Namaste JavaScript - Promise APIs - Comprehensive Notes

Introduction & Importance

- **What are Promise APIs?** They are built-in static methods on the Promise object that help handle multiple promises efficiently.
- **Why are they important?**
  - **Interviews:** Very common and crucial topic. Interviewers often ask to differentiate between them.
  - **Real-World Applications:** Essential for making parallel API calls in day-to-day applications (e.g., in React projects).
  - **Use Case:** The instructor used them in the Namaste React course for parallel API calls to populate data.

The Four Major Promise APIs

There are four main Promise APIs to study:

1. Promise.all
2. Promise.allSettled
3. Promise.race
4. Promise.any

---

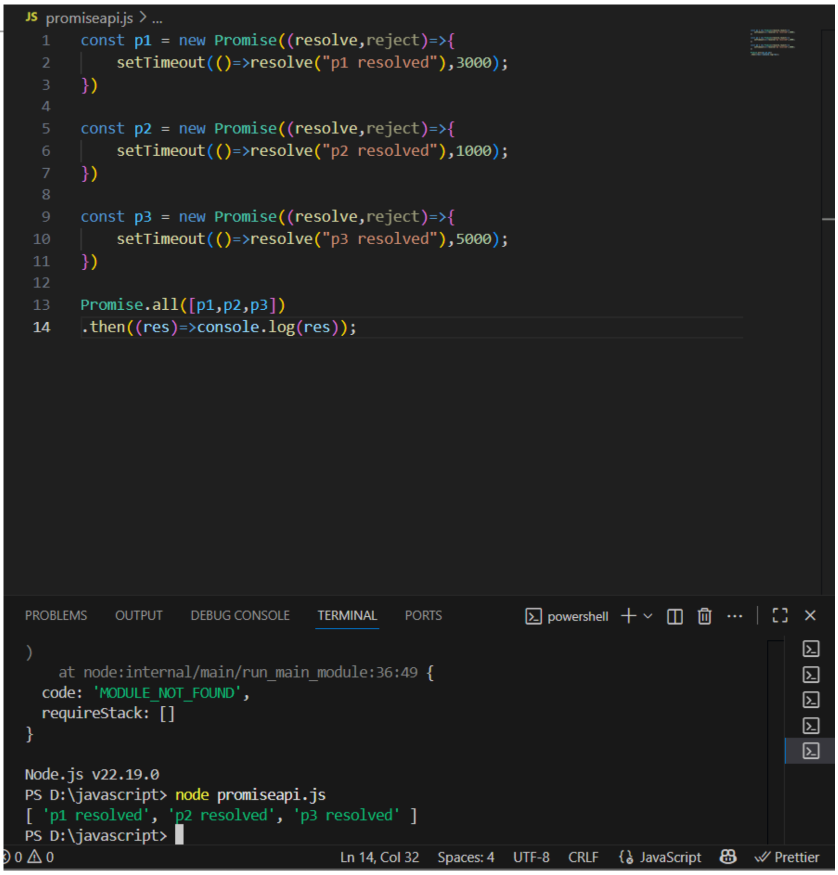
1. Promise.all

- **Use Case:** When you need to make multiple parallel API calls and get the results, but you need **all** calls to be successful to proceed.
- **Input:** An iterable (commonly an array) of promises. [P1, P2, P3]
- **Output:** A single Promise.

Scenarios:

A) Success Case (All promises resolve):

- **Behavior:** It waits for **all** promises to finish. The output is an **array** of the results, in the same order as the input promises.



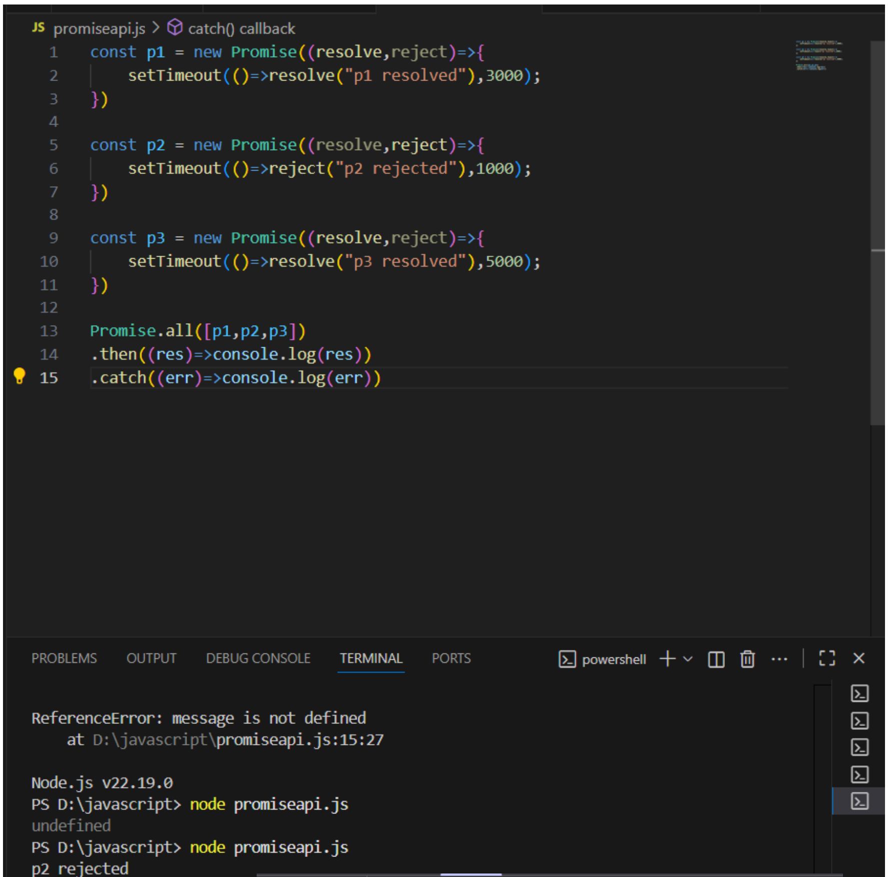
```javascript
const p1 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p1 resolved"),3000);
})

const p2 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p2 resolved"),1000);
})

const p3 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p3 resolved"),5000);
})

Promise.all([p1,p2,p3])
.then((res)=>console.log(res));
```

```
)
    at node:internal/main/run_main_module:36:49 {
  code: 'MODULE_NOT_FOUND',
  requireStack: []
}

Node.js v22.19.0
PS D:\javascript> node promiseapi.js
[ 'p1 resolved', 'p2 resolved', 'p3 resolved' ]
PS D:\javascript>
```

- **Failure Handling:** It is **NOT** a fail-fast in this case; it waits for all to complete successfully.
- **Timing:** The total time is the time taken by the **slowest** promise.

Example:

  - P1 (3 sec): Resolves with "Val1"
  - P2 (1 sec): Resolves with "Val2"
  - P3 (2 sec): Resolves with "Val3"
  - **Output (after 3 seconds):** ["Val1", "Val2", "Val3"]

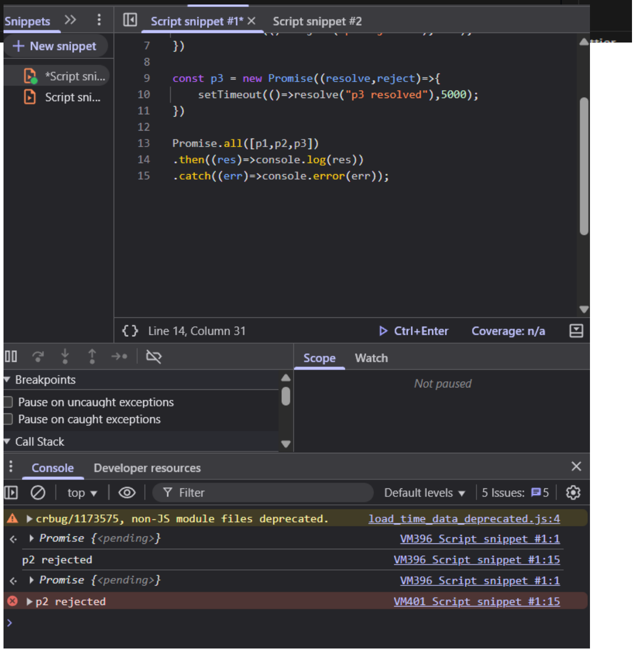


```
const p1 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p1 resolved"),3000);
})

const p2 = new Promise((resolve,reject)=>{
    setTimeout(()=>reject("p2 rejected"),1000);
})

const p3 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p3 resolved"),5000);
})

Promise.all([p1,p2,p3])
.then((res)=>console.log(res))
.catch((err)=>console.log(err))
```

```
ReferenceError: message is not defined
    at D:\javascript\promiseapi.js:15:27

Node.js v22.19.0
PS D:\javascript> node promiseapi.js
undefined
PS D:\javascript> node promiseapi.js
p2 rejected
PS D:\javascript>
```

B) Failure Case (Any one promise rejects):

- **Behavior:** It is **Fail-Fast**. As soon as **any** promise in the iterable rejects, Promise.all immediately rejects with that same error.
- **It does NOT wait** for the other promises to settle (whether they resolve or reject later is irrelevant at this point).
- **Note:** The other API calls are already made and cannot be cancelled, but their results are ignored by Promise.all.



```
})

const p3 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p3 resolved"),5000);
})

Promise.all([p1,p2,p3])
.then((res)=>console.log(res))
.catch((err)=>console.error(err));
```

```
crbug/1173575, non-JS module files deprecated.    load_time_data_deprecated.js:4
< Promise {<pending>}                              VM396 Script snippet #1:1
p2 rejected                                        VM396 Script snippet #1:15
< Promise {<pending>}                              VM396 Script snippet #1:1
p2 rejected                                        VM401 Script snippet #1:15
>
```

Example:

  - P1 (3 sec): Resolves
  - P2 (1 sec): Rejects with "P2 Fail"
  - P3 (2 sec): Resolves

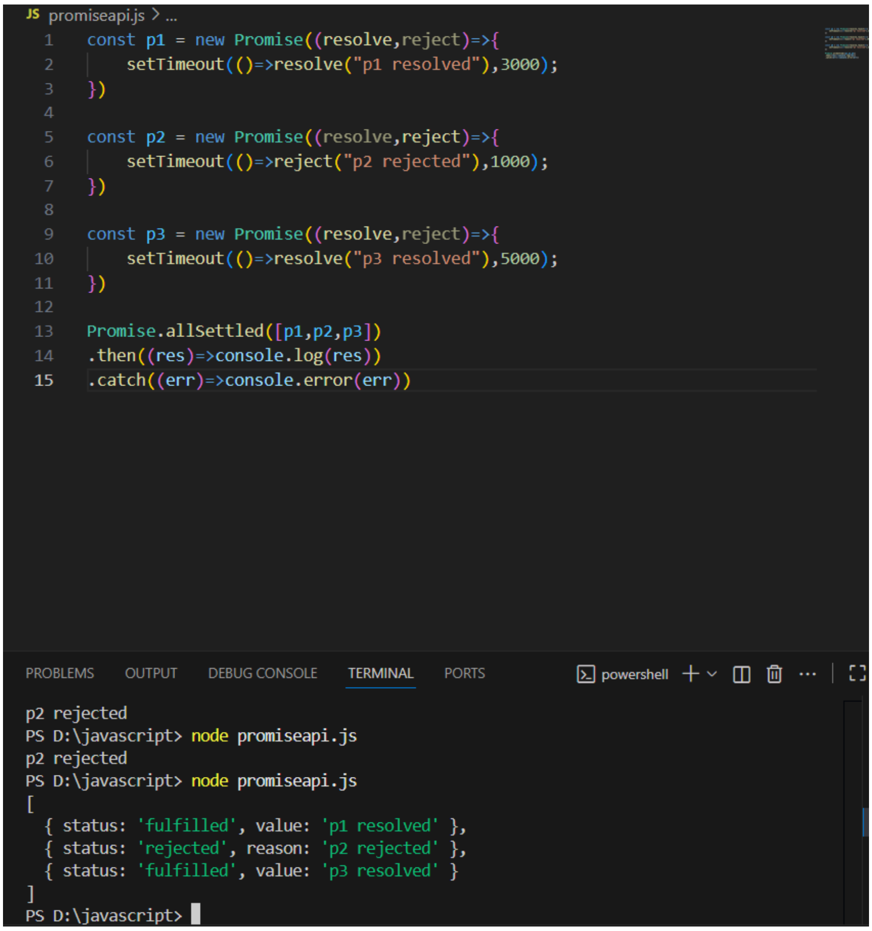
- **Output (after 1 second):** Error "P2 Fail"

- **Summary:** "All or None." You either get an array of all results, or a single error from the first rejected promise.

---

2. Promise.allSettled

- **Use Case:** When you want to know the outcome of all promises, regardless of whether they were successful or failed. (e.g., showing 4 out of 5 cards on a page if one API fails).



```javascript
const p1 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p1 resolved"),3000);
})

const p2 = new Promise((resolve,reject)=>{
    setTimeout(()=>reject("p2 rejected"),1000);
})

const p3 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p3 resolved"),5000);
})

Promise.allSettled([p1,p2,p3])
.then((res)=>console.log(res))
.catch((err)=>console.error(err))
```

```
p2 rejected
PS D:\javascript> node promiseapi.js
p2 rejected
PS D:\javascript> node promiseapi.js
[
  { status: 'fulfilled', value: 'p1 resolved' },
  { status: 'rejected', reason: 'p2 rejected' },
  { status: 'fulfilled', value: 'p3 resolved' }
]
PS D:\javascript> 
```

- **Input:** An iterable (array) of promises. [P1, P2, P3]

- **Output:** A single Promise that resolves after all input promises have **settled** (either fulfilled or rejected).

Scenarios:

A) Success Case (All promises resolve):

- **Behavior:** Waits for all promises to finish and returns an array of objects describing the outcome of each promise.

- **Output Format:** Each object has a status and a value.
  - status: "fulfilled"
  - value: The resolved value.

Example (All succeed):

- **Output:** [{status: "fulfilled", value: "Val1"}, {status: "fulfilled", value: "Val2"}, {status: "fulfilled", value: "Val3"}]

B) Failure Case (Some promises reject):

- **Behavior:** It **still waits for all promises** to settle. It does not fail fast.

- **Output Format:** The result array contains a mix of fulfilled and rejected outcome objects. For rejected promises, the object has a status and a reason.
  - status: "rejected"
  - reason: The error/rejection reason.

Example:

- P1 (3 sec): Resolves with "Val1"

- ○ P2 (1 sec): Rejects with "P2 Fail"
- ○ P3 (2 sec): Resolves with "Val3"
- ○ **Output (after 3 seconds):**

javascript

```
[
 {status: "fulfilled", value: "Val1"},
 {status: "rejected", reason: "P2 Fail"},
 {status: "fulfilled", value: "Val3"}
]
```

- **Summary:** The "safest" option. It always waits for all results and gives you a complete picture of what happened.

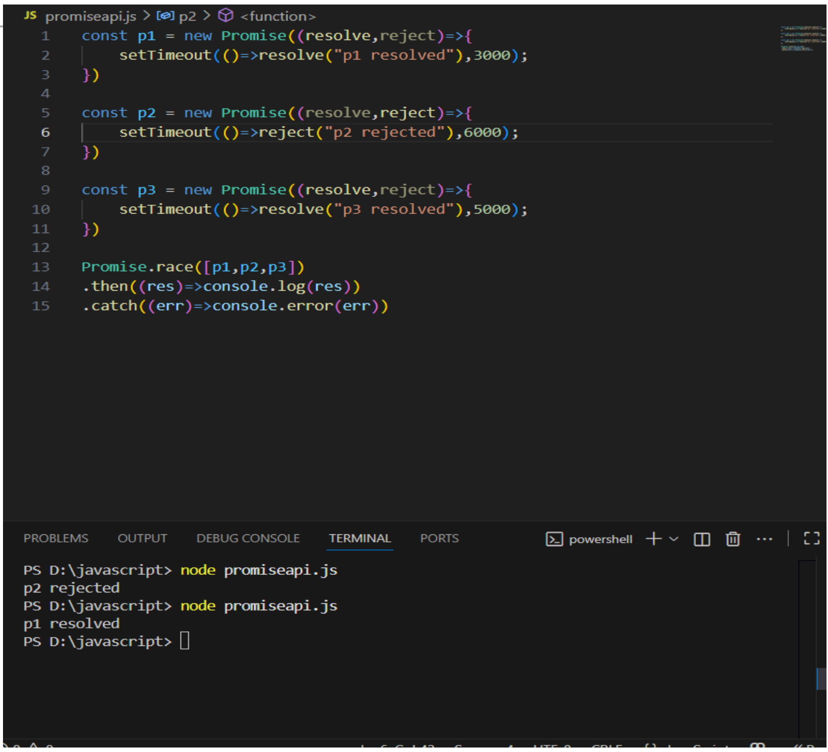
---

3. Promise.race

- **Use Case:** When you are only interested in the result of the first promise that settles (completes, whether successfully or not). It's a literal "race."
- **Input:** An iterable (array) of promises. [P1, P2, P3]
- **Output:** A single Promise.

Scenarios:

- **Behavior:** Returns a promise that fulfills or rejects as soon as **the first promise in the iterable fulfills or rejects**. The result is the value or error from that first settled promise.



```javascript
const p1 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p1 resolved"),3000);
})

const p2 = new Promise((resolve,reject)=>{
    setTimeout(()=>reject("p2 rejected"),6000);
})

const p3 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p3 resolved"),5000);
})

Promise.race([p1,p2,p3])
.then((res)=>console.log(res))
.catch((err)=>console.error(err))
```

```
PS D:\javascript> node promiseapi.js
p2 rejected
PS D:\javascript> node promiseapi.js
p1 resolved
PS D:\javascript>
```

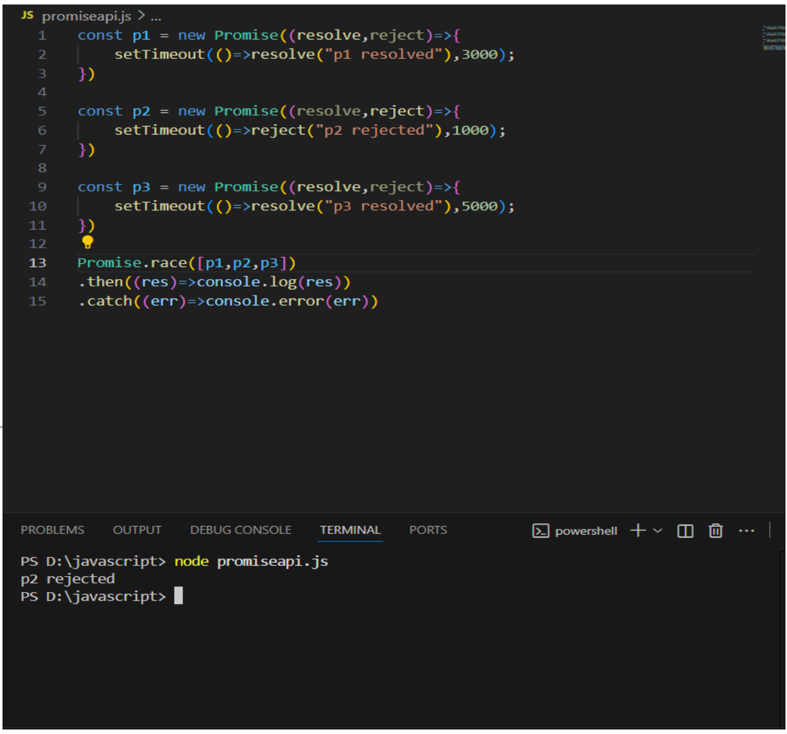
Example 1 (First promise resolves):

- P1 (3 sec): Resolves
- P2 (1 sec): Resolves with "Val2"
- P3 (2 sec): Rejects
- **Output (after 1 second):** "Val2" (The result of the first settled promise, P2)

Example 2 (First promise rejects):

- P1 (3 sec): Resolves

- P2 (5 sec): Resolves
- P3 (2 sec): Rejects with "P3 Fail"
- **Output (after 2 seconds):** Error "P3 Fail" (The result of the first settled promise, P3)
- **Summary:** "First settled promise wins the race." Outcome can be success or failure.



```javascript
const p1 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p1 resolved"),3000);
})

const p2 = new Promise((resolve,reject)=>{
    setTimeout(()=>reject("p2 rejected"),1000);
})

const p3 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p3 resolved"),5000);
})

Promise.race([p1,p2,p3])
.then((res)=>console.log(res))
.catch((err)=>console.error(err))
```

```
PS D:\javascript> node promiseapi.js
p2 rejected
PS D:\javascript>
```

4. Promise.any

- **Use Case:** When you are interested in the **first successful promise**. You want to ignore rejections and wait for a success. It's a "success-seeking" race.
- **Input:** An iterable (array) of promises. [P1, P2, P3]
- **Output:** A single Promise.

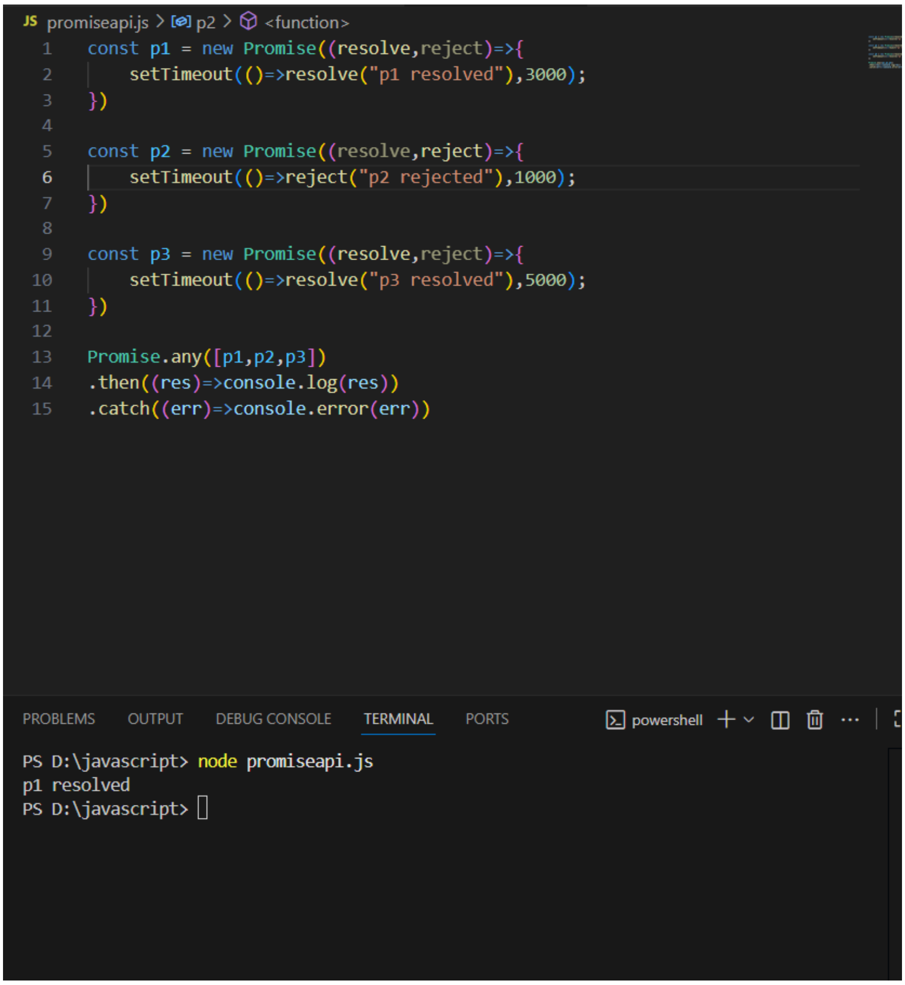
Scenarios:

A) At least one promise resolves:

- **Behavior:** Returns a promise that fulfills as soon as **the first promise in the iterable fulfills**. It ignores rejections.

Example:

  - P1 (3 sec): Resolves with "Val1"
  - P2 (1 sec): Rejects
  - P3 (2 sec): Rejects



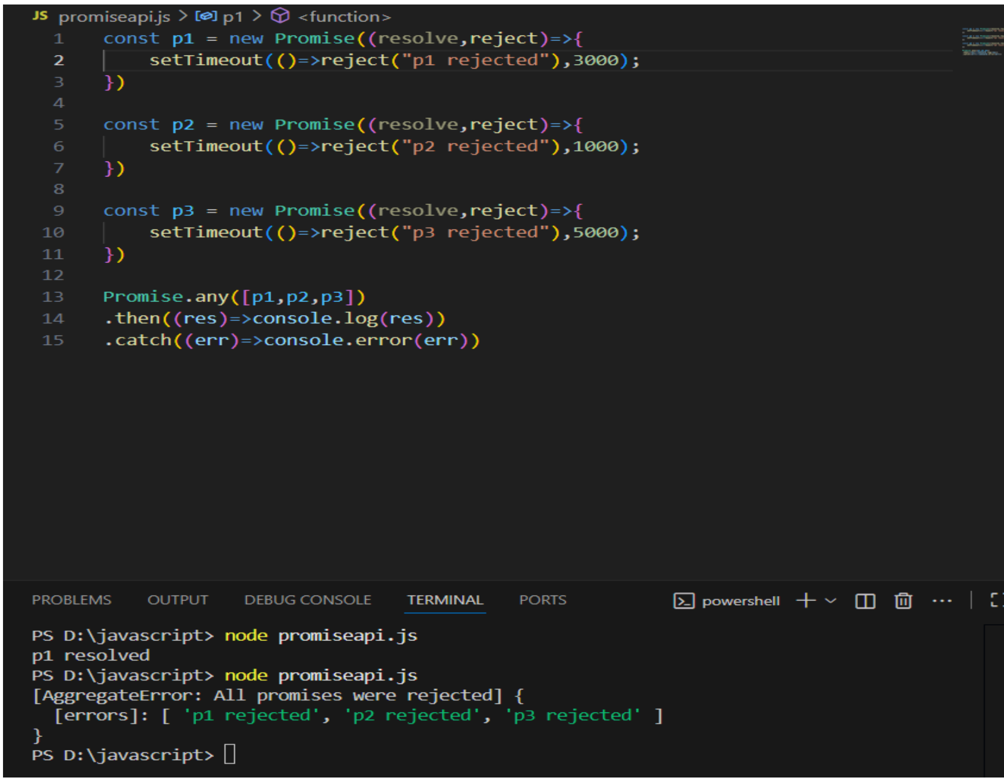
```javascript
const p1 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p1 resolved"),3000);
})

const p2 = new Promise((resolve,reject)=>{
    setTimeout(()=>reject("p2 rejected"),1000);
})

const p3 = new Promise((resolve,reject)=>{
    setTimeout(()=>resolve("p3 resolved"),5000);
})

Promise.any([p1,p2,p3])
.then((res)=>console.log(res))
.catch((err)=>console.error(err))
```

```
PS D:\javascript> node promiseapi.js
p1 resolved
PS D:\javascript>
```

- ○ **Output (after 3 seconds):** "Val1" (The result of the first *successful* promise)

B) All promises reject:

- **Behavior:** It waits until all promises have been rejected. Then, it returns a special AggregateError.



```javascript
const p1 = new Promise((resolve,reject)=>{
    setTimeout(()=>reject("p1 rejected"),3000);
})

const p2 = new Promise((resolve,reject)=>{
    setTimeout(()=>reject("p2 rejected"),1000);
})

const p3 = new Promise((resolve,reject)=>{
    setTimeout(()=>reject("p3 rejected"),5000);
})

Promise.any([p1,p2,p3])
.then((res)=>console.log(res))
.catch((err)=>console.error(err))
```

```
PS D:\javascript> node promiseapi.js
p1 resolved
PS D:\javascript> node promiseapi.js
[AggregateError: All promises were rejected] {
  [errors]: [ 'p1 rejected', 'p2 rejected', 'p3 rejected' ]
}
PS D:\javascript> 
```

- AggregateError: An error object that holds an array of all the rejection reasons inside its .errors property.

Example (All fail):

- ○ P1 (3 sec): Rejects with "P1 Fail"
- ○ P2 (1 sec): Rejects with "P2 Fail"
- ○ P3 (2 sec): Rejects with "P3 Fail"
- ○ **Output (after 3 seconds):** An AggregateError. You can access the errors via error.errors:

javascript

// error.errors would be:

["P1 Fail", "P2 Fail", "P3 Fail"]

- **Summary:** "Seeking the first success." If it finds one, it returns it. If all fail, it returns an aggregated list of errors.

Key Terminology (Lingo) - CRITICAL FOR INTERVIEWS

- **Settled:** A promise is said to be *settled* when it is no longer pending—it has either been **fulfilled** (resolved) or **rejected**. This is the key umbrella term.
- **Fulfilled / Resolved:** The successful state of a promise.
- **Rejected:** The failed state of a promise.

API	Input	Output (Success)	Output (Failure)	Key Behavior
Promise.all	Promises Array	Array of results	**Fail-Fast.** Error from first rejection.	"All or None." Waits for all successes, fails fast on any error.
Promise.allSettled	Promises Array	Array of outcome objects	Array of outcome objects (incl. errors).	"Safe & Complete." Always waits for all to settle.
Promise.race	Promises Array	Value of first settled promise	Error of first settled promise.	"First Settled Wins." Outcome can be success or failure.
Promise.any	Promises Array	Value of first **fulfilled** promise	AggregateError (if all reject).	"First Success Wins." Ignores rejections until it finds a success.

Relationship:

text

A Promise is..

Pending -> Becomes...

    Settled -> (Either)

        Fulfilled/Resolved (Success)

        Rejected (Failure)

Quick Revision & Interview Cheat Sheet