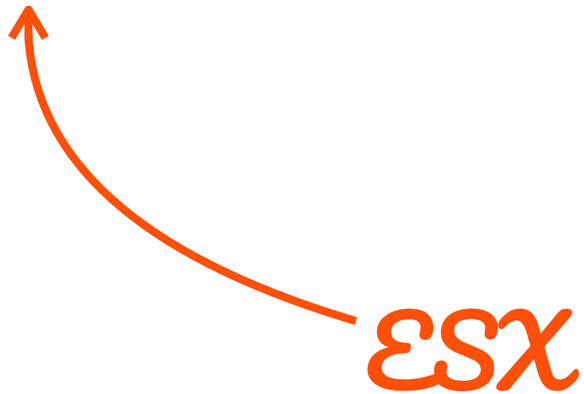
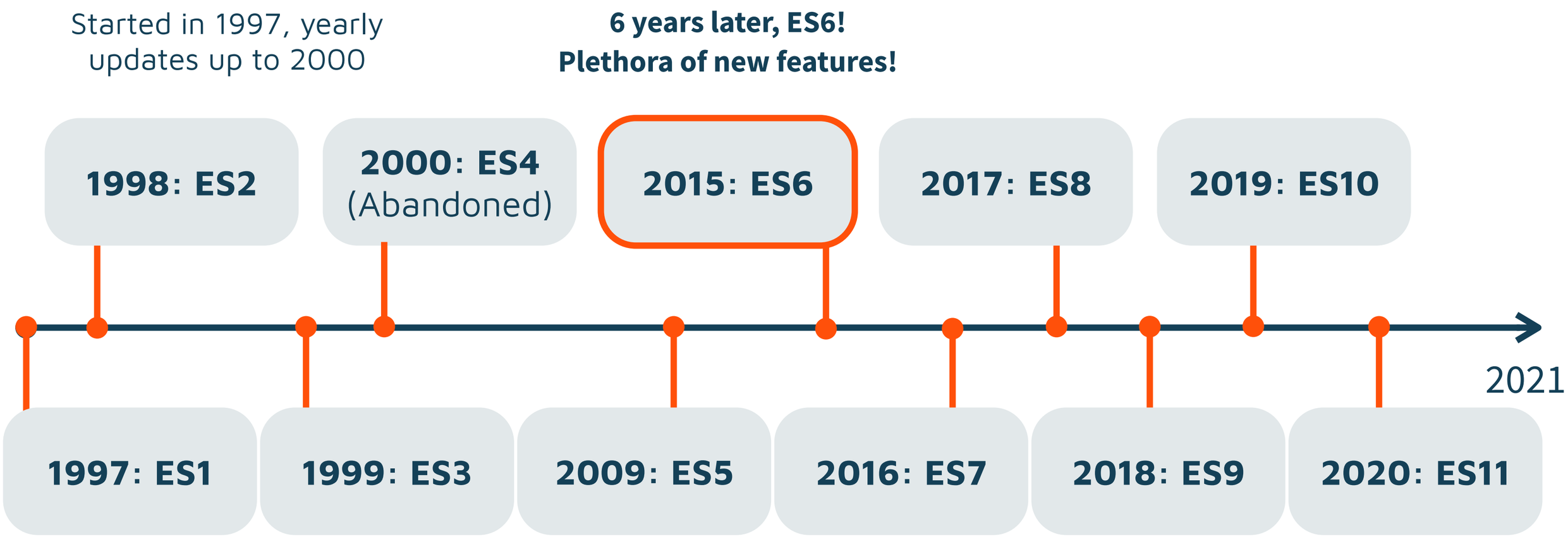
ADVANCED JS



Timeline


Started in 1997, yearly updates up to 2000
6 years later, ES6!
Plethora of new features!
1998: ES2
2000: ES4 (Abandoned)
2015: ES6
2017: ES8
2019: ES10
1997: ES1
1999: ES3
2009: ES5
2016: ES7
2018: ES9
2020: ES11
2021
10 years later, ES5.
Not much new stuff.
Yearly updates since ES6.

ES6

WHAT IS ES6?

ES (ECMAScript) is a **scripting language standard**.

JavaScript **implements** ECMAScript.

ES6 means the **6th Edition of ECMAScript.**

ES6 FEATURES

Let & Const

Arrow Functions

Default Parameters

Template Literals

Destructuring

Rest & Spread

New Array Methods

Classes

Modules

and more!

LET

before

```
var x = 1;

if (x === 1) {
  var x = 2;
  console.log(x); // 2
}

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

after

```
let x = 1;

if (x === 1) {
  let x = 2;
  console.log(x); // 2
}

console.log(x); // 1
```

CONST

before

```
var num = 42;

try {
  num = 99;
} catch (err) {
  console.log(x); // no err
}

console.log(num); // 99
```

after

```
const num = 42;

try {
  num = 99;
} catch (err) {
  console.log(x); // TypeError
}

console.log(num); // 42
```

ARROW FUNCTIONS

before

```
const arr = [1, 2, 3];
const squares = arr.map(function (x) { return x * x });
```

after

```
const arr = [1, 2, 3];
const squares = arr.map(x => x * x );
```

ARROW FUNCTIONS

before

```
function MyComponent() {
  const button = document.getElementById('myButton');
  const obj = this;
  button.addEventListener('click', function () {
    obj.handleClick();
  });
}
```

after

```
function MyComponent() {
  const button = document.getElementById('myButton');
  button.addEventListener('click', () => {
    this.handleClick();
  });
}
```

DEFAULT PARAMETERS

before

```
function magic(x, y) {
  x = x || 0;
  y = y || 0;
  ...
}
```

after

```
function magic(x = 0, y = 0) {
  ...
}
```

TEMPLATE LITERALS

before

```
function greet(first, last) {
  console.log('Hello ' + first + ' ' + last + '!');
}
```

after

```
function greet(first, last) {
  console.log(`Hello ${first} ${last}!`);
}
```

TEMPLATE LITERALS

before

```
var HTML5_SKELETON =
  '<!doctype html>\n' +
  '<html>\n' +
  '  <head>\n' +
  '    <title></title>\n' +
  '    <meta charset="UTF-8">\n' +
  '  </head>\n' +
  '  <body>\n' +
  '  </body>\n' +
  '</html>\n'
```

after

```
const HTML5_SKELETON = `
<!doctype html>
<html>
  <head>
    <title></title>
    <meta charset="UTF-8">
  </head>
  <body>
  </body>
</html>`
```

DESTRUCTURING

```
// Assignment
const numbers = [1, 2, 3];
const [one, two, three] = numbers;

// Swapping
let a = 1;
let b = 2;
[a, b] = [b, a]

// Ignoring
const foo = () => [1, 2, 3];
let [p, , q] = foo();
```

```
// Works with objects too!
const obj = { bar: 42, baz: true };
const { bar, baz } = obj;

// Can also rename variables
const { bar: qux, baz: quux } = obj;
```

REST

```
function secret(first, ...others) {
  console.log(first);
  console.log(others);
}
secret(1, 2, 3, 4, 5);
```

accessing the "rest" of the arguments

SPREAD

before

```
var arr1 = ['a', 'b'];
var arr2 = ['c'];
var arr3 = ['d', 'e'];

console.log(arr1.concat(arr2, arr3));
// ['a', 'b', 'c', 'd', 'e']
```

after

```
const arr1 = ['a', 'b'];
const arr2 = ['c'];
const arr3 = ['d', 'e'];

console.log([...arr1, ...arr2, ...arr3]);
// ['a', 'b', 'c', 'd', 'e']
```

SPREAD

```
const arr = [1, 2, 3];
const copyOfArr = [...arr];
```

copying an array

NEW ARRAY METHODS

```
const arr = [1, 2, 3];
arr.forEach(n => console.log(n));      // prints 1, 2, 3
arr.map(n => n * n);                   // returns [1, 4, 9]
arr.find(n => n > 1);                  // returns 2
arr.filter(n => n > 1);                // returns [2, 3]
arr.some(n => n > 1);                  // returns true
arr.every(n => n > 1);                 // returns false
arr.reduce((n, acc) => acc + n, 0);    // returns 6
```

and more...

CLASSES

before

```
function Person(name) {
  this.name = name;
}

Person.prototype.describe = function () {
  return 'Person called' + this.name;
}
```

after

```
class Person {
  constructor(name) {
    this.name = name;
  }
  describe() {
    return `Person called ${this.name}`;
  }
}
```

CLASSES

before

```
function Employee(name, title) {
  Person.call(this, name);
  this.title = title;
}

Employee.prototype = Object.create(Person.prototype);
Employee.prototype.constructor = Employee;
Employee.prototype.describe = function () {
  return Person.prototype.describe.call(this)
    + ' (' + this.title + ') ';
}
```

CLASSES

after

```
class Employee extends Person {
  constructor(name, title) {
    super(name);
    this.title = title;
  }

  describe() {
    return `${super.describe()} (${this.title})`;
  }
}
```

MODULES

before

```
// ----- lib.js -------
var sqrt = Math.sqrt;
function square(x) {
  return x * x;
}
function diag(x, y) {
  return sqrt(square(x) + square(y));
}

module.exports = {
  sqrt: sqrt,
  square: square,
  diag: diag,
}
```

```
// ------ main.js ------
var square = require('./lib').square;
var diag = require('./lib').diag;

console.log(square(11)) // 121
console.log(diag(3, 4)) // 5
```

MODULES

after

```
// ----- lib.js -------
export const sqrt = Math.sqrt;
export const square = x => x * x;
export const diag = (x, y) => sqrt(square(x) +
square(y));

// ------ main.js ------
import { square, diag } from './lib';
console.log(square(11)) // 121
console.log(diag(3, 4)) // 5
```

MODULES

before

```
// ------ myFunc.js ------
module.exports = function () { ... }

// ------- main.js -------
var myFunc = require('./myFunc')
myFunc();
```

after

```
// ------ myFunc.js ------
export default function () { ... }

// ------- main.js -------
import myFunc from './myFunc';
myFunc();
```

RESOURCES

http://exploringjs.com/es6/

https://github.com/lukehoban/es6features

http://es6-features.org

NEXT CLASS: DATA BINDING

https://uiuc-web-programming.gitlab.io/fa21/