WebPack

Practice: https://blog.revillweb.com/using-es2015-es6-modules-with-babel-6-3ffc0870095b

Tutorial: http://blog.andrewray.me/webpack-when-to-use-and-why/

ES6 Guidelines & Tutorials

Nelson from Udemy

- https://github.com/djdjalas/es6-course

javaScript Patterns for 2017

- https://www.youtube.com/watch?v=hO7mzO83N1Q&t=294s

ES6 CheatSheet

- https://es6cheatsheet.com/
- https://gist.github.com/vasco3/22b09ef0ca5e0f8c5996

Babel Installation

Running a Babel 6.x project using npm version 2 can cause performance problems because of the way npm installs dependencies. This problem can be eliminated by either switching to npm version 3.

```
> npm install --save-dev babel-cli babel-preset-env
```

- https://babeljs.io/
- https://stackoverflow.com/a/34747852

- Babel has support for the latest version of JavaScript through syntax transformers. These plugins allow you to use new syntax, right now without waiting for browser support. Check out our env preset to get started.

1. install babel-cli

```
> npm install babel-cli
```

- Note: Since it's generally a bad idea to run Babel globally you may want to uninstall the global copy by running:

  ```
  > npm uninstall --global babel-cli
  ```

1a. Configuring Babel (I have not done this prior to using Babel)

https://babeljs.io/docs/setup/#installation

- Instead of running Babel directly from the command line we're going to put our commands in npm scripts which will use our local version. Add following entry to package.json:

```
"scripts": {"build": "babel src -d lib"}
```

- From the terminal run command:

```
> npm run build
```

2. install preset and create/modify .babelrc configuration file

- Create a .babelrc config in your project root. You can use the env preset, which enables transforms for ES2015+:

```
> npm install --save-dev babel-preset-env
```

- In order to enable the preset you have to define it in your .babelrc file:

```
{"presets": ["env"]}
```

3. run babel on original es6 js file

Using Babel as npm script

```
> npm run babel -- original.js --out-file bundle.js -w --source-maps

-w for watching file changes
--source-maps for debugging in the browser
```

https://stackoverflow.com/questions/34747693/how-do-i-get-babel-6-to-compile-to-es5-javascript?answertab=active#tab-top

4. polyfill

```
> npm install --save-dev babel-polyfill
```

babel-preset-env

http://2ality.com/2017/02/babel-preset-env.html

babel-preset-env: a preset that configures Babel for you

- babel-preset-env is a new preset that lets you specify an environment and automatically enables the necessary plugins.

Node.js

- If you compile your code for Node.js on the fly via Babel, babel-preset-env is especially useful, because it reacts to the currently running version of Node.js if you set the target "node" to "current":

Asynchronous iteration demo

https://github.com/rauschma/async-iter-demo

Axel Rauschmayer

Debugging

- in Chrome DevTools go to Source tab, in Setting ensure Enable JavaScript source map is checked.

Variables and Scoping

In [18]:
```
var a = 1;
{ let b = 2;
}
const C = 3;
```
Out[18]:
```
undefined
```

Constants

- Constants are immutable objects
- You can not assign some other value or object, but you can mutate it through changing of its properties.

In [1]:
```
const imm = "this is immutable string";
```
Out[1]:
```
undefined
```

In [2]:
```
console.log(imm);
imm;
```
```
this is immutable string
```
Out[2]:
```
'this is immutable string'
```

In [3]:
```
const immObj = {};
immObj;
```
Out[3]:
```
{}
```

In [7]:
```
immObj.test = "Test String";
immObj.test;
```
Out[7]:
```
'Test String'
```

In [8]:
```
immObj;
```
Out[8]:
```
{ test: 'Test String' }
```

SPREAD Operator ...

In [16]:

```
var best = [4, 5, 6];
var first = [1, 2, 3];

var all = [1, 2, 3, ...best, 7, 8, 9];
console.log(all);


var showItems = function(...args) {
    for (var x = 0; x < args.length; x++) {
        console.log(args[x]);
    }
}

showItems(...best);
console.log(best);
```

```
[ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
4
5
6
[ 4, 5, 6 ]
```

Out[16]:

```
undefined
```

Destructuring List and Objects

http://exploringjs.com/es6/ch_destructuring.html

DESTRUCTURING ASSIGNMENT

In [20]:

```
var a, b, rest;
[a, b, ...rest] = [1, 2, 3, 4, 5, 6, 7, 8, 9];
console.log(a);
console.log(b);
console.log(rest);
```

```
1
2
[ 3, 4, 5, 6, 7, 8, 9 ]
```

Out[20]:

```
undefined
```

DESTRUCTURING OBJECT

In [29]:

```
var myObj = {a: 1, b: 2}

var {a: c, b: d} = myObj;

console.log(c);
console.log(d);
```

```
1
2
```

Out[29]:

```
undefined
```

Object Shorthand Creation

In [21]:

```
var myObject = {x: "cat", y: "dog"};

console.log(myObject.x);

var a = "1";
var b = "2";
var c = "3";

var newObj = {a, b, c};
console.log(newObj);
```

```
cat
{ a: '1', b: '2', c: '3' }
```

Out[21]:

```
undefined
```

Alex Video on Object Literals, Proto chaining, Constructors

In [38]:

```
var ProtoPerson = {
    field: function() {
        return this.name
    }
};

var Jane = {
    __proto__: ProtoPerson,
    name: "Jane"
};
Jane;
```

Out[38]:

```
{ name: 'Jane' }
```

Arrow Functions

In [43]:

```
var oldWay = function() {
    console.log("Hello world");
};
oldWay();

var newWay = () => {
    console.log("arrow function");
};

var newWay2 = arg2 => {
    console.log(arg2);
};

var newWay3 = (arg3, arg4) => {
    console.log(arg3 + arg4);
};

newWay();
newWay2("2");
newWay3("3", "4");
```

```
Hello world
```

```
arrow function
2
34
```

Out[43]:
```
undefined
```

Template Literals

In [53]:
```
var band = "Maroon5";

var longString = `
this is long string
more long due to ${band}
testing
`;

band = "NotSureWhatsTheirName"
console.log(longString);

var getDescription = (band, period) =>  {
    return `
    this is long string
    more long due to ${band} being around for ${period} years
    testing-123
    `;
}

console.log(getDescription("Matrix", 15));
```

```
this is long string
more long due to Maroon5
testing


    this is long string
    more long due to Matrix being around for 15 years
    testing-123
```

Out[53]:
```
undefined
```

CLASSES

In [1]:
```
class Band {

    constructor(name, origin) {
        this.name = name;
        this.origin = origin;
    }

    printName() {
        console.log("name = ", this.name);
    }

    printOrigin() {
        console.log("origin = ", this.origin);
    }
}
```

```
var bayside = new Band("Bayside", "Queens");

bayside.printName();
bayside.printOrigin();
```

```
name =  Bayside
origin =  Queens
```

Out[1]:

```
undefined
```

In [2]:

```
var bayside = new Band("Bayside", "Queens");

bayside.printName();
bayside.printOrigin();
```

```
name =  Bayside
origin =  Queens
```

Out[2]:

```
undefined
```

SUB-CLASSING and SUPER()

Super() provides more than access to "this", it is used in overrides as a mean of augmenting the base method, it can take arguments.

- When used in constructor() you would pass common arguments up to the parent rather than duplicating the code.

In [11]:

```
class Band {

    constructor(name, origin) {
        this.name = name;
        this.origin = origin;
    }

    printName() {
        console.log("name = ", this.name);
    }

    printOrigin() {
        console.log("origin = ", this.origin);
    }
}


class Member extends Band {

    constructor(name, origin, genre) {
        super(name, origin);
        this.genre = genre;
    }
    printName() {
        console.log("this is an override + ", this.name);
    }
    printGenre() {
        console.log(this.genre);
    }
}
```

```
var bayside = new Member("Bayside", "Queens", "Alternative");
bayside.printName();
bayside.printOrigin();
bayside.printGenre();
```

```
this is an override +  Bayside
origin =  Queens
Alternative
```

Out[11]:

```
undefined
```

SourceMaps allow to debug

```
> npm run babel
```

Default Arguments

NaN - Not a Number

- unique object in JavaScript (not a type)

In [7]:

```
function test(a = 0) {
    console.log(a + 10);
}

test();
```

```
10
```

Out[7]:

```
undefined
```

for() vs forEach() vs map()

- https://ryanpcmcquen.org/javascript/2015/10/25/map-vs-foreach-vs-for.html

For Loop, becoming obsolete in JavaScript

In [20]:

```
var myArray = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'];

for (var x = 0; x < myArray.length; x++) {
    console.log(myArray[x]);
}
```

```
a
b
c
d
e
f
g
h
```

Out[20]:

```
undefined
```

New For Loop

- for () loops should be avoided unless you have determined that there is some necessary benefit they deliver to your end user that no other iteration method is capable of (such as a performance necessity).

```
In [16]:
for (let i of myArray) {
    console.log(i);
}
```

```
a
b
c
d
e
f
g
h
```

```
Out[16]: undefined
```

forEach()

- forEach() method executes provided function over each array element.
- For other than functional programming paradigms (and even in some rare cases within the functional paradigm), .forEach() is the proper choice.

```
In [18]:
myArray.forEach(function(elm) {
    console.log(elm);
});
```

```
a
b
c
d
e
f
g
h
```

```
Out[18]: undefined
```

map()

- You should favor .map() and .reduce(), if you prefer the functional paradigm of programming.

Callback & Higher Order Function

http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/

JavaScript Promises

https://wildermuth.com/2013/08/03/JavaScript_Promises

Compare forEach() with reduce()

In [22]:

```
var task = {
    name: 1,
    level: 2,
    group: 3
};
```

Out[22]:

```
undefined
```

In [30]:

```
var bar = {};
Object.keys(task).forEach(function (prop) {
    bar[prop] = null;
});

var bar_2 = Object.keys(task).reduce(function (newObj, prop) {
    newObj[prop] = null;
    return newObj;
}, {});

console.log(bar);
console.log(bar_2);
```

```
{ name: null, level: null, group: null }
{ name: null, level: null, group: null }
```

Out[30]:

```
undefined
```

Stuck at recreating map() for give forEach()

- https://stackoverflow.com/questions/12482961/is-it-possible-to-change-values-of-the-array-when-doing-foreach-in-javascript

In [46]:

```
var nums = [
    5,
    9,
    7
];

nums.forEach(function (num, index, first) {
    return num + 1;
});


var second = nums.map(function (num) {
    return num + 1;
});

var second = nums.map(function (num) {
    return console.log(num + 1);
});
```

```
console.log(nums);
console.log(first);
console.log(second);


// arr.forEach(function(part, index, theArray) {
//   theArray[index] = "hello world";
// });
```

```
6
10
8
[ 5, 9, 7 ]
undefined
[ undefined, undefined, undefined ]
```

Out[46]:
```
undefined
```

In [52]:
```
var first = {};
nums.forEach((o, i, first) => first[i] = o + 1);
first;
```

Out[52]:
```
{}
```

Modules

- check if browser supports modules https://paulirish.github.io/es-modules-todomvc/

Study Later on

- great discussion on comments: https://css-tricks.com/how-do-you-structure-javascript-the-module-pattern-edition/
- https://jakearchibald.com/2017/es-modules-in-browsers/
- webpack: http://blog.andrewray.me/webpack-when-to-use-and-why/
- modules: https://medium.freecodecamp.org/javascript-modules-a-beginner-s-guide-783f7d7a5fcc

Creating Classes in JavaScript withuot *class* keyword - Closure

- Example taken from TypeScript training on Pluralsight

In [2]:
```
// JavaScript creating Class without class keyword
var Greeter = (function () {
    function Inner(message) {

        this.greeting = message;
    }

    Inner.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };

    return Inner;
})();

var aaa = new Greeter("world");
console.log(aaa.greet());
```

```
Hello, world
```

Out[2]:
```
undefined
```

In [10]:
```
// ES6 class keyword
class Greeter_3 {
    constructor(message) {
        this.greeting = message;
    }

    greet() {
        return "Hello, " + this.greeting;
    }
}

var bbb = new Greeter_3("Biljana");
console.log(bbb.greet());
```

```
Hello, Biljana
```

Out[10]:
```
undefined
```