ES7 asynchronous patterns

The new frontiers of JavaScript and NodeJS


NODESCHOOL
S V G


NODECONF
BARCELONA


W

Questions

1. Javascript Asynchronicity
2. Promises
3. async / await

Javascript is asynchronous

Callbacks

```
function foo (param1, param2, callback) {
  bar(param1, function(result) {
    callback(result);
  })
}

foo('a', 'b', function (result) {
  console.log('Result:', result);
});
```

Callbacks

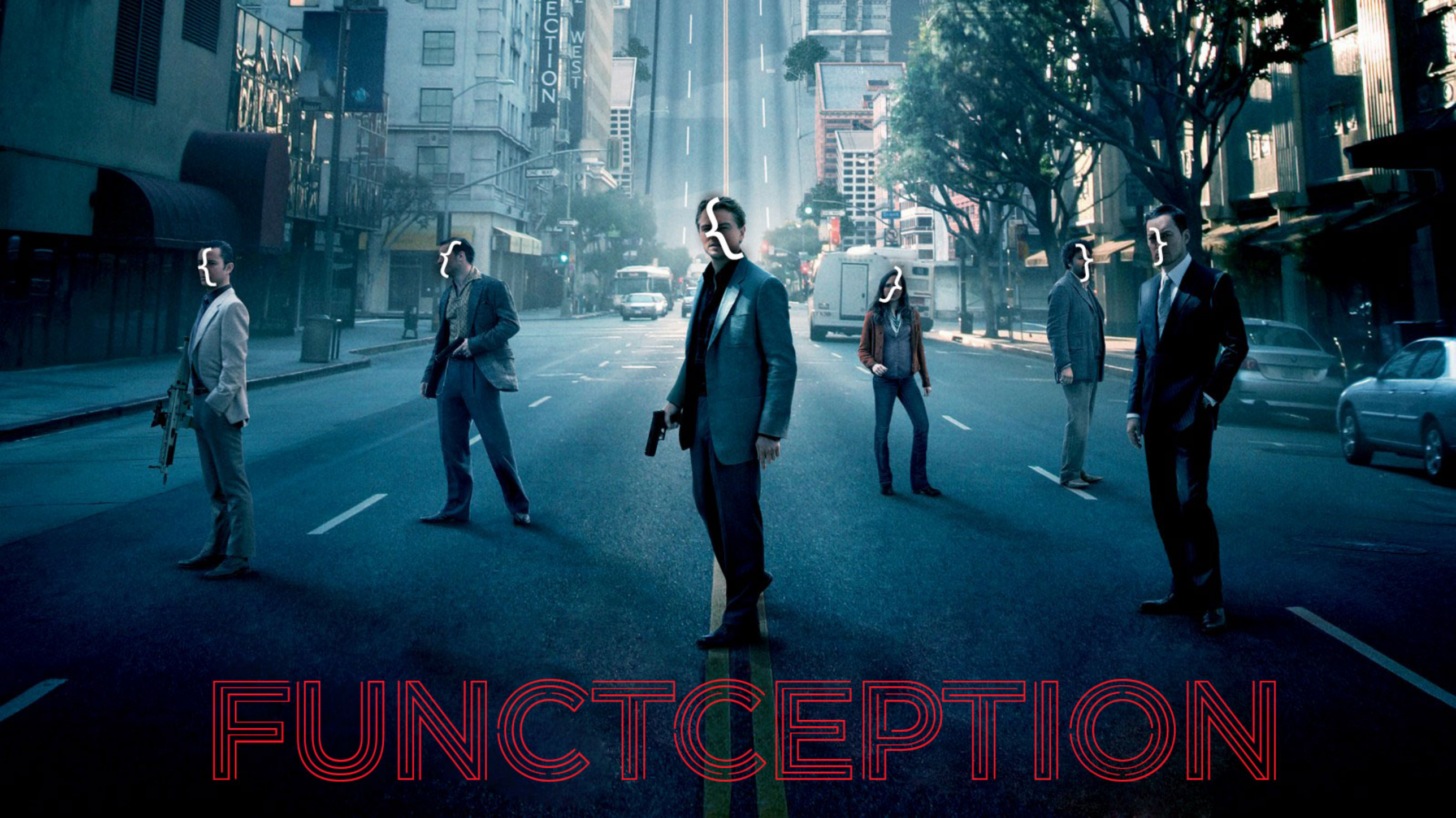
Error handling

```
function('a', 'b', function (err, result) {
  if(err) { // 😕
    console.error('Error occurred: ', err);
    return;
  }
  console.log(result);
});
```

Callbacks

Callback hell

```
doSomething('a', function (err, result) {
  …
  doSomethingElse('b', function (err, theOtherResult) {
    …
    yetAnotherOne(function(err, finalResult) {
      😓
    });
  });
});
```


FUNCTCEPTION

Callbacks

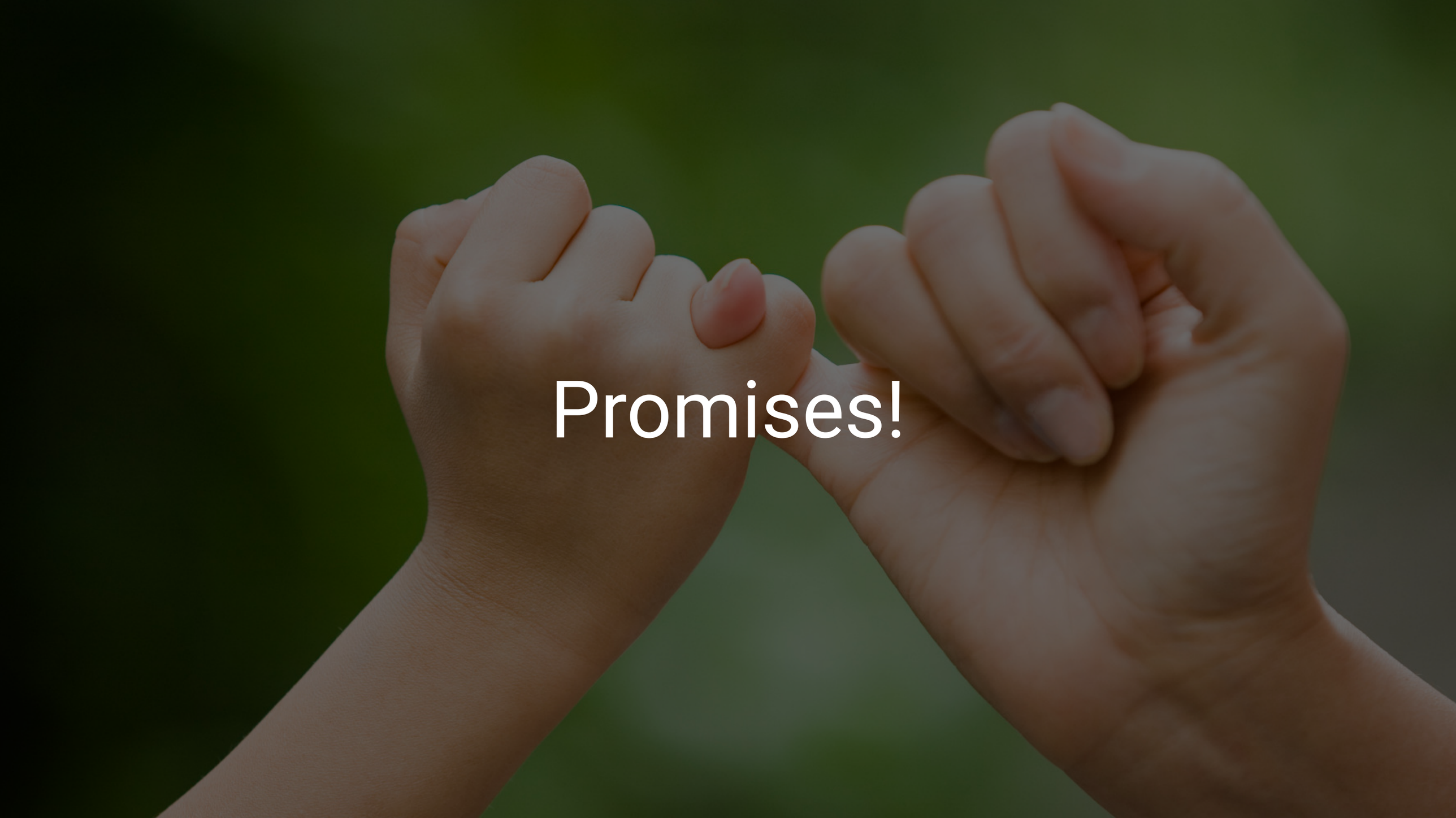
Callback hell

```
doSomething('a', doSomethingResult);

function doSomethingResult (err, result) {
  …
  doSomethingElse('b');
}
```

Callbacks

Callback hell

```
doSomething('a', doSomethingResult(doSomethingElse));

function doSomethingResult (chainedFn) {
  …
  return function (err, result) {
    chainedFn('b');
  }
}
```


Promises!

Promises

```
function foo (bar) {
  return new Promise((resolve, reject) => {
    if(succeed) {
      resolve(data)
    } else {
      reject(error)
    }
  })
}

foo('a').then(data => console.log(data))
```

Promises

Error handling

```
foo('a')
.then(data => console.log(data))
.catch(error => console.error(error)); 😊
```

Promises

Promises hell

```
foo('a')
.then(data => {
  processDataAsync(data)
  .then(processedData => {
    yetAnotherAsyncFn(processedData)
    .then(moreData => {
      console.log()
    })
  })
})
```

Promises

Chaining flow

```
foo('a')
.then(data => anotherPromise(data))
.then(processedData => data.syncValue)
.then(value => console.log(value))
.catch(error => console.error(error));
```

ES7 Async / Await

async / await

async generator

```
const foo = async () => {
  return 42;
}

// is actually
const foo = () => {
  return new Promise((resolve, reject) => {
    resolve(42);
  })
}
```

async / await

await keyword

```
const foo = async () => {
  …
  const bar = await doSomethingAsync();
  …
  return whatever;
}
```

async / await

Error handling

```
const foo = async () => {
  try {
    return await doSomethingAsync()
  } catch (e) {
    return Promise.reject(e)
  }
}

foo
.then(result => console.log(result))
.catch(error =>  console.error(error));
```

It's cool, but it isn't magic! 💫

Let's play with it!