

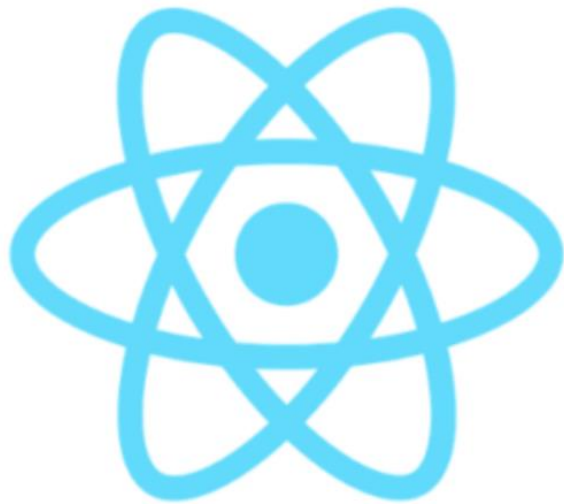


# Full-stack Application Development

Novelties in ECMAScript 6+

# Where to Find The Code and Materials?

<https://github.com/iproduct/react-typescript-academy-2022>



# EcmaScript 6 – ES 2015, Harmony

[\[https://github.com/lukehoban/es6features\]](https://github.com/lukehoban/es6features)

- arrows
- classes
- enhanced object literals
- template strings
- destructuring
- default + rest + spread
- let + const
- iterators + for..of
- Generators
- unicode
- Modules + module loaders
- map + set + weakmap + weakset
- proxies
- symbols
- subclassable built-ins
- Promises
- math + number + string + array + object APIs
- binary and octal literals
- reflect api
- tail calls

# ES6 Classes [<http://es6-features.org/>]

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

```
class Rectangle extends Shape {  
  constructor (id, x, y, width, height) {  
    super(id, x, y)  
    this.width = width  
    this.height = height  
  }  
}  
  
class Circle extends Shape {  
  constructor (id, x, y, radius) {  
    super(id, x, y)  
    this.radius = radius  
  }  
}
```

# Enhanced Object Literals

```
var obj = {  
    // __proto__  
    __proto__: theProtoObj,  
    // Shorthand for 'handler: handler'  
    handler,  
    // Methods  
    toString() {  
        // Super calls  
        return "d " + super.toString();  
    },  
    // Computed (dynamic) property names  
    [ 'prop_' + (() => 42)() ]: 42  
};
```

# Template Strings

// Basic literal string creation

```
`In JavaScript '\n' is a line-feed.`
```

// Multiline strings

```
`In JavaScript this is  
not legal.`
```

// String interpolation

```
var name = "Bob", time = "today";
```

```
`Hello ${name}, how are you ${time}?`
```

# Tagged Templates

```
function myTag(strings, personExp, ageExp) {  
  let str0 = strings[0]; // "That "  
  let str1 = strings[1]; // " is a "  
  let str2 = strings[2]; // "."  
  let ageStr;  
  if (ageExp > 99){  
    ageStr = 'centenarian';  
  } else {  
    ageStr = 'youngster';  
  }  
  return `${str0}${personExp}${str1}${ageStr}${str2}`;  
}
```

```
let person = 'Mike';  
let age = 28;  
  
let output =  
  myTag`That ${ person } is a ${ age }.`;  
  
console.log(output);  
// That Mike is a youngster.
```

# Symbols, Iterators, for-of

```
let fibonacci = {  
  [Symbol.iterator]() {  
    let pre = 0, cur = 1;  
    return {  
      next() {  
        [pre, cur] = [cur, pre + cur];  
        return { value: cur }  
      }  
    }  
  }  
}
```

```
for (var n of fibonacci) {  
  // truncate the sequence at 1000  
  if (n > 1000)  
    break;  
  console.log(n);  
}
```



# Iterators - II

```
let fibonacci = {  
  [Symbol.iterator]() {  
    let pre = 0, cur = 1, index = 0;  
    return {  
      next() {  
        [pre, cur] = [cur, pre + cur];  
        index++;  
        return { done: cur > 1000, value: cur };  
      }  
    }  
  }  
}
```

```
for (var n of fibonacci) {  
  console.log(n);  
}
```

# Generators

```
var fibonacci = {  
  [Symbol.iterator]: function*() {  
    var pre = 0,  
        cur = 1;  
    for (let i = 0; i < 15; i++) {  
      var temp = pre;  
      pre = cur;  
      cur += temp;  
      if (cur < 1000) yield cur;  
    }  
  }  
};
```

```
for (var n of fibonacci) {  
  // truncate the sequence at 1000  
  console.log(n);  
}
```

# Block Scope Vars: let [<http://es6-features.org/>]

```
for (let i = 0; i < a.length;  
    i++) {  
    let x = a[i]  
    ...  
}
```

```
for (let i = 0; i < b.length;  
    i++) {  
    let y = b[i]  
    ...  
}
```

```
const callbacks = []  
for (let i = 0; i <= 2; i++) {  
    callbacks[i] =  
        function () { return i * 2 }  
}
```

```
callbacks[0]() === 0  
callbacks[1]() === 2  
callbacks[2]() === 4
```

# ES6 Arrow Functions and this

- ECMAScript 6:

```
this.nums.forEach((v) => {  
  if (v % 5 === 0)  
    this.fives.push(v)  
})
```

- ECMAScript 5:

```
var self = this;  
this.nums.forEach(function (v) {  
  if (v % 5 === 0)  
    self.fives.push(v);  
});
```

# Array and Object Destructuring

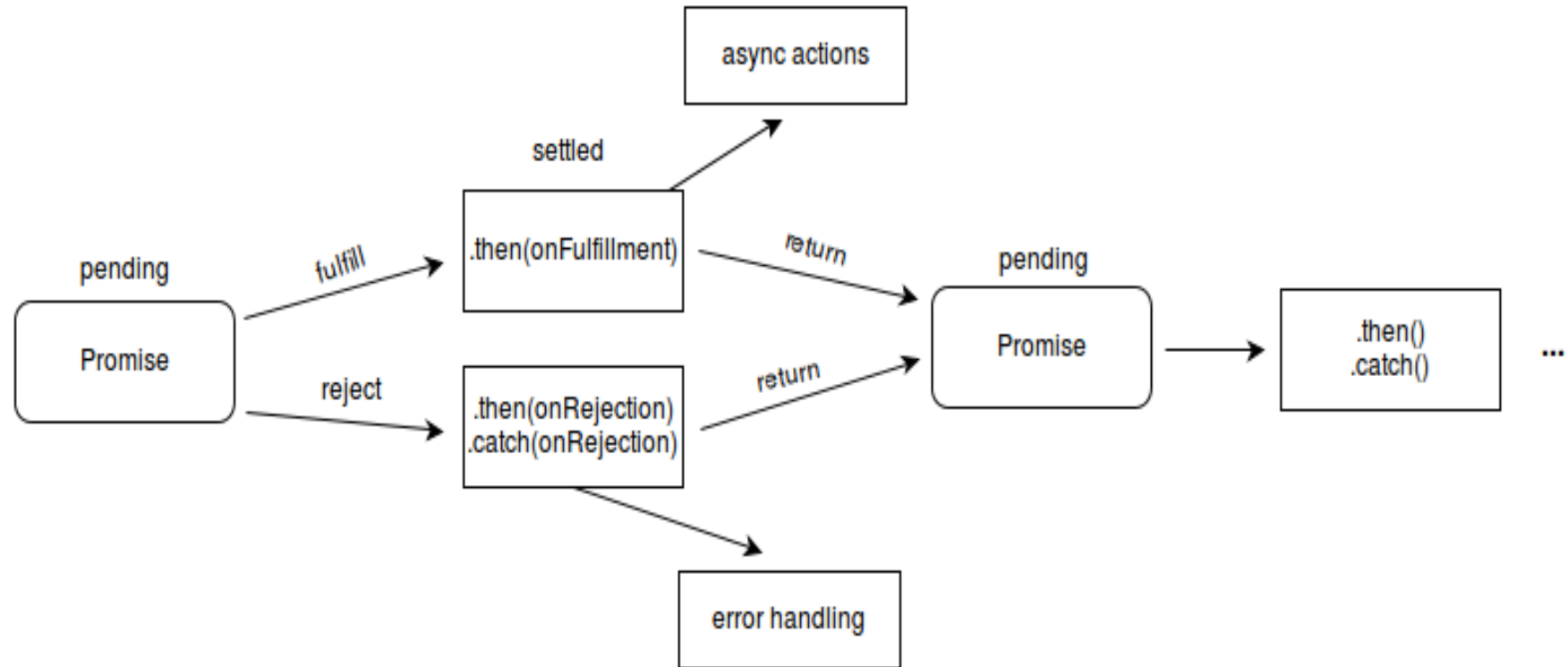
```
let persons = [  
  { name: 'Michael Harrison',  
    parents: {  
      mother: 'Melinda Harrison',  
      father: 'Simon Harrison',  
    }, age: 35},  
  { name: 'Robert Moore',  
    parents: {  
      mother: 'Sheila Moore',  
      father: 'John Moore',  
    }, age: 25}];  
  
for (let {name: n, parents: { father: f }, age } of persons) {  
  console.log(`Name: ${n}, Father: ${f}, age: ${age}`);  
}
```

# ES6 Promises [<http://es6-features.org/>]

```
function msgAfterTimeout (msg, who, timeout) {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => resolve(`${msg} Hello ${who}!`), timeout)  
  })  
}
```

```
msgAfterTimeout("", "Foo", 1000).then((msg) => {  
  console.log(`done after 1000ms:${msg}`);  
  return msgAfterTimeout(msg, "Bar", 2000);  
}).then((msg) => {  
  console.log(`done after 3000ms:${msg}`)  
})
```

# ES6 Promises



# Combining ES6 Promises

```
function fetchAsync (url, timeout, onData, onError) { ... }  
fetchPromised = (url, timeout) => {  
  return new Promise((resolve, reject) => {  
    fetchAsync(url, timeout, resolve, reject)  
  })  
}
```

```
Promise.all([  
  fetchPromised("http://backend/foo.txt", 500),  
  fetchPromised("http://backend/bar.txt", 500)  
]).then( (data) => {  
  let [ foo, bar ] = data  
  console.log(`success: foo=${foo} bar=${bar}`)  
}).catch( (err) => {  
  console.log(`error: ${err}`)  
})
```



# Combining ES6 Promises

```
function fetchAsync (url, timeout, onData, onError) { ... }  
fetchPromised = (url, timeout) => {  
  return new Promise((resolve, reject) => {  
    fetchAsync(url, timeout, resolve, reject)  
  })  
}
```

```
Promise.all([  
  fetchPromised("http://backend/foo.txt", 500),  
  fetchPromised("http://backend/bar.txt", 500)  
]).then( (data) => {  
  let [ foo, bar ] = data  
  console.log(`success: foo=${foo} bar=${bar}`)  
}, (err) => {  
  console.log(`error: ${err}`)  
})
```

# Async – Await – Try – Catch

```
async function init() {  
  try {  
    const userResult = await fetch("user.json");  
    const user = await userResult.json();  
    const gitResp = await fetch(`http://api.github.com/users/${user.name}`);  
    const githubUser = await gitResp.json();  
    const img = document.createElement("img");  
    img.src = githubUser.avatar_url;  
    document.body.appendChild(img);  
    await new Promise((resolve, reject) => setTimeout(resolve, 6000));  
    img.remove();  
    console.log("Demo finished.");  
  } catch (err) { console.log(err); }  
}
```

# JavaScript Module Systems - CommonJS

- math.js:

```
exports.add = function() {  
    var sum = 0, i = 0, args = arguments, len = args.length;  
    while (i < len) {  
        sum += args[i++];  
    }  
    return sum;  
};
```

- increment.js:

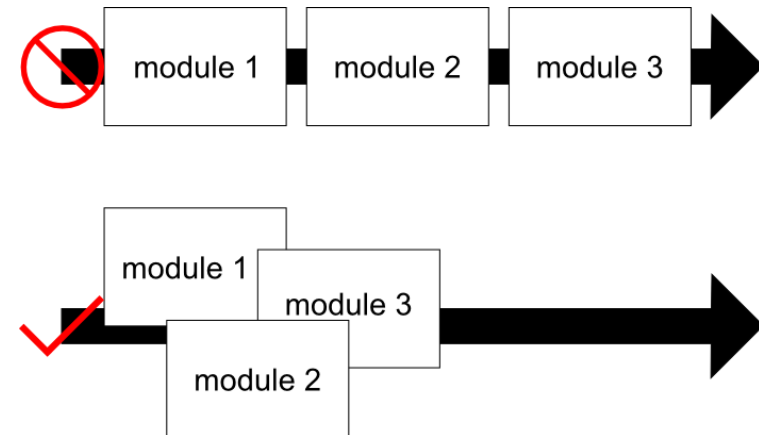
```
var add = require('./math').add;  
exports.increment = function(val) {  
    return add(val, 1);  
};
```

# JavaScript Module Systems – AMD I

```
//Calling define with module ID, dependency array, and factory  
//function
```

```
define('myModule', ['dep1', 'dep2'], function (dep1, dep2) {  
    //Define the module value by returning a value.  
    return function () {};  
});
```

```
define(["alpha"], function (alpha) {  
    return {  
        verb: function(){  
            return alpha.verb() + 2;  
        }  
    };  
});
```



# JavaScript Module Systems - AMD II

- Asynchronous module definition (AMD) – API for defining code modules and their dependencies, loading them asynchronously, on demand (lazy), dependencies managed, client-side

```
define("alpha", ["require", "exports", "beta"],
    function(require, exports, beta) {
        exports.verb = function() {
            return beta.verb();
            //OR
            return require("beta").verb();
        }
    });

define(function (require) {
    require(['a', 'b'], function (a, b) { //use modules a and b
    });
});
```

# JavaScript Module Systems – ES6

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

// someApp.js
import * as math from "./lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

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

// default export from hello.js and import
export default () => ( <div>Hello from React!</div>);
import Hello from "./hello";
```

# Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>