

asynchronous JS cheatsheet

 Pending promises can become either...

 Fulfilled with a value, or...

 Rejected with an error.

 Either way, they are settled with an outcome.

Combining promises

Use `all()` to turn an array of promises into a promise to an array.

```
Promise.all([
  value1,
  value2,
  value3,
]) → [value1, value2, value3]
```

If any promise is rejected, the error will be passed through.

```
Promise.all([
  ...,
  ...,
  error,
]) → error
```

Use `race()` instead to pass through the first settled promise.

```
Promise.race([
  ...,
  ...,
  value,
]) → value
```

async/await

Calling an async function *always* results in a promise.

`(async () => value)() → value`

`(async () => outcome)() → outcome`


`(async () => throw error)() → error`


`await` waits for a promise to be fulfilled, then returns its value.

```
async function() {
  try {
    let value = await outcome
    // ...
  }
  catch (error) {
    // ...
  }
}
```

You can pass non-promise values to `await`

```
const fn = async () => {
  let value = await value
  // ...
}
```

`await` may only be used within async functions. 

`await` will wait until at least the next tick before returning, even when awaiting already-fulfilled promises or non-promise values. 

promise.then(onFulfilled, onRejected)

Calls onFulfilled once the promise is fulfilled.

`value.then(value => nextValue, ...?) → nextValue`

`value.then(value => outcome, ...?) → outcome`

`value.then(value => throw error, ...?) → error`

Calls onRejected if the promise is rejected.

`error.then(...?, error => value) → value`

`error.then(...?, error => outcome) → outcome`

`error.then(...?, error => throw nextError) → nextError`

Passes errors through if onRejected is undefined.

`error.then(...) → error`

promise.catch(onRejected)

Behaves identically to `then` when onFulfilled is omitted.

`error.catch(onRejected) ⇔ error.then(...?, onRejected)`


Passes fulfilled values through.

`value.catch(...) → value`

promise.finally(onFinally)

Calls onFinally with *no arguments* once any outcome is available. Passes through input promise.

`outcome.finally(() => ...) → outcome`

The onFulfilled, onRejected and onFinally functions will not be executed until at least the next tick, even for promises that already have an outcome. 

Making promises

The function passed to `new Promise` will be executed synchronously.

```
new Promise((resolve, reject) => {
  doImportantStuff((error, value) => {
    if (error)
      reject(error)
    else
      resolve(value)
  })
})
```

Use `resolve()` or `reject()` to create promises from values.

`Promise.resolve(value) → value`

`Promise.reject(error) → error`

If you put a fulfilled promise into a fulfilled promise, they'll collapse into one.

`Promise.resolve(value) → value`

Sometimes you might not need `reject`, or might not resolve to a value.

```
function delay(milliseconds) {
  return new Promise(resolve =>
    setTimeout(resolve, milliseconds)
  )
}
```