.log('Success:', result);
});
```

2. catch() : Used to handle errors in a promise chain.

```
myPromise.catch((error) => {
  console.error('Error:', error);
});
```

Promise.all():

Promise.all() takes an array of promises and returns a single promise that resolves when all of the promises in the array have resolved or rejects with the reason of the first promise that rejects

```
const promise1 = Promise.resolve("one")
const promise2 = 42
const promise3 = new Promise ((resolve, reject) => {
    setTimeout(() => {
```

```
        resolve("Three")
    }, 3000)
})

Promise.all([promise1, promise2, promise3])
.then((result) => console.log(result))
.catch(() => console.log("error!"))
```

Promise Chaining:

Promise chaining allows you to perform multiple asynchronous operations sequentially. Each .then() block can return another promise, allowing you to chain operations.

```
myPromise.then((result) => {
  console.log('Step 1:', result);
  return anotherPromise; // Return a new promise
}).then((result) => {
  console.log('Step 2:', result);
}).catch((error) => {
  console.error('Error:', error);
});

// Function simulating an asynchronous operation that resolves after a delay
function asyncOperation1() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Result of asyncOperation1');
    }, 1000);
  });
}

// Function simulating another asynchronous operation that resolves after a delay
function asyncOperation2() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Result of asyncOperation2');
    }, 500);
  });
}

// Chain the promises
asyncOperation1()
  .then((result1) => {
    console.log(result1);
    // Return another promise to continue the chain
    return asyncOperation2();
  })
  .then((result2) => {
    console.log(result2);
    console.log('Promise chain completed');
  })
  .catch((error) => {
    console.error('Error:', error);
```

```
  });
```

Promise.race():

Promise.race() takes an iterable of promises and returns a promise that resolves or rejects as soon as one of the promises in the iterable resolves or rejects.

```
const promise1 = new Promise((resolve, reject) => {
  setTimeout(resolve, 500, 'One');
});

const promise2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, 'Two');
});

Promise.race([promise1, promise2]).then((value) => {
  console.log(value); // Output: "Two"
});
```