PROMISE.RACE VS. PROMISE.ANY AND PROMISE.ALL VS. PROMISE.ALLSETTLED

☸ PROMISE STATE DEFINITIONS

- **Fulfilled** - When a promise is resolved successfully.
- **Rejected** - When a promise failed.
- **Pending** - When a promise is "neither fulfilled nor rejected".
- **Settled** - Not really a state but an *umbrella term* to describe that a promise is either fulfilled or rejected

🚀 DIFFERENCES

Let's first take a look at difference between existing & new combinator methods.

☸ Promise.all vs. Promise.allSettled

- Both accepts an iterable object but
- **Promise.all** rejects as soon as a promise within the iterable object rejected.
- **Promise.allSettled** resolves regardless of rejected promise(s) within the iterable object.

☸ Promise.race vs. Promise.any

- Both accepts an iterable object but
- **Promise.race** short-circuits on the first settled (fulfilled or rejected) promise within the iterable object.
- **Promise.any** short-circuits on the first fulfilled promise and continues to resolve regardless of rejected promises unless all within the iterable object reject.

🚀 Comparison Table

Now let's take a look at existing/upcoming combinator methods.

	Short-circuit?	**Short-circuits on?**	**Fulfilled on?**	**Rejected on?**
Promise.all	Yes	First rejected promise	All promise fulfilled	First rejected promise
Promise.allSettled	No	N/A	Always	N/A
Promise.race	Yes	First settled	First promise fulfilled	First rejected promise
Promise.any	Yes	First fulfilled	First promise fulfilled	All rejected promises

🚀 Promise.all: **What is this?** Resolve all promises passed as an iterable object.

Promise.all resolve all promises passed as an iterable object. The main characteristic is that it rejects completely when an input value is rejected. Basically, it follows an all-or-nothing methodology.

```
const promises = [
  Promise.resolve('Luke'),
  'Lea',
  new Promise((resolve, reject) => setTimeout(resolve, 100, 'Han Solo'))
];
Promise.all(promises)
.then((characters) =>
  console.log(`They are all getting along `, characters)
);

//output : "They are all getting along ", ["Luke", "Lea", "Han Solo"]
```

```
const promises = [
  Promise.resolve('Luke'),
  'Lea',
  new Promise((resolve, reject) => setTimeout(resolve, 100, 'Han Solo')),
  new Promise((_, reject) => setTimeout(reject, 100, 'Darth Vader'))
];
Promise.all(promises)
  .then((characters) => console.log(`They are all getting along `, characters))
  .catch((characters) =>
    console.error(`They are getting along because of `, characters)
  );

//output : "They are getting along because of ", "Darth Vader"
```

So in the first example, you can see that Promise.all is fulfilled and all the names are returned. But in the next example, you can see that one rejected promise results in rejecting all the promises (all-or-nothing logic).

The important piece about it is the last part, that is you cannot handle partial failures. If one of the promises fails, Promse.all gets rejected.Potential use cases for Promise.all would be when you want to aggregate the success of multiple promises into an array when they all are successful and not get the result if even one of them fails. A potential use case will be that you are making multiple API calls to different services and you want to display a component only once after all of them have succeeded. And if any of the calls fail, you want to show an error message to the user since the data set is incomplete. In that case, Promise.all makes perfect sense.

🚀 Promise.allSettled : **What is this?** all promises regardless of settled (fulfilled/rejected) status.

Promise.allSettled method returns a promise that resolves after all of the given promises have either "fulfilled" or "rejected", with an array of objects that each describes the outcome of each promise.

```
const promises = [
  Promise.resolve('Luke'),
  'Lea',
  new Promise((resolve, reject) => setTimeout(resolve, 100, 'Han Solo'))
];
Promise.allSettled(promises).then((characters) => console.log(characters));

/*output:[{ status: "fulfilled", value: "Luke"},
{ status: "fulfilled", value: "Lea"},
{ status: "fulfilled", value: "Han Solo"}]
*/
```

```
const promises = [
  Promise.resolve('Luke'),
  'Lea',
  new Promise((resolve, reject) => setTimeout(resolve, 100, 'Han Solo')),
  new Promise((_, reject) => setTimeout(reject, 100, 'Darth Vader'))
];
Promise.allSettled(promises).then((characters) => console.log(characters));

/*output:[{ status: "fulfilled", value: "Luke"},
{ status: "fulfilled", value: "Lea"},
{ status: "fulfilled", value: "Han Solo"},
{ reason: "Darth Vader", status: "rejected"}]
*/
```

As you might have noticed Promise.allSettled is a method that can be typically used when you have multiple asynchronous tasks that are not dependent on one another to complete successfully, unlike Promise.all() may be more appropriate if the tasks are dependent on each other. Once all promises are settled, this method returns an array of objects. Each object has a status key with the value of the promise (fulfilled or rejected), and the other one will be the value if the promise is fulfilled or the reason if it got rejected.
Promise.allSettled does not get rejected if any of the promises are rejected

Promise.race: **What is this?** The first fulfilled promise or reject the whole promise when even one promise rejects.

Promise.race() method returns a promise that fulfills or rejects as soon as one of the promises in an iterable fulfills or rejects, with the value or reason from that promise. Hence, it short-circuits when the first input promise value is "**settled**".

```
const promises = [
  Promise.resolve('Luke'),
  'Lea',
  new Promise((resolve, reject) => setTimeout(resolve, 100, 'Han Solo'))
];
Promise.race(promises).then((characters) => console.log(characters));

//output: "Luke"
```

```
const promises = [
  new Promise((resolve, reject) => setTimeout(resolve, 100, 'Han Solo')),
  new Promise((_, reject) => setTimeout(reject, 200, 'Darth Vader'))
];
Promise.race(promises)
  .then((characters) => console.log(characters))
  .catch((characters) => console.error(characters));

//output: "Han Solo"
```

```
const promises = [
  new Promise((resolve, reject) => setTimeout(resolve, 200, 'Han Solo')),
  new Promise((_, reject) => setTimeout(reject, 100, 'Darth Vader'))
];
Promise.race(promises)
  .then((characters) => console.log(characters))
  .catch((characters) =>
    console.error(`They are getting along because of `, characters)
  );

//output: "They are getting along because of ", "Darth Vader"
```

🚀 Promise.any : **What is this?** Returns the first fulfilled promise regardless of other rejected promises. If all promises reject, then reject by providing errors for all rejects.

```
const promises = [
  Promise.resolve('Luke'),
  'Lea',
  new Promise((resolve, reject) => setTimeout(resolve, 100, 'Han Solo'))
];
Promise.any(promises).then((characters) => console.log(`Statues `, characters));

//output: "Statues ", "Luke"
```

Promise.any takes an iterable of Promise objects and, as soon as one of the promises in the iterable fulfills, returns a single promise that resolves with the value from that promise. Each difference with respect to Promise.race short-circuits on the first "fulfilled" promise. Essentially, as you can see this method is the opposite of Promise.all. An important note for Promise.any would be to check the compatibility since it might not be completely available on all environments.

```
const promises = [
  new Promise((resolve, reject) => setTimeout(resolve, 200, 'Han Solo')),
  new Promise((_, reject) => setTimeout(reject, 100, 'Darth Vader'))
];
Promise.any(promises)
  .then((characters) => console.log(characters))
  .catch((characters) => console.error(characters));

//output: "Han Solo"
```

```
const promises = [
  new Promise((_, reject) => setTimeout(reject, 200, 'Ben Solo')),
  new Promise((_, reject) => setTimeout(reject, 100, 'Darth Vader'))
];
Promise.any(promises)
  .then((characters) => console.log(characters))
  .catch((characters) => console.error(characters));

//output: AggregateError
```

Follow me for such info

https://www.linkedin.com/in/priya-bagde/

GitHub

https://github.com/priya42bagde

YouTube

https://www.youtube.com/channel/UCK1_Op30_pZ1zBs9l3HNyBw/videos (Priya Frontend Vlogs)