Promises

-

According to MDN The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

Basically it allows you to add handlers with an asynchronous action's eventual success value or failure. So you get the result with little bit of delay which is a promise which will give the result at some point.

A Promise is in one of these states:

1. pending: initial state, neither fulfilled nor rejected.
2. fulfilled: meaning that the operation was completed successfully.
3. rejected: meaning that the operation failed.

```
console.log('start')

const sub = new Promise((resolve,reject)=>{
    setTimeout(()=>{
        const result = false;
        if(result) resolve('go to sleep')
        else reject(new Error("Solve the problem first"))
    },2000)
})
sub.then((res)=>{
    console.log(res)
})
.catch((err)=>{
    console.log(err)
})

console.log('stop')//output:start
                    //stop
                    //Error: Solve the problem first
```

Chaining the Promise with then():

```
// returns a promise

let countValue = new Promise(function (resolve, reject) {
  resolve("Promise resolved");
});
```

```
// executes when promise is resolved successfully

countValue
  .then(function successValue(result) {
    console.log(result);
  })

  .then(function successValue1() {
    console.log("You can call multiple functions this way.");
  });
```

JavaScript Promise Methods:

There are various methods available to the Promise object.

Method	Description
`all(iterable)`	Waits for all promises to be resolved or any one to be rejected
`allSettled(iterable)`	Waits until all promises are either resolved or rejected
`any(iterable)`	Returns the promise value as soon as any one of the promises is fulfilled
`race(iterable)`	Wait until any of the promises is resolved or rejected
`reject(reason)`	Returns a new Promise object that is rejected for the given reason
`resolve(value)`	Returns a new Promise object that is resolved with the given value
`catch()`	Appends the rejection handler callback
`then()`	Appends the resolved handler callback
`finally()`	Appends a handler to the promise

Promise.all():

The `Promise.all()` static method takes an iterable of promises as input and returns a single `Promise`. This returned promise fulfills when all of the input's promises fulfill (including when an empty iterable is passed), with an array of the fulfillment values. It rejects when any of the input's promises rejects, with this first rejection reason.

```
const promise1 = Promise.resolve(3);
const promise2 = 42;
const promise3 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, 'foo');
});

Promise.all([promise1, promise2, promise3]).then((values) => {
  console.log(values);
});
// Expected output: Array [3, 42, "foo"]
```

Promise.allSettled():

```
const promise1 = Promise.resolve(3);
const promise2 = new Promise((resolve, reject) => setTimeout(reject, 100, 'foo'));
const promises = [promise1, promise2];

Promise.allSettled(promises).then((results) => results.forEach((result) => console.log(result)));

// Expected output:
// Object { status: "fulfilled", value: 3 }
// Object { status: "rejected", reason: "foo" }
```

Promise.any():

```
const promises = [
  Promise.reject("First promise failed"),
  Promise.resolve("Second promise succeeded"),
  Promise.reject("Third promise failed")
];

Promise.any(promises)
  .then(result => {
    console.log("First resolved promise:", result);
  })
  .catch(error => {
    console.error("All promises were rejected:", error);
  });
```

Question-01: What’s the output?

```
console.log('start')

const promise1 = new Promise((resolve,reject)=>{
    console.log(1);
    resolve(2);
    console.log(3);
})
promise1.then((res)=>{
    console.log(res)
})
.catch((err)=>{
    console.log(err)
})

console.log('stop')//output:start 1 3 stop 2
```

if resolve and reject is never used it will not output anything:

```
const promise1 = new Promise((resolve,reject)=>{
    console.log(1);

})
promise1.then((res)=>{
    console.log(res); //no output
})
```

Question-02: What’s the output?

```
console.log('start')

const fn= () => new Promise((resolve,reject)=>{
    console.log(1);
    resolve("success");
})
console.log("middle")
fn().then((res)=>{
    console.log(res)
})
.catch((err)=>{
    console.log(err)
})

console.log('stop')//output:start  middle 1 stop success
```