ESNext

Agenda

- const & let
- Block Scoping
- Arrow Notation
- This Binding
- Classes
- Class Inheritance
- Decorators
- Modules
- Template Strings
- for..of Iterator
- Default, Rest & Spread
- Restructuring
- Sets & Maps
- WeakSets & WeakMaps
- Native Promises
- Generators & Async Await

Const

```
var pi = 3.141592653;

// is now
const pi = 3.141592653;
```

Let - Block Scoping

```
// this works. but not good practice
for (var i = 0; i < 10; i++) {
  console.log(i);
}

console.log(i);
```

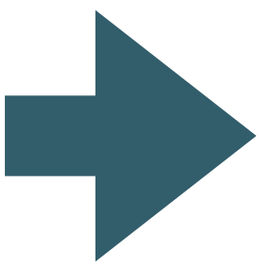
```
// this does not work :)
for (let i = 0; i < 10; i++) {
  console.log(i);
}

console.log(i);
```

i undefined

Arrow Notation

```
const numbers1 = [1, 2, 3];
```

```
const numbers2= numbers1.map(function (value) {
  return value + 1;
});
```



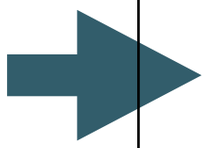
```
const numbers2 = numbers1.map(a => a + 1);
```

```
console.log(numbers1);
console.log(numbers2);
```

[1, 2, 3]
[2, 3, 4]

This Binding

this.name
undefined in inner scope



```
function problem() {
  this.name = 'WIT';
  setInterval(function () {
    console.log(this.name);
  });
}

const a = new problem();
```

common idiom for accessing this from inner function scope

```
function old() {
  this.name = 'WIT';
  const _this = this;
  setInterval(function () {
    console.log(_this.name);
  });
}

const b = new old();
```

Arrow Functions - Lexical This

shares same **this** with surrounding code

```
function good() {
  this.name = 'WIT';
  setInterval(() => {
    console.log(this.name);
  });
}

const c = new good();
```

Classes

```
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getFullName() {
    return this.firstName + ' ' + this.lastName;
  }
}
```

Class Inheritance

Built-in classes like Array, Date and DOM Elements can be subclassed

```
class Developer extends Person {
  // static method called with Developer.curse();
  static curse() {
    return 'thou shalt forever be off by one...';
  }

  constructor(firstName, lastName, isRemote) {
    super(firstName, lastName);
    this._isRemote = isRemote;
  }

  // getter, used via developerInstance.isRemote
  get isRemote() {
    return this._isRemote;
  }

  // setter, used via developerInstance.isRemote = false
  set isRemote(newIsRemote) {
    throw new Error('Cannot re-assign isRemote!');
  }
}
```

Decorators

```
@isTestable(true)
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  @readonly
  getFullName() {
    return this.firstName + ' ' + this.lastName;
  }
}
```

- Decorators are annotations which allow you to define cross-cutting modifications to classes and methods.

- Decorators are executed at runtime

Modules

- Module syntax a native part of the language

```
// lib/math.js
export function sum(x, y) {
  return x + y;
}
export var pi = 3.141593;
```

```
// app.js
import * as math from "lib/math";
alert("2π = " + math.sum(math.pi, math.pi));
```

```
// otherApp.js
import {sum, pi} from "lib/math";
alert("2π = " + sum(pi, pi));
```

Template Strings

old

```
function getFullName() {
  return this.firstName + ' ' + this.lastName;
}
```

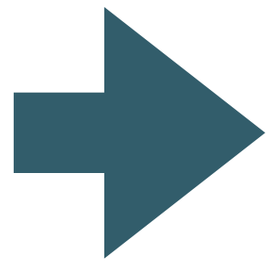
new

```
function getFullName() {
  return `${this.firstName} ${this.lastName}`;
}
```

`backtick` quotation mark
can span multiple lines

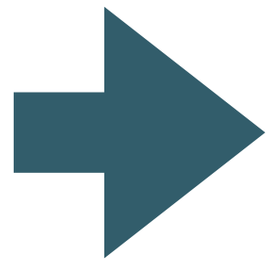
for..of Iterator

```
let a = ['a', 'b', 'c'];
for (let i in a) {
  console.log(i);
}
```



0
1
2

```
for (let i of a) {
  console.log(i);
}
```



a
b
c

Can use the Iterator protocol in your own functions and classes to make anything iterable via for...of

Default, Rest & Spread

default arguments

```
function f(x, y = 12) {
  return x + y;
}
```

```
f(3) == 15
```

rest parameters

```
function f(x, ...y) {
  // y is an Array
  return x * y.length;
}
```

```
f(3, "hello", true) == 6
```

spread operator

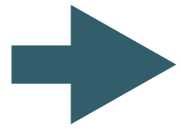
```
function f(x, y, z) {
  return x + y + z;
}
```

```
f(...[1, 2, 3]) == 6
```

Destructuring

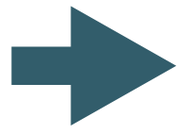
```
let a, b, rest;
```

```
[a, b] = [1, 2]
{a, b} = {a:1, b:2}
```



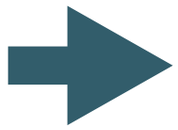
```
// a === 1, b === 2
```

```
[a, b, ...rest] = [1, 2, 3, 4, 5];
```



```
// a === 1, b === 2,
// rest === [3,4,5]
```

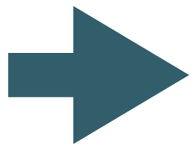
```
function f() {
  return [1,2];
}
[a, b] = f();
```



```
// a === 1, b === 2
```

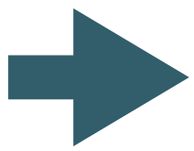
Sets & Maps

```
const s = new Set();
s.add("hello").add("goodbye").add("hello");
```



```
s.has("hello") === true
s.size === 2
```

```
const m = new Map();
m.set("hello", 42);
m.set("goodbye", 34);
```



```
m.get("goodbye") == 34
```

WeakSets & WeakMaps

```
const obj = {
  // ...
}
const wm = new WeakMap();
wm.set(obj, 42)
```

- Key must be an object
- Unless held else where, objects in collection may be disposed of by garbage collector

Native Promises

```
const p = new Promise((resolve, reject) => {
  setTimeout(() => {
    Math.random() < 0.5 ? resolve() : reject();
  }, 500);
});

p.then(() => {
  console.log('Resolved!');
}).catch(() => {
  console.log('Rejected!');
});
```

To Come…

- Generators
- Async Await

ECMAScript 6 — New Features: Overview & Comparison

Constants
- Constants

Scoping
- Block-Scoped Variables
- Block-Scoped Functions

Arrow Functions
- Expression Bodies
- Statement Bodies
- Lexical `this`

Extended Parameter Handling
- Default Parameter Values
- Rest Parameter
- Spread Operator

Template Literals
- String Interpolation
- Custom Interpolation
- Raw String Access

Extended Literals
- Binary & Octal Literal
- Unicode String & RegExp Literal

Enhanced Regular Expression
- Regular Expression Sticky Matching

Enhanced Object Properties
- Property Shorthand
- Computed Property Names
- Method Properties

Destructuring Assignment
- Array Matching
- Object Matching, Shorthand Notation
- Object Matching, Deep Matching
- Parameter Context Matching
- Fail-Soft Destructuring

Modules
- Value Export/Import
- Default & Wildcard

Classes
- Class Definition
- Class Inheritance
- Class Inheritance, From Expressions
- Base Class Access
- Static Members
- Getter/Setter

Symbol Type
- Symbol Type
- Global Symbols

Iterators
- Iterator & For-Of Operator

Generators
- Generator Function, Iterator Protocol
- Generator Function, Direct Use
- Generator Matching
- Generator Control-Flow
- Generator Methods

Map/Set & WeakMap/WeakSet
- Set Data-Structure
- Map Data-Structure
- Weak-Link Data-Structures

Typed Arrays
- Typed Arrays

New Built-In Methods
- Object Property Assignment

Classes
Class Definition

More intuitive, OOP-style and boilerplate-free classes.

ECMAScript 6 — syntactic sugar: reduced | traditional

```
class Shape {
    constructor (id, x, y) {
        this.id = id
        this.move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
}
```

ECMAScript 5 — syntactic sugar: reduced | traditional

```
var Shape = function (id, x, y) {
    this.id = id;
    this.move(x, y);
};
Shape.prototype.move = function (x, y) {
    this.x = x;
    this.y = y;
};
```

http://es6-features.org/

"A good programming language is a conceptual universe for thinking about programming." — Alan J. Perlis

See how cleaner and more concise your JavaScript code can look and start coding in ES6 now !!