

JS Overview

**JavaScript has
changed a lot in
recent years...**

ES2015 and beyond

ES6 became ES2015

**And JS is now updated
yearly.**

ES2015

every feature under the sun

ES2016

`Array.prototype.includes`, `x ** y`

ES2017

`async functions`, `Object.values()`, `Object.entries()`, trailing commas
in function definitions, string padding

ES2018

object rest/spread, regex features, Promise.prototype.finally

**New features go
through a 4 stage
process**

Stage 0

Strawman: anyone can propose an idea.

Stage 1

Proposal: formal proposal for a feature, someone as a champion to lead the proposal.

http://exploringjs.com/es2016-es2017/ch_tc39-process.html

Stage 2

Draft: something that's starting to look like a proper proposal. Two experimental implementations are required.

Stage 3

Candidate: mostly finished proposal, at least two compliant implementations exist in browsers.

http://exploringjs.com/es2016-es2017/ch_tc39-process.html

Stage 4

Finished! Proposal will be included in the next version of JS.

http://exploringjs.com/es2016-es2017/ch_tc39-process.html

Arrow functions

```
const adder = function(x, y) {  
  return x + y;  
}
```

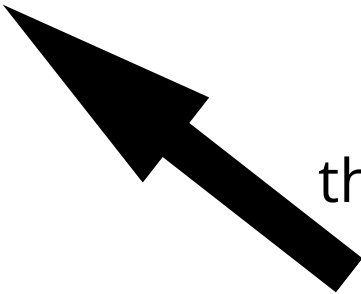
```
const adder = (x, y) => {  
  return x + y;  
}
```

you can omit the `return` if the arrow function is missing braces round the body

```
const adder = (x, y) => x + y
```

Arrow functions

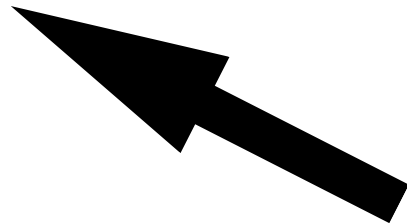
```
const data = {  
  person: 'jack',  
  friends: ['alice', 'bob'],  
  log() {  
    this.friends.forEach(function(name) {  
      console.log(this.person, 'has friend', name)  
    })  
  },  
}
```



this will error

Arrow functions

```
const data = {  
  person: 'jack',  
  friends: ['alice', 'bob'],  
  log() {  
    this.friends.forEach(function(name) {  
      console.log(this.person, 'has friend', name)  
    }.bind(this))  
  },  
}
```



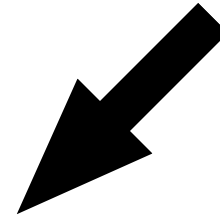
this fixes it

Arrow functions

// arrow functions are bound to the same scope always

```
const data = {  
  person: 'jack',  
  friends: ['alice', 'bob'],  
  log() {  
    this.friends.forEach((name) => {  
      console.log(this.person, 'has friend', name)  
    })  
  },  
}
```

this fixes it too!



Arrow functions

open the console!

```
npm run exercise js-overview 1
```

classes

```
class Person {  
    constructor(name) {  
        this.name = name  
    }  
}  
  
const jack = new Person('jack')
```


classes

```
class Person {  
    constructor(name) {  
        this.name = name  
    }  
  
    changeName(newName) {  
        this.name = newName  
    }  
}  
  
const jack = new Person('jack')  
  
jack.changeName('bob')
```

classes

```
npm run exercise js-overview 2
```

proposal-class-fields

stage 3 proposal

// before:

```
Person.foo = 'bar'
```

// after:

```
class Person {  
  static foo = 'bar'  
}
```

proposal-class-fields

stage 3 proposal

```
class Person {  
  constructor() {  
    this.foo = 'bar'  
  }  
}
```

// after:

```
class Person {  
  foo = 'bar'  
}
```

npm run exercise js-overview 3

object rest spread

stage 4 proposal - included in ES2018!

// before:

```
const team = { team: 'newcastle' }
```

```
const newObj = Object.assign({ name: 'jack' }, team)
```

```
newObj // { name: 'jack', team: 'newcastle' }
```

object rest spread

stage 4 proposal - included in ES2018!

```
// after:
```

```
const team = { team: 'newcastle' }  
const newObj = {  
  name: 'jack',  
  ...team,  
}
```

```
newObj // { name: 'jack', team: 'newcastle' }
```

```
npm run exercise js-overview 4
```

promises

we'll dive into this more later, but here's a quick start...

```
// the fetch API:  
  
fetch( '/api' ).then(response => {  
  ...  
})
```

promises

a function can return a promise that will *resolve* with some value at a later point in time.

```
// the fetch API:  
  
fetch( '/api' ).then(response => {  
  ...  
})
```


promises

with each `then`, you can return a new value (which will always be wrapped in a promise).

```
// the fetch API:  
  
fetch('/api').then(response => {  
  // parse the response as json()  
  return response.json()  
}).then(data => {  
  // data here is the JSON response  
})
```

promises

you can use `Promise.resolve` to create a promise

```
Promise.resolve('foo').then(data => {  
  return data + 'bar';  
}).then(data => {  
  console.log('Final data', data) //foobar  
})
```

```
npm run exercise js-overview 5
```

template literals

a much easier way to insert data into strings

// before:

```
const firstName = 'Jack'
```

```
console.log('Hi, my name is ' + firstName)
```

// after:

```
const firstName = 'Jack'
```

```
console.log(`Hi, my name is ${firstName}`)
```

no exercise here! Lucky you.

destructuring

```
const person = { name: 'jack', team: 'newcastle' }
```

```
// BEFORE:
```

```
const name = person.name
```

```
const team = person.team
```

```
// AFTER:
```

```
const { name, team } = person
```

```
const person = { name: 'jack', team: 'newcastle' }
```

```
// you can even set defaults if a value is missing:
```

```
const { colour = 'blue' } = person
```

```
const person = { name: 'jack', team: 'newcastle' }
```

```
// NOTE: this uses the object-rest-spread proposal
```

```
// BEFORE:
```

```
const name = person.name
```

```
// how to get all keys apart from name?
```

```
// AFTER:
```

```
const { name, ...rest } = person
```

```
rest === { team: 'newcastle' }
```

```
// and it works for arrays
```

```
const people = [ 'alice', 'bob', 'charlie' ]
```

```
const [ first, second ] = people
```

```
const [ first, ...others ] = people
```

```
// others === [ 'bob', 'charlie' ]
```


get destructuring!

```
npm run exercise js-overview 6
```

Any questions?

Let's take a break before moving onto React!