MODERN JAVASCRIPT

- John Knoll
- jpknoll@ucdavis.edu
- CAES
- @jpknoll

BROWSER SUPPORT

https://kangax.github.io/compat-table/es6/

Install Babel

```
npm install --save-dev babel-loader babel-core babel-preset-env
```

Configure Babel

```
{
  "presets": [
    ["env", { "targets": { "browsers": [ "> 1%" ] } } ]
  ]
}
```

Configure Webpack

```
const config = {
  module: {
    rules: [
      { test: /\.js$/, exclude: /node_modules/, loader: "babel-loader" }
    ]
  }
}
```

SCOPING

Block-scoped binding constructs. let is the new var. const is single-assignment. Static restrictions prevent use before assignment.

```
function f() {
  {
    let x;
    {
      // okay, block scoped name
      const x = "sneaky";
      // error, const
      x = "foo";
    }
    // error, already declared in block
    const x = "inner";
  }
}
```

VARIABLE DECONSTRUCTION

Simple way to decompose objects/arrays into variables.

```
const obj = {
  foo: 'bar',
  boo: 'far'
};

// Pull `foo` and `boo` from `obj`
const { foo, boo } = obj;

// Same as:
// foo = obj.foo;
// bar = obj.bar;

console.log(foo, boo);
>'bar'
>'far'
```

```
var myArray = [11, 12, 13, 14, 15];

var a, b;

// Pull values from `myArray` into `a` and `b`
[a, b] = myArray;

// Same as:
// a = myArray[0];
// b = myArray[1];

console.log(a, b)
> 11
> 12
```

SPREAD OPERATOR

Collect values and expand iterators

```
const obj = {
  foo: 'bar',
  boo: 'far'
};

const wat = {
    something: 'here',
    // Expand properties of obj into this object
    ...obj
}

console.log(wat);
> {
>   something: 'here',
>   foo: 'bar',
>   boo: 'far'
> }
```

```
const others = [9, 8, 7];

// Expand `others` into this array
const all = [1, 2, 3, ...others, 4, 5];

console.log(all)
> [1, 2, 3, 9, 8, 7, 4, 5]
```

REST OPERATOR

Dump everything else

```
const obj = {
  foo: 'bar',
  boo: 'far',
  baz: 'bat'
};

// Collect remaining props into an `others` object
const { foo, ...others } = obj;

// Collect entries after the first two into an `others` array
const [first, second, ...others] = [1,2,3,4,5];

console.log(others)
> [3, 4, 5]

// Collect any params after the first two in the `others` array
function test( first, second, ...others ) {
  // No more `var others = Array.prototype.slice.call(arguments, 2)`
  console.log(first, second, others);
}
```

ARROW FUNCTIONS

```
function sum(a, b) {
    return a + b;
}

// anonymous function
const magic = (a, b) => {
    return a + b;
};

const black_magic = (a, b) => a + b;
```

```
// great for callbacks

setTimeout(() => {
  // do something later
  console.log('literally a second');
}, 1000);
```

DEFAULT PARAMS

```
function say_hello(name = 'bob') {
    console.log('hello ' + name);
}

say_hello('john');
> 'hello john'

say_hello();
> 'hello bob'
```

STRING INTERPOLATION

```
const size = 'huge';
const feels = 'awesome';

"This feature is " + feels + "! It's going to be " + size + "!";

`This feature is ${feels}! It's going to be ${size}!`;

const amazing =
`It also works on
multiline strings`
```

ARRAY FUNCTIONS

```
const ar = [1, 2, 3, 4];

// loops, but better
ar.forEach((value) => {
  console.log(value);
});

// get odds
const odds = ar
  .filter((value) => value % 2);

// square
const square = ar
  .map((value) => value * value);

// sum
const sum = ar
  .reduce((prev, curr, i) => curr + ar[i], 0);
```

```
// also known as 'any'
const any = ar
  .map((value) => value * 3)
  .some((value) => value > 10);

// also known as 'all'
const all = ar
  .map((value) => value * 2)
  .every((value) => value < 10);
```

```
const found = ar
  .find((value) => value === 3);

const foundIndex = ar
  .findIndex((value) => value === 3);
```

PROMISES

Now with real browser support!

```
// now without a polyfill!
const p = new Promise((r, x) => {
  setTimeout(() => {
    r('success');
  }, 2000);
  console.log('started');
});

p.then((result) => {
    console.log(result);
  })
  .error((ex) => {
    console.error(ex);
  });
```

FETCH API

```
import 'isomorphic-fetch';

fetch('https://myapi.com')
  .then(response => {
    if (response.ok) {
      return response.json();
    }

    throw new Error('Network ERROR!');
  })
  .then(result => {
    console.log(result);
  })
  .catch(err => {
    console.error(err);
  });
```

```
import 'isomorphic-fetch';

var form = new FormData(document.getElementById('login-form'));

fetch('https://myapi.com', {
  method: 'POST',
  body: form
})
  .then(response => {
    if (response.ok) {
      return response.json();
    }

    throw new Error('Network ERROR!')
  })
  .then(result => {
    console.log(result);
  })
  .catch(err => {
    console.error(err);
  });
```

ASYNC/AWAIT

```
async function getData() {
  try {
    const response = await fetch('https://www.myapi.com/');
    if (!response.ok) {
      throw new Error("Bad response");
    }

    return await response.json();
  }
  catch(ex) {
    console.error(ex);
  }
}

getData()
  .then(data => console.log(data));
```

ES6 MODULES

```
const someLib = require('some-lib')

import someLib, { somefunc } from 'some-lib';

import { default as FooLib } from 'some-lib';

import * as FooLib from 'some-lib';
```

```
export default 123;

const value = 123;
export default value;

export function MyFunction() {
}

export class MyClass {
}
```