INTRODUCTION TO

ES6

IN THIS TALK

- modules
- let & const
- enhanced object literal
- Set and Map
- for … of loop
- string templates
- the fat arrow =>
- spread operator …
- destructuring
- generators
- promises

WHAT IS ES6

- ES6 aka ECMAScript 2015 adds new syntax to Javascript
- Finalized in 2016

ES5 VS ES6

- ES5 has browser support
- ES6 must be transpiled with tools such as Babel. Babel will convert ES6 into ES5. Browser support is sparse.
- Node.js support. Check http://node.green/ for feature availability

CONST

- read-only
- block scoped
- cannot be redeclared
- object keys are not protected

```
const PI = 3.1416;

PI='tasty'; //will throw error
```

http://jsbin.com/wububi/edit?js,console

CONST

- values are not immutable, only the binding is immutable
- to make an object immutable, use Object.freeze()

```
const animal = { type: ‘dog’ };
animal.type = ‘cat'; // no error thrown
delete animal.type; // no error thrown

const beast = Object.freeze({type:’bear'});
beast.type = ‘fox'; // silently fails
```

http://jsbin.com/wububi/edit?js,console

LET

- block scoped
- cannot be redeclared

ENHANCED OBJECT LITERAL

- short hand

```
{                          =          {
  name,                                   name: name,
  address                                 address: address
}                                      }
```

ENHANCED OBJECT LITERAL

- computed values

```
{                                          {
  name: 'Bob',                               name: name,
  ['address' + 1]: '1st St'          =       address1:'1st St'
}                                          }
```

ENHANCED OBJECT LITERAL

- method definition shorthand

```
{
  name: 'Bob',
  sayHello() {
    console.log('hello');
  }
}
```

=

```
{
  name: name,
  sayHello: function() {
    console.log('hello');
  }
}
```

STRING TEMPLATES

- specify with in backquotes
- line breaks

```
console.log(`string text line 1
string text line 2`);
```

=

```
console.log('string text line 1\n' +
'string text line 2');
```

STRING TEMPLATES

- enclosed within back ticks `
- line breaks

```
console.log(`string text line 1
string text line 2`);
```

=

```
console.log('string text line 1\n' +
'string text line 2');
```

http://jsbin.com/helolug/edit?js,console

STRING TEMPLATES

- embedded expressions with ${ }. no need for string concatenation with +

```
`an hour = ${ 60 * 60 } secs`        =        'an hour = 3600 secs'
```

http://jsbin.com/helolug/edit?js,console

STRING TEMPLATES

- Nested templates.

```
let t = 14;
`it costs ${ t < 100 ? `${t}c` : `$${t/100}`}`          'it costs 14c'

                                                        =
t = 140;
`it costs ${ t < 100 ? `${t}c` : `$${t/100}`}`          'it costs $1.4'
```

http://jsbin.com/helolug/edit?js,console

FOR ... OF LOOP

- Iterate over arrays, sets, maps and iterables

https://jsbin.com/vozifin/edit?js,console

THE FAT ARROW =>

- short hand
- single argument arrow functions do not need ()

```
(x) => x * x;

(x) => { return x * x; }

 x  => x * x;
```

```
function(x) {
  return x * x;
}
```

```
() => 3.14
```

```
function() {
  return 3.14;
}
```

http://jsbin.com/cumaqec/edit?js,console

THE FAT ARROW =>

- quite useful within map/filter/reduce

```
const odds = [1, 3, 5, 7, 9];

let lessThan7= odds.filter(num => num < 7);

let evens = odds.map(num => num + 1);
```

http://jsbin.com/cumaqec/edit?js,console

THE FAT ARROW =>

- no ‘this’. refers to the ‘this’ outside the arrow function

```
function Person(){
  this.age = 0;

  setInterval(() => {
    this.age++; // properly refers to the person object
  }, 1000);

}
```

http://jsbin.com/cumaqec/edit?js,console

THE FAT ARROW =>

- returning objects

```
() => { name: 'bob' }; // will not work

() => ({ name: 'bob' });
```

http://jsbin.com/cumaqec/edit?js,console

SPREAD OPERATOR ...

- for arrays

http://jsbin.com/hacaxaj/edit?js,console

```
const list = [1, 2, 3];

let newList = [...list];
```

SPREAD OPERATOR ...

- concatenate two arrays

http://jsbin.com/hacaxaj/edit?js,console

```
const list1 = [1, 2, 3];
const list1 = [7, 8, 9];

let newList = [...list1, ...list2]; // [1,2,3,7,8,9]
```

SPREAD OPERATOR ...

- use to clone or merge objects

http://jsbin.com/hacaxaj/edit?js,console

```
var obj1 = { foo: 'bar', x: 42 };

var obj2 = { foo: 'baz', y: 13 };


var clonedObj = { ...obj1 }; // Object { foo: "bar", x: 42 }


var mergedObj = { ...obj1, ...obj2 }; // Object { foo: "baz", x: 42, y: 13 }
```

DESTRUCTURING ASSIGNMENT – ARRAYS

- unpack values into distinct variable

http://jsbin.com/cubetom/edit?js,output

```
//assign
let [a, b] = [1, 4]; // a = 1, b = 4

//defaults
[a = 3, b = 8] = [1]; // a = 1, b = 8

//swap
[a, b] = [b, a]; // a = 4, b = 1;

//ignore values
[a, ,b] = [1, 2, 3]; // a = 1, b = 3;
```

DESTRUCTURING ASSIGNMENT – ARRAYS

http://jsbin.com/cubetom/edit?js,output

```
//assign remaining
let [a, ...b] = [1, 2, 3, 4, 5]; // a = 1, b = [2, 3, 4, 5]
```

DESTRUCTURING ASSIGNMENT – OBJECTS

- unpack and assign remaining
- use () during assignment without declaration

http://jsbin.com/cubetom/edit?js,output

```
//assign remaining
let {a, b} = {a:1, b:2}; // a = 1, b = 2

//defaults. note the ()
({a = 3, b = 8} = {a:1}); // a = 1, b = 8

//assign to new variable
({a:aa, b:bb} = {a:1, b:7}); // aa = 1, bb = 7

//assign to new variable with defaults
({a:aa = 3, b:bb = 8} = {a:1}); // aa = 1, bb = 8
```

DESTRUCTURING ASSIGNMENT – FUNCTION ARGS

- assign to {} to allow the function to be called without any args

http://jsbin.com/geselim/edit?js,console

```
function drawCircle({radius=10, center={x:0,y:0}} = {}) {
// draw the circle with radius and center
}

drawCircle() // = drawCircle(10, {x:0, y:0})

drawCircle(3, {x:2, y:2});
```

DESTRUCTURING AND REST

http://jsbin.com/geselim/edit?js,console

```
function head([ head, ...tail ]) {
  return head;
}

function tail([ head, ...tail ]) {
  return tail;
}

// or with arrow function syntax

const head = ([ head, ...tail ]) => head;

const tail = ([ head, ...tail ]) => tail;
```

GENERATORS – *FUNCTION

- function* defines a generator function
- returns a Generator object.
- you can enter and exit a generator without losing its internal state

http://jsbin.com/pezamup/edit?js,console

GENERATORS – *FUNCTION

- generator returns an object with two properties:
  - value – the return value
  - done – true/false based on whether the generator is done or not.

http://jsbin.com/pezamup/edit?js,console

ITERATORS & ITERABLES

- Iterators have a function next() that returns
  - {
    value: <return value>
    done: <true/false. Are there any more values?>
    }
- A generator is has an iterator

https://jsbin.com/nozemiq/edit?js,console

ITERABLE

- Iterables have a @@iterator method that returns an iterator
  - {
    value: <return value>
    done: <true/false. Are there any more values?>
    }
- A generator is also an iterable
- You can loop through iterable using for ... in.

https://jsbin.com/yitayup/edit?js,console

WHY AREN'T WE COVERING CLASSES?

- Learn at your own risk
- Trying to do Java like OOP in JavaScript is risky
- Object literals are good enough in many case
- Use Factories
- Favor Composition over Inheritance
- Read up Eric Elliot's blogs on OOP

WHAT NEXT