Modern JavaScript

Shan-Hung Wu & DataLab
CS, NTHU

ES5, ES6 and ES7

- Javascript: implementation of ***ECMAScript*** (ES)
- ES5 = ECMAScript 5 (2009)
- ES6 = ECMAScript 6 = ES2015
- ES7 = ECMAScript 7 = ES2016

Outline

- Project-based Development
  - Node.js
  - Webpack
- Modern Javascript
  - Babel
  - ES6 and 7
- Architectural Design
  - OOP vs. FP
  - Component-based Design

Outline

- Project-based Development
  - Node.js
  - Webpack
- Modern Javascript
  - Babel
  - ES6 and 7
- Architectural Design
  - OOP vs. FP
  - Component-based Design

- A Javascript runtime environment based on Google Chrome's V8 engine

```
$ node app.js
$ node    // REPL
```

- Also provides `npm` managing various ***modules***

```
$ npm init
$ npm install --[save|save-dev] <pkg-name>

var _ = require('module');
```

`--save` vs. `--save-dev`?

- Given dependency tree:

```
Your proj → Pkg 1
          → Pkg 2
Pkg 1 → Pkg 3
      → Pkg 4
Pkg 2 → Pkg 3
      → Pkg 5
```

- People who clone/fork your package will download the following packages:

```
{Pkg 1, Pkg 2, Pkg 3}  // via 'npm install'
```

Exports

```
// in module.js
exports.p = 32;
exports.f = function () {...};
// or
module.exports = ...;

// in main.js
var module = require('./module.js');
module.p   // 32
```

- API is Node.js-specific (only works at server side)

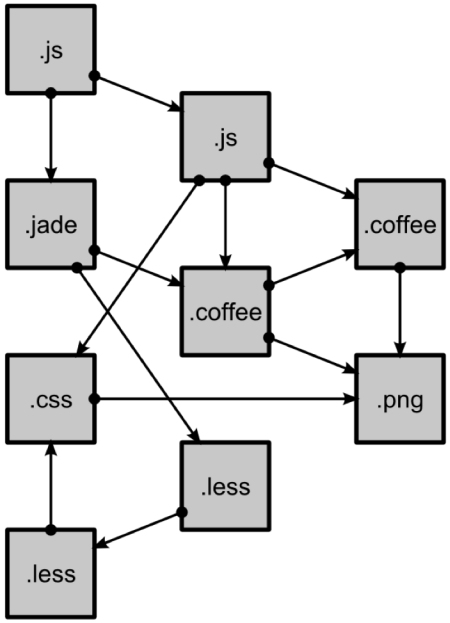
Outline

- Project-based Development
  - Node.js
  - Webpack
- Modern Javascript
  - Babel
  - ES6 and 7
- Architectural Design
  - OOP vs. FP
  - Component-based Design

Modules as `<script>`'s

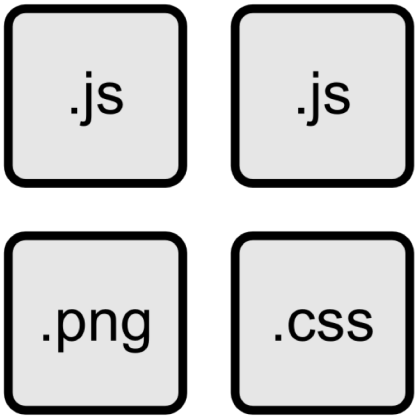
```
<script>scripts/lib/module-1.js</script>
<script>scripts/lib/module-2.js</script>
<script>scripts/main.js</script>
```

- Why not?
  - Developers have to resolve dependencies between (many) modules
  - Order of loading is important
  - May have conflicts in the global scope
  - Version update becomes a nightmare



modules
with dependencies

webpack
MODULE BUNDLER



static
assets

ES6 Imports and Exports

```
// exports.p = ...;
export var p = ...;
// module.export = function() {...};
export default function () {...}

// ... = require('./module.js');
import f, {p} from './module.js';
```

- ES6 module loaders are asynchronous while Node.js module loaders are not

Webpack

```
$ npm install --save-dev webpack
$ ./node_modules/.bin/webpack src/main.js \
  dist/main.bundle.js
```

Config File

```
// in webpack.config.js
var path = require('path');
module.exports = {
  context: path.resolve(__dirname, './src'),
  entry: './main.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].bundle.js'
  }
};

// add to the "scripts" in package.json:
"build": "webpack"  // opt. with "-w" or "-p"

$ npm run build
```

Multiple Entries

- Bundled together:

```
entry: {
  main: ['./main.js', './module.js', ...]
},
```

- Multiple bundles:

```
entry: {
  main: './main.js',
  module: './module.js'
},
```

  - May have duplicated code if referring same modules

Automatic Vendor Bundling

```
var webpack = require('webpack');
module.exports = {
  optimization: {
    splitChunks: {
      cacheGroups: {
        vendor: {
          minChunks: 2,
          name: 'vendor',
          chunks: 'all'
        }
```

- Any modules that get loaded 2 or more times it will be bundled into a separate file
  - To be used as a new `<script>` in HTML
- Speeds up loading due to browser caching

Manual Vendor Bundling

```
entry: {
  main: './main.js',
}, ...
optimization: {
  splitChunks: {
    cacheGroups: {
      vendor: {
        test: /[\\/]node_modules[\\/]lodash[\\/]/,
        name: 'vendor’,
        chunks: 'all’,
        enforce: true
      }
    }
  }
},
```

Packing CSS Files

```
$ npm install --save-dev css-loader style-loader

// in module.js
import './module.css';

// in webpack.config.js
module.exports = {
  module: {
    rules: [{
      test: /\.css$/,
      use: ['style-loader', 'css-loader'],
    }],
  }, ...
}
```

- Allows ***modularized CSS***

Loaders

- Transform non-Javascript files into modules

```
module: {
  rules: [{
    test: /\.css$/,
    /* processed in reverse array order */
    use: ['style-loader', 'css-loader'],
  }],
},
```

- `css-loader` first transforms CSS into modules
- Then, `style-loader` adds `<style>` to DOM

Outline

- Project-based Development
  - Node.js
  - Webpack
- Modern Javascript
  - Babel
  - ES6 and 7
- Architectural Design
  - OOP vs. FP
  - Component-based Design

ES6/7 and BABEL

- ES6 (2015) and 7 (2016) are not fully supported by major browsers yet
- Babel: a transpiler that transforms ES6/7 syntax into ES5
- Modular: plug-ins and presets
  - E.g., preset-es2015 (deprecated), preset-env
  - Only syntax translation by default
- Requires Ployfill for new global objects
  - E.g., Symbols, generators, Promise, etc.

Babel Loader

```
$ npm install --save-dev \
 babel-loader @babel/core @babel/preset-env

// in webpack.config.js
module.exports = {
  module: {
    rules: [{
      test: /\.js$/,
      exclude: [/node_modules/],
      use: [{
        loader: 'babel-loader',
        options: {
          presets: [['@babel/preset-env',
                     {modules: false}]]
        }
      }]
    }],
  }, ...
}
```

- Turn off module transpiling to allow tree shaking in Webpack

Polyfill

```
$ npm install --save @babel/polyfill

// in main.js (entry point)
import ‘@babel/polyfill';

// or, in webpack.config.js
entry: [‘@babel/polyfill', './main.js'],
```

Outline

- Project-based Development
  - Node.js
  - Webpack
- Modern Javascript
  - Babel
  - ES6 and 7
- Architectural Design
  - OOP vs. FP
  - Component-based Design

Block Scoped Variables

```
function f() {
  let x;
  for (let i = 0; i < 10; i++) {
    x = 'foo';
  }
  console.log(i); // error
  const y = 10;
  if (...) {
    y++; // error
  }
}
```

Arrow Functions and Lexical `this`

```
// implicit return
let nums2 = nums1.map((v, i) => v + i);

let user = {
  name: 'Bob',
  friends: ['Alice', 'John'],
  greet: function() {
    this.friends.forEach(f => {
      /* lexical this */
      console.log('Hi ' + f + ", I'm " +
        this.name);
    });
  }
}
```

- Also lexical `arguments`

Default, Rest, and Spread

```
function f(x, y = 12) {    // default param
  return x + y;
}
f(3)  // 15

function f(x, ...y) {      // rest param
  return x + y.length;     // y is an array
}
f(3, 'hello', true)  // 6

function f(x, y, z) {
  return x + y + z;
}
f(...[1, 2, 3])            // spread op
```

Destructuring

```
let {name, friends} = user;
name       // 'Bob'
friends    // ['Alice', 'John']

let {name: n, friends: f} = user;
n          // 'Bob'
f          // ['Alice', 'John']

let [a, , b = 3] = [1, 2];
a          // 1
b          // 3

function f({name = 'Alice'}) {return name;}
f(user)    // 'Bob'
```

- with pattern matching

Template String Literals

```
let name = 'Bob', score = 3.1416;
`${name} gets a score ${score.toFixed(2)}`
// 'Bob gets a score 3.14'

`This is not legal
in ES5.`
```

Enhanced Object Literals

```
let name = 'Bob';
let user = {
  /* shorthand for 'name: name' */
  name,
  /* method */
  greet() {  // method
    return `I\'m ${this.name}`;
  },
  /* computed (dynamic) property name */
  ['isOlderThan' + (() => 18)()]: true
} ;
```

Classes

```
class User {
  constructor(name) {
    this.name = name;
  }
  greet() {
    return `I\'m ${this.name}`;
  }
  static yell() {
    return 'Ahh~';
  }
}
let user = new User('Bob');
user.greet()     // "I'm Bob"
User.yell()      // 'Ahh~'
```

- Just a syntax sugar for `function User() {}`

Inheritance

```
class Vip extends User {
  constructor(name, title) {
    super(name);
    this.title = title;
  }
  pay() {...}
  greet() {  // overriding
    return `I\'m ${this.title} ${this.name}`;
  }
}
let vip = new Vip('Bob', 'Mr.');
vip.greet()     // 'Hi, I am Mr. Bob'
Vip.yell()      // 'Ahh~'
```

- Classes save repeated code for objects
- Inheritance saves repeated code for classes

`instanceof` Operator

- How to tell if an object is an instance of a class?

```
user.constructor === User // true
vip.constructor === User  // false
```

- How to tell if an object is an instance of a class ***or its subclass***?

```
user instanceof User // true
vip instanceof User  // true
```

Symbols

- Values must be unique
- Of ***new primitive type***

```
let s1 = Symbol('key'); // factory function
let s2 = Symbol('key');
console.log(s1);  // 'Sumbol(key)'
typeof s2         // 'symbol'
s1 === s2         // always false
```

- Requires Babel Polyfill

Mixins

```
class User {
  constructor(name) {
    this.name = name;
  }
  log() {
    console.log(`I\'m ${this.name}`);
  }
}
let user = new User('Bob');

function mixinLogger(obj) {
  obj[mixinLogger.symbol] = function() {
    for (let prop in obj) console.log(prop);
  };
}
mixinLogger.symbol = Symbol('log');

mixinLogger(user);
user[mixinLogger.symbol]();  // 'name'
```

- What if `obj['log']`?

Iterators & Generators

```
let arr = [3, 5, 7];
arr.foo = 'bar';
for (let i in arr) {
  console.log(i);     // '0', '1', '2', 'foo'
}
for (let i of arr) {
  console.log(i);     // 3, 5, 7
}

let user = {
  name: 'Bob',
  friends: ['Alice', 'John'],
  [Symbol.iterator]: function* () {  // generator
    for (let i = 0; i < this.friends.length; i++) {
      yield this.friends[i];
    }
  }
}
for (let f of user) {
  console.log(f);     // 'Alice', 'John'
}
```

- See predefined symbols
- Requires Polyfill

(ES7) Property Initializers

```
class User {
  /* same as this.xxx = yyy in constructor */
  nickName = 'Andy';
  sayHi = () => {
    console.log(`Hi, I'm ${this.nickName}`);
  }

  /* same as User.xxx = yyy */
  static privilege = 7;
  static canRead = function() {
    return User.privilage >= 4;
  }
}

let user = new User();
setInterval(user.sayHi, 1000);

// "Hi, I'm Andy"
```

- Still “experimental”

Babel Plugin

```
$ npm install --save-dev \
  @babel/plugin-proposal-class-properties

// in webpack.config.js
module.exports = {
  module: {
    rules: [{
      test: /\.js$/,
      exclude: [/node_modules/],
      use: [{
        loader: 'babel-loader',
        options: {
          presets: [[‘@babel/preset-env’,
                     {modules: false}]],
          plugins:
            ['@babel/plugin-proposal-class-properties']
        }
      }], ...
    }],
  }, ...
}
```

Reference

- ES6 in Depth
  - A series of ES6 articles by Mozilla

- Babel and ES6
- ECMAScript Compatibility Table

ESLint (Optional)

```
$ npm install --save-dev eslint
$ ./node_modules/.bin/eslint --init


// Atom package manager
apm install linter-eslint
```

Outline

- Project-based Development
  - Node.js
  - Webpack
- Modern Javascript
  - Babel
  - ES6 and 7
- Architectural Design
  - OOP vs. FP
  - Component-based Design

OOP vs. FP

- Two common programming paradigms:
- ***Object-Oriented Programming*** (OOP)
  - Uses objects to accomplish a task
  - Each object has its own data (properties) and operations (methods)
  - An objects can interact with another object
- ***Functional Programming*** (FP)
  - Uses stateless (static) functions to accomplish a task
  - Data are stored separately and are immutable

Raise Salary, the OOP Way

```
class Employee {
  constructor(name) {
    this.name = name;
    this.salary = 1000;
  }
  addSalary(amt) {
    this.salary += amt;
  }
}
let e1 = new Employee('Bob');
...
e1.addSalary(500);
...
let e2 = new Employee('Alice');
```

- Can apply existing operations (methods) to new data (employees) easily

Raise Salary, the FP Way

```
const employees = [
  ['Alice', 1000],
  ['Bob', 1000]
];

const happyEmployees = employees.map(e => {
  const clone = e.slice();
  clone[1] += 500;
  return clone;
});
```

- Can apply new operations to existing data easily

Which One Is Better?

- It's not about #lines of code
- OOP: tasks having ***fixed operations*** on ***evolving data***
  - Handles new data by adding objects
  - E.g., system software
- FP: tasks having ***evolving operations*** on ***fixed data***
  - Handles new operations by adding stateless functions
  - E.g., GUI event handing, data analysis, compilers
- Modern languages (e.g., Java, Python, and Javascript) support both
  - Use ***both*** in your project

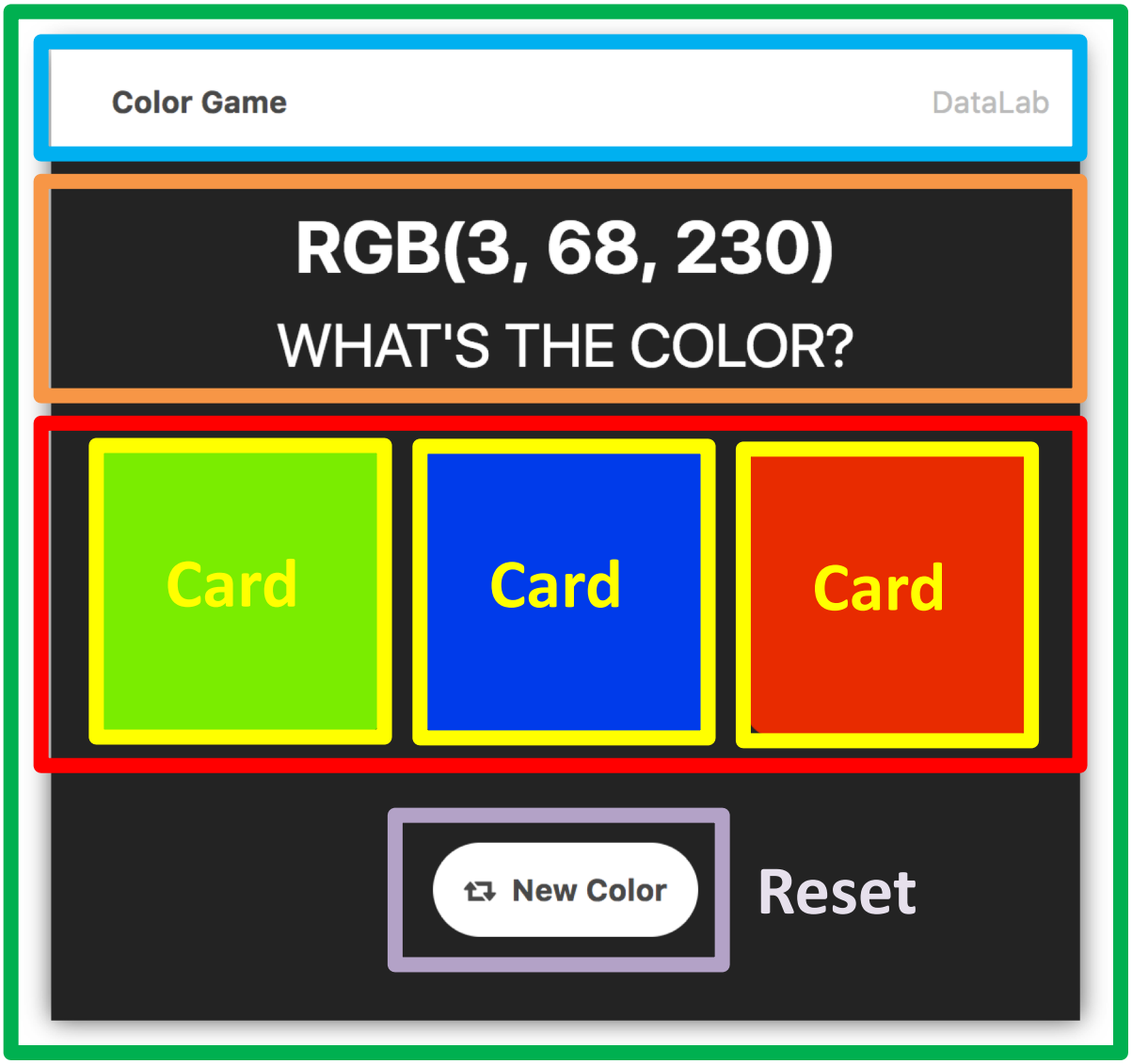
Outline

- Project-based Development
  - Node.js
  - Webpack
- Modern Javascript
  - Babel
  - ES6 and 7
- Architectural Design
  - OOP vs. FP
  - Component-based Design

GUI and Component-based Design

- When writing a GUI, it's a good practice to divide code based on visual ***components***
- OOP: one class for a component
- FP: relevant callback hooks for changes of component state

Components?


Color Game
DataLab
RGB(3, 68, 230)
WHAT'S THE COLOR?
Card
Card
Card
New Color
Reset
Navbar
Board
Main
Deck

Demo: Component-based Color Game

- In the `component-based` branch
- Run `$npm install` first
- `*.js` and `*.css` are divided by components
- Every component extends the `Component` class
  - Renders to a root DOM element
  - Interacts with nested components via method calls
  - Interacts with containers via event firing

Exercise

- Code the “Hard” and “Nightmare” modes using the components
- Be sure to configure a project ***from scratch*** by your own