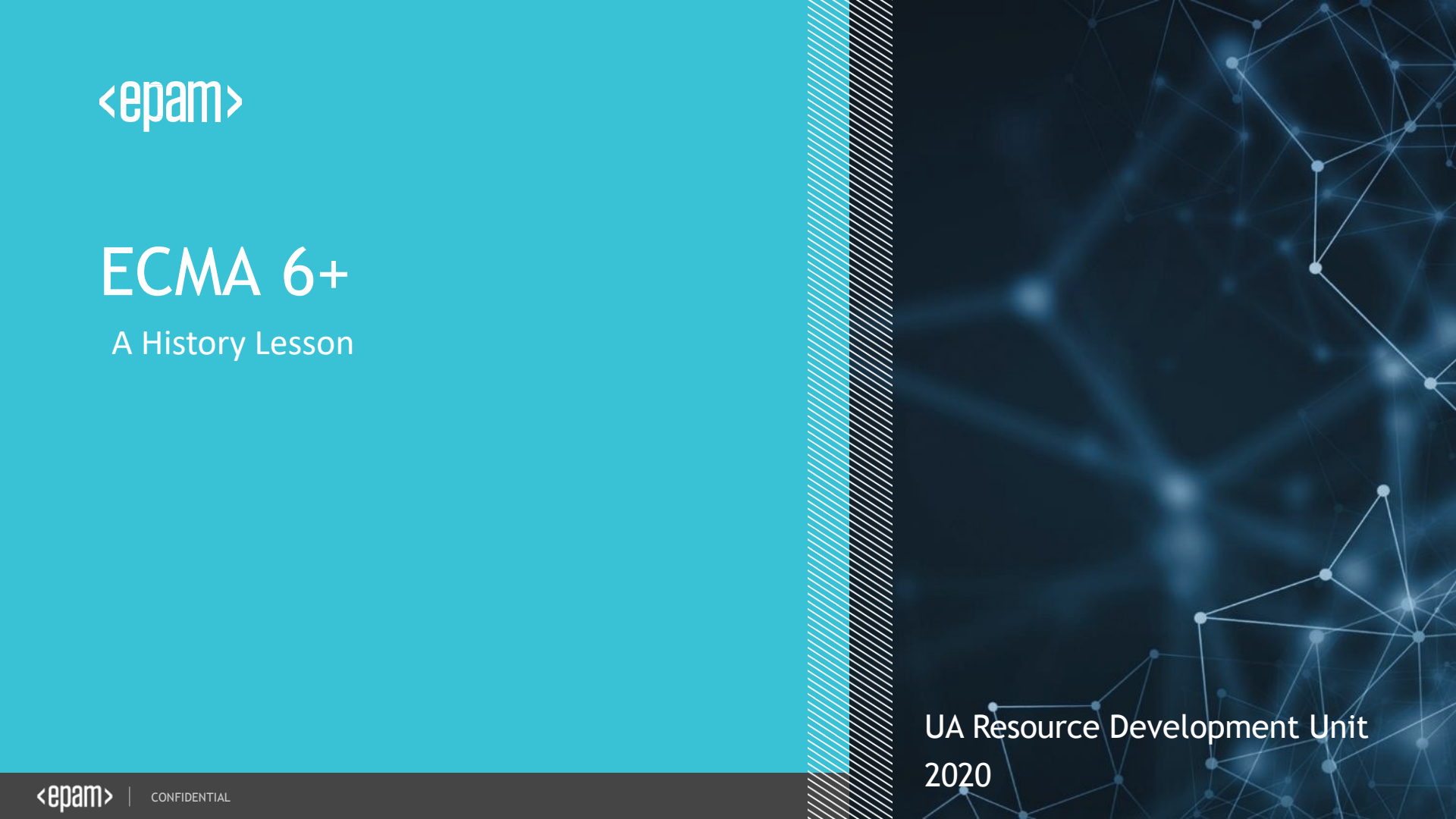
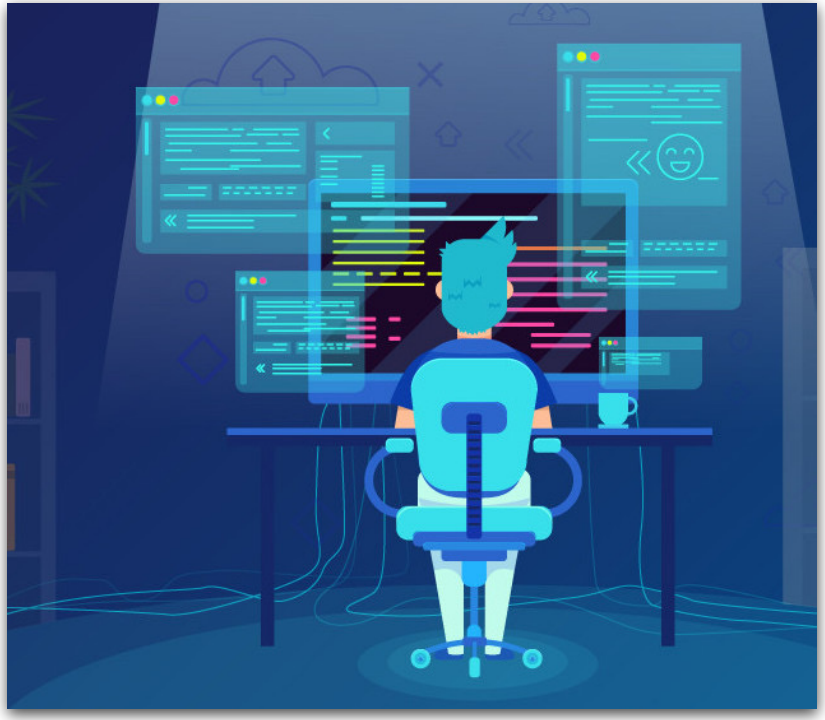

ECMA 6+
A History Lesson
UA Resource Development Unit
2020

AGENDA

1. ES Features
2. Arrow Functions
3. Spread operators
4. Promises
5. Proxy, Generators



ECMA-262 timeline

- June 1997 - First edition of ECMAScript
- August 1998 - ES2, to align with ISO/IEC 16262
- December 1999 - ES3, added regex, try/catch and more
- July 2008 - ES4, Abandoned due to complexity
- December 2009 - ES5, strict mode, JSON, get/set, and more
- June 2011 - ES5.1, to align with ISO/IEC 16262 3rd edition
- June 2015 - ES6/ES2015, a lot of stuff!
- 28 January 2016 - final feature set for ES2016
- 29 January 2017 - final feature set for ES2017
- 23-25 January 2018 - final feature set for ES2018
- End of January 2019 - final feature set for ES2019

ECMAScript 2015 is huge

Arrow functions
Binary and Octal Literals
Block-scoped variables (let / const)
Classes
Default + rest + spread parameters
Destructuring
Enhanced object literals
Generators
Iterators + for..of
Map + Set + WeakMap + WeakSet

•new Math + Number + String + Array + Object APIs
•Modules
•Promises
•Proxies
•Subclassable Built-ins
•Symbols
•Template strings
•Unicode (full support)

All changes can be found here

ECMAScript 2016 even bigger

Array.prototype.includes
Exponentiation Operator

ECMAScript 2017

Object.values/Object.entries
String padding
Object.getOwnPropertyDescriptors()
Trailing commas in function parameter lists and calls
Async Functions

ECMAScript 2018

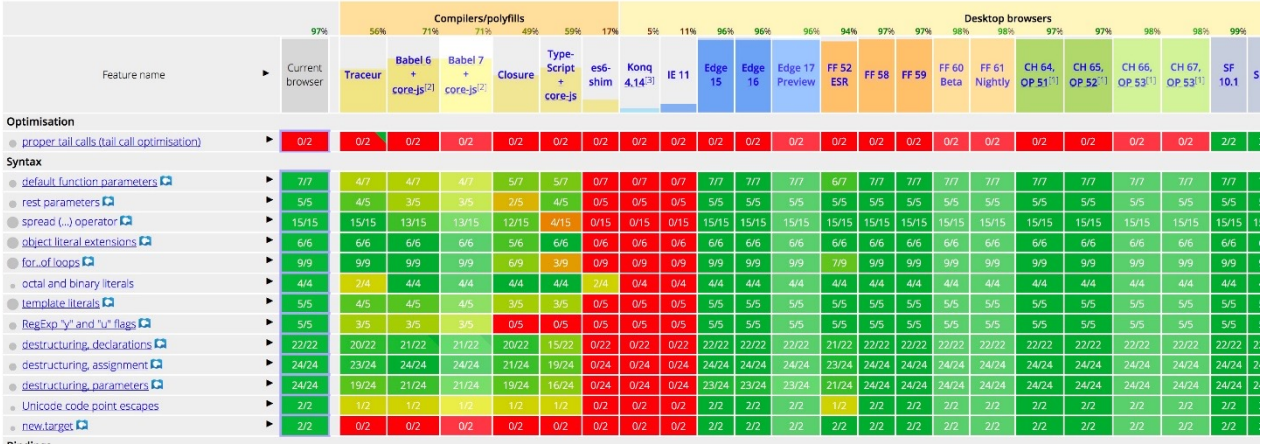
Shared memory and atomics
Lifting template literal restriction
s (dotAll) flag for regular expressions
RegExp named capture groups
Rest/Spread Properties
RegExp Lookbehind Assertions
RegExp Unicode Property Escapes
Promise.prototype.finally()
Asynchronous Iteration
Optional catch binding
JSON superset

ECMAScript 2019

Symbol.prototype.description
Function.prototype.toString revision
Object.fromEntries
Well-formed JSON.stringify
String.prototype.{trimStart,trimEnd}
Array.prototype.{flat,flatMap}

ECMAScript compatibility tables

http://kangax.github.io/compat-table/es6/
or
https://caniuse.com/#feat=es6



		Compilers/polyfills						Desktop browsers																
	97%	56%	71%	71%	49%	59%	17%	5%	11%	96%	96%	96%	94%	97%	97%	98%	98%	97%	97%	98%	98%	99%	
Feature name	Current browser	Traceur	Babel 6 + core-js[2]	Babel 7 + core-js[2]	Closure	TypeScript + core-js	es6-shim	Konq 4.14[3]	IE 11	Edge 15	Edge 16	Edge 17 Preview	FF 52 ESR	FF 58	FF 59	FF 60 Beta	FF 61 Nightly	CH 64, OP 51[1]	CH 65, OP 52[1]	CH 66, OP 53[1]	CH 67, OP 53[1]	SF 10.1	S
Optimisation																								
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	
Syntax																								
default function parameters	7/7	4/7	4/7	4/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7	7/7	6/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7		
rest parameters	5/5	4/5	3/5	3/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5		
spread (...) operator	15/15	15/15	13/15	13/15	12/15	4/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	1	
object literal extensions	6/6	6/6	6/6	6/6	5/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6		
for..of loops	9/9	9/9	9/9	9/9	6/9	3/9	0/9	0/9	0/9	9/9	9/9	9/9	7/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9		
octal and binary literals	4/4	2/4	4/4	4/4	4/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4		
template literals	5/5	4/5	4/5	4/5	3/5	3/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5		
RegExp "y" and "u" flags	5/5	3/5	3/5	3/5	0/5	0/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5		
destructuring, declarations	22/22	20/22	21/22	21/22	20/22	15/22	0/22	0/22	0/22	22/22	22/22	22/22	21/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	2	
destructuring, assignment	24/24	23/24	24/24	24/24	21/24	19/24	0/24	0/24	0/24	24/24	24/24	24/24	23/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	2	
destructuring, parameters	24/24	19/24	21/24	21/24	19/24	16/24	0/24	0/24	0/24	23/24	23/24	23/24	21/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	2	
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	2/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2		
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2		

ES6 FEATURES

Let's look more closely - Variable Declarations

```
console.log(variable); // ??

var variable = 10;

console.log(variable); // ??
```

```
console.log(variable); // ??

let variable = 10;

console.log(variable); // ??
```

```
const PI = 3.141593
PI > 3.0
```

Constant

Support for constants (also known as "immutable variables"), i.e., variables which cannot be re-assigned new content. Notice: this only makes the variable itself immutable, not its assigned content (for instance, in case the content is an object, this means the object itself can still be altered).

```
{
 const variable = 5; // same scoping rules as let
 variable = variable*2; // ??
}
```

```
const variable = [5];
variable[0] = variable[0]*2; // ??
```

Variable Declarations

```
const a = {};

a.id = 0;

console.log(a);
```

```
for (var i = 0; i < 5; i++) {}

console.log(i);
```

```
for (let j = 0; j < 5; j++) {}

console.log(j);
```

Template Literals. String Interpolation

```
const world = 'World';
const str = `Hello ${world}!`;
```

```
const multiline = `
 Sir, in my heart there was a kind of fighting
 That would not let me sleep. Methought I lay
`;
```

```
const multiline = `
 Here can be any function ${(function(){ console.log('Hello, world!') })()}
`;
```

Intuitive expression interpolation for single-line and multi-line strings. (Notice: don't be confused, Template Literals were originally named "Template Strings" in the drafts of the ECMAScript 6 language specification)

```
var customer = { name: "Foo" }
var card = { amount: 7, product: "Bar", unitprice: 42 }
var message = `Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?`
```

Destructuring

Convenient syntax to extract values from objects and arrays.

```
const arr = [1, 2, 3, 4];
const [a, , b, c] = arr;

console.log(a, b, c); // 1, 3, 4
```

```
let obj = {e: 1, f: 2, g: 3};
let {e, f, h} = obj;

console.log(e, f, h); // 1, 2, undefined
```

Spread

Convenient way to get "everything else that remains".

```
const [head, ...tail] = [1,2,3,4,5];
console.log(head, tail); // 1, [2, 3, 4, 5]
```

```
let {a, ...rest} = {a: 1, b: 2, c: 3};
console.log(a, rest); // SyntaxError: Unexpected token ... // not es2015 😢,
// but added in ES2018 🎉 so 1, { b: 2, c: 3 }
```

```
const {...rest, foo} = obj; // SyntaxError
const {foo, ...rest1, ...rest2} = obj; // SyntaxError
```

Arrow Functions

An arrow function is always bound to the scope it's defined in. Arrow functions are always anonymous.

```
let arr = [1,2,3,4].reduce((mem, n) => { return mem + n });
console.log(arr); // 10
```

Shorthand

```
let localIncrementer1 = (n) => { return n + 1 };
let localIncrementer2 = (n) => n + 1;
let localIncrementer3 = n => n + 1;
```

More expressive closure syntax.

```
odds  = evens.map(v => v + 1)
pairs = evens.map(v => ({ even: v, odd: v + 1 }))
nums  = evens.map((v, i) => v + i)
```

Function Parameters

Easier way to achieve named params, default params, and optional params.

```
function sum (a = 0, b = 0) {
 return a + b;
}

console.log(sum(2)); // 2
```

```
function sum (...vals) {
 // better than arguments!
 return vals.reduce((mem, n) => mem + n, 0);
}
console.log(sum(1, 2, 3, 4)); // 10
```

```
function rectArea ({ width = 5, height = 5 } = {}) {
 return width * height;
}
console.log(rectArea({height: 30})); // 150
```

Extended Parameter Handling. Default Parameter Values

Simple and intuitive default values for function parameters.

```
function f (x, y = 7, z = 42) {
    return x + y + z
}
f(1) === 50
```

Extended Parameter Handling. Rest Parameter

Aggregation of remaining arguments into single parameter of variadic functions.

```
function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, "hello", true, 7) === 9
```

One caveat with default function params

```
function test(a, b) {}
test.length // 2
function test(a, b = 1) {}
test.length // 1
function test(a = 1, b ) {}
test.length // 0
```

Classes. Class Definition

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

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

Classes. Class Definition

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

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

```
class Animal {
constructor (name, legs) {
this.name = name;
this.legs = legs;
}
describe () {
return `${this.name} has ${this.legs} legs`;
}
}
class Cat extends Animal {
constructor (name) {
super(name, 4);
}
describe () {
return `${super.describe()} and says meow`;
}
}
let cat = new Cat('Tom');
console.log(cat.describe()); // Tom has 4 legs and says
meow
```

Getters/Setters

```
const person = {
age: 5,
get myAge() {
    return this.age;
},
set name(name) {
if (!name) throw new Error('I need a name');
   this._name = name
 }
}
person.myAge;
person.name = 'Tom';
```

Modules

```
// imports
import React from 'react';
import { NavigationMixin as Navigation } from
'navigation';
// exports
export default MiscUtilities;
export { buildQuery as toQuery };
```

Generators

Generators, a new feature of ECMAScript 2015, are functions that can be paused and resumed. This helps with many applications: iterators, asynchronous programming, etc.

```
function* genFunc() {
yield 'a';
yield 'b';
}
function* objectEntries(obj) {
const propKeys = Reflect.ownKeys(obj);
for (const propKey of propKeys) {
yield [propKey, obj[propKey]];
}
}
```

More info here http://2ality.com/2015/03/es6-generators.html

Generators. Generator Function, Iterator Protocol

Support for generators, a special case of Iterators containing a generator function, where the control flow can be paused and resumed, in order to produce sequence of values (either finite or infinite).

```
let fibonacci = {
    *[Symbol.iterator]() {
        let pre = 0, cur = 1
        for (;;) {
            [ pre, cur ] = [ cur, pre + cur ]
            yield cur
        }
    }
}

for (let n of fibonacci) {
    if (n > 1000)
        break
    console.log(n)
}
```

Promises. Promise Usage

First class representation of a value that may be made asynchronously and be available in the future.

```
function msgAfterTimeout (msg, who, timeout) {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve(`${msg} Hello ${who}!`), timeout)
    })
}
msgAfterTimeout("", "Foo", 100).then((msg) =>
    msgAfterTimeout(msg, "Bar", 200)
).then((msg) => {
    console.log(`done after 300ms:${msg}`)
})
```

Proxying

Hooking into runtime-level object meta-operations.

```
let target = {
    foo: "Welcome, foo"
}
let proxy = new Proxy(target, {
    get (receiver, name) {
        return name in receiver ? receiver[name] : `Hello, ${name}`
    }
})
proxy.foo   === "Welcome, foo"
proxy.world === "Hello, world"
```

Async functions

An async function can contain an await expression that pauses the execution of the async function and waits for the passed Promise's resolution, and then resumes the async function's execution and returns the resolved value.

```
function fetchJson(url) {
return fetch(url)
.then(request => request.text())
.then(text => JSON.parse(text))
.catch(error => console.log(`ERROR: ${error.stack}`));
}
fetchJson('https://api.github.com/users/${user}/repos').then(res => console.log(res))
async function fetchJson(url) {
try {
const request = await fetch(url);
const text = await request.text();
return JSON.parse(text);
}
catch (error) {
console.log(`ERROR: ${error.stack}`);
}
}
fetchJson('https://api.github.com/users/${user}/repos').then(res => console.log(res))
```

Async functions

More info here:
https://developers.google.com/web/fundamentals/primers/async-functions
http://2ality.com/2016/02/async-functions.html

Asynchronous Iteration

With ECMAScript 2015, JavaScript got built-in support for synchronously iterating over data. But what about data that is delivered asynchronously? For example, lines of text, read asynchronously from a file or an HTTP connection.

```
function* genFunc() {
yield 'a';
// How about some async operation here 🤔?
yield 'b';
}
function* genFunc() {
// await somethisngAsync()
yield 'a';
// await somethisngAsync()
yield 'b';
}
```

Asynchronous Iteration

```
async function* asyncRandomNumbers() {
// This is a web service that returns a random number
const url = 'https://www.random.org/decimal-
fractions/?num=1&dec=10&col=1&format=plain&rnd=new';
while (true) {
const response = await fetch(url);
const text = await response.text();
yield Number(text);
}
}
async function example() {
for await (const number of asyncRandomNumbers()) {
console.log(number);
if (number > 0.90) break;
}
}
```

Asynchronous Iteration

More info here:
https://jakearchibald.com/2017/async-iterators-and-generators/
http://2ality.com/2016/10/asynchronous-iteration.html

New data types

- Symbol

```
const SOME_CONSTANT_1 = Symbol();
const SOME_CONSTANT_2 = Symbol('hello');
```

- Map

```
let myMap = new Map();
let keyString = 'a string';
let keyObj = {};
let keyFunc = function() {};
myMap.set(keyString, "value associated with 'a string'");
myMap.set(keyObj, 'value associated with keyObj');
myMap.set(keyFunc, 'value associated with keyFunc');
myMap.get(keyString); // "value associated with 'a string'"
myMap.get(keyObj); // "value associated with keyObj"
myMap.get(keyFunc); // "value associated with keyFunc"
```

New data types

- Set

```
let mySet = new Set();
mySet.add(1); // Set { 1 }
mySet.add(5); // Set { 1, 5 }
mySet.add(5); // Set { 1, 5 }
mySet.has(1); // true
mySet.has(3); // false
mySet.has(5); // true
mySet.has(Math.sqrt(25)); // true
```

- WeakMap
- WeakSet

Other changes

(ES6) find, findIndex
(ES6) Math.sign
(ES6) startsWith, endsWith, includes
(ES 2016) exponention operator
(ES 2017) Object values, entries
(ES 2017) trailing commas
(ES 2017) async/await
And other stuff. Check materials in the end to find out more!

Read at home

Articles:
http://2ality.com/2015/08/getting-started-es6.html
http://2ality.com/2016/01/ecmascript-2016.html
http://2ality.com/2016/02/ecmascript-2017.html
http://2ality.com/2017/02/ecmascript-2018.html
Books:
http://exploringjs.com/es6/
http://exploringjs.com/es2016-es2017.html
http://exploringjs.com/es2018-es2019/index.html
Subscribe to http://esnextnews.com/ newsletter (it's FREE 🎉)

Questions

What are Promises?
Promise constructor
What is async/await and How does it work?
What's the difference between Spread operator and Rest operator?
What are Default Parameters?
What are Arrow functions?
What are the new features in ES6 or ECMAScript 2015?

Homework

1. Rewrite setTimeout as Promise(delay (1000));
2. Upload files from array one by one(not in parallel way);

FE Online UA Training Course Feedback

I hope that you will find this material useful.

If you find errors or inaccuracies in this material or know how to improve it, please report on to the electronic address:

serhii_shcherbak@epam.com

With the note [FE Online UA Training Course Feedback]

Thank you.

Q&A

UA Frontend Online LAB