COMP-308 Winter 2018

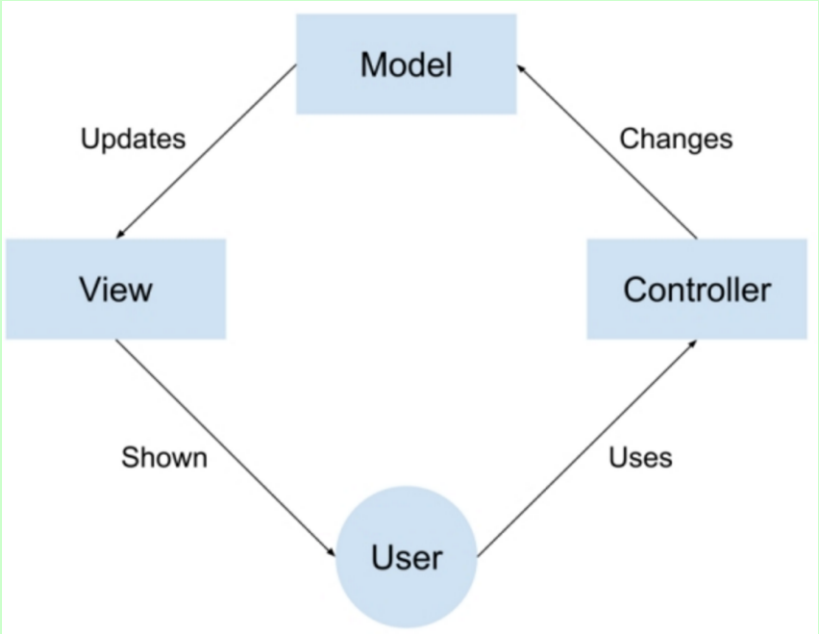
Evolution of JavaScript and new ECMASCRIPT 2015 features

Objectives:

- Explain the **advancements** in JavaScript.
- Explain **Module** pattern.
- Use ECMASCRIPT 2015 **classes**, **Arrow** functions, **new keywords for symbol declaration**, and **new features related to function parameters**.

Three-tier web application development

- **Model-View-Controller** (**MVC**) architectural pattern:
  - **Model** handles data manipulation.
  - **View** handles the visual part.
  - **Controller** responds to system and user events, commanding the Model and View to change appropriately.



The Evolution of JavaScript

- In 2008, Google released its Chrome browser, along with its fast **JIT-compiling V8 JavaScript engine**.
  - Google's **V8 engine** made JavaScript run so much faster that it completely transformed web application development.
  - One of the first products of this revolution was **Node.js**.
  - a V8-based database called **MongoDB**.
  - A frontend open source framework called **AngularJS**.
- Now JavaScript is the programming language **across all three layers** - an idea that is commonly referred to as the **full-stack JavaScript**.
  - The **MEAN stack** is just a single example of this idea

The Evolution of JavaScript

- The standard for JavaScript is ECMAScript.
- As of 2012, all modern browsers fully support ECMAScript 5.1.
- Older browsers support at least ECMAScript 3.
- On June 17, 2015, ECMA International published the sixth major version of ECMAScript, which is officially called ECMAScript 2015, and is more commonly referred to as **ECMAScript 6 or ES6**.
- Since then ECMAScript standards are on yearly release cycles.
- Here is the ES6 guide:
  - https://leanpub.com/ecmascript2015es6guide/read
- ECMAScript 2018 is still a working draft.

https://developer.mozilla.org/en-US/docs/Web/JavaScript

ECMAScript 2015

- Modules
  - **export –** to expose the module
  - **import** – to make the module available to use
  - Example: suppose you have a file named lib.js that contains the following code:

```
export function halfOf(x) {
    return x / 2;
}
```

- In your main.js file, you can use the following code:

```
import halfOf from 'lib';
console.log(halfOf(84));
```

ECMAScript 2015 - Modules

- Exporting multiple functions:
  - If lib.js file looks like this:

```
export function halfOf(x) {
    return x / 2;
}
export function multiply(x, y) {
    return x * y;
}
```

  - In your main.js file, use the following code:

```
import {halfOf, multiply} from 'lib';
console.log(halfOf(84));
console.log(multiply(21, 2));
```

ECMAScript 2015 - Modules

- Named export are used to export multiple things from a module by adding the keyword export to their declaration
- ES2015 modules also support default export values.
- Default export allows only a single default export per module
  - Example: doSomething.js that contains the following code:

```
export default function () {
    console.log('I did something')
};
```

- You'll be able to use it as follows in your main.js file:

```
import doSomething from 'doSomething';
doSomething();
```

- Is not possible to use var, let or const with export default

ECMAScript 2015 - Modules

- **Modules export bindings - live connections to values, not just values.**
  - Example: a validator.js file that looks like this:

```
export let flag = false;
export function touch() {
    flag = true;
}
```

  - You also have a main.js file that looks like this:

```
import { flag, touch } from 'validator';
console.log(flag);
touch();
console.log(flag);
```

  - The first output would be false, and the second would be true because is modified in function touch.

Let and Const

- Let and Const are new keywords **used for symbol declaration**.
- Let statement declares **a block scope local variable**.
  - Here is an example:

```
function iterateVar() {
  for(var i = 0; i < 10; i++) {
    console.log(i);
  }
  console.log(i) //will print i
}
function iterateLet() {
  for(let i = 0; i < 10; i++) {
    console.log(i);
  }
  console.log(i) //will throw an error
}
```

Let and Const

- The first function will print i after the loop, but the second one will throw an error, since i is defined by let, therefore not defined outside the block.

- The const keyword forces single assignment.
  - So, this code will throw an error as well:

```
const me = 1
me = 2 //cannot reinitialize
```

Trying Out ECMAScript 2015

- Use jsfiddle (https://jsfiddle.net/) or Babeljs:

Babel · The compil

Secure | https://babeljs.io/repl/#?babili=false&evaluate=true&lineWrap=false&presets=es2016%2Cstage-2%2Cstage-3&code=function%20iterateVar()%20%7B%0D%0A%20%20for(var%20i%20%3D

BABEL Learn ES2015 Docs Try it out Blog FAQ Forum

Evaluate Presets: es2016, stage-2, stage-3 Line Wrap Minify (Babili) Babel 6.21.1

```
function iterateVar() {
  for(var i = 0; i < 10; i++) {
    console.log(i);
  }
  console.log(i)
}
function iterateLet() {
  for(let i = 0; i < 10; i++) {
    console.log(i);
  }
  console.log(i)
}

iterateLet()
```

```
function iterateVar() {
  for (var i = 0; i < 10; i++) {
    console.log(i);
  }
  console.log(i);
}
function iterateLet() {
  for (let i = 0; i < 10; i++) {
    console.log(i);
  }
  console.log(i);
}

iterateLet();
```

```
i is not defined
```

ECMAScript 2015 - Classes

- Classes in ES2015 are basically just a *syntactic sugar* over the prototype-based inheritance.

```
class Vehicle {
    constructor(wheels) {
        this.wheels = wheels;
    }
    toString() {
        return '(' + this.wheels + ')';
    }
}
class Car extends Vehicle {
    constructor(color) {
        super(4);
        this.color = color;
    }
    toString() {
        return super.toString() + ' colored:  ' + this.color;
    }
}
```

ECMAScript 2015 - Classes

```
let car = new Car('blue');
car.toString();
console.log(car instanceof Car);
console.log(car instanceof Vehicle);
```

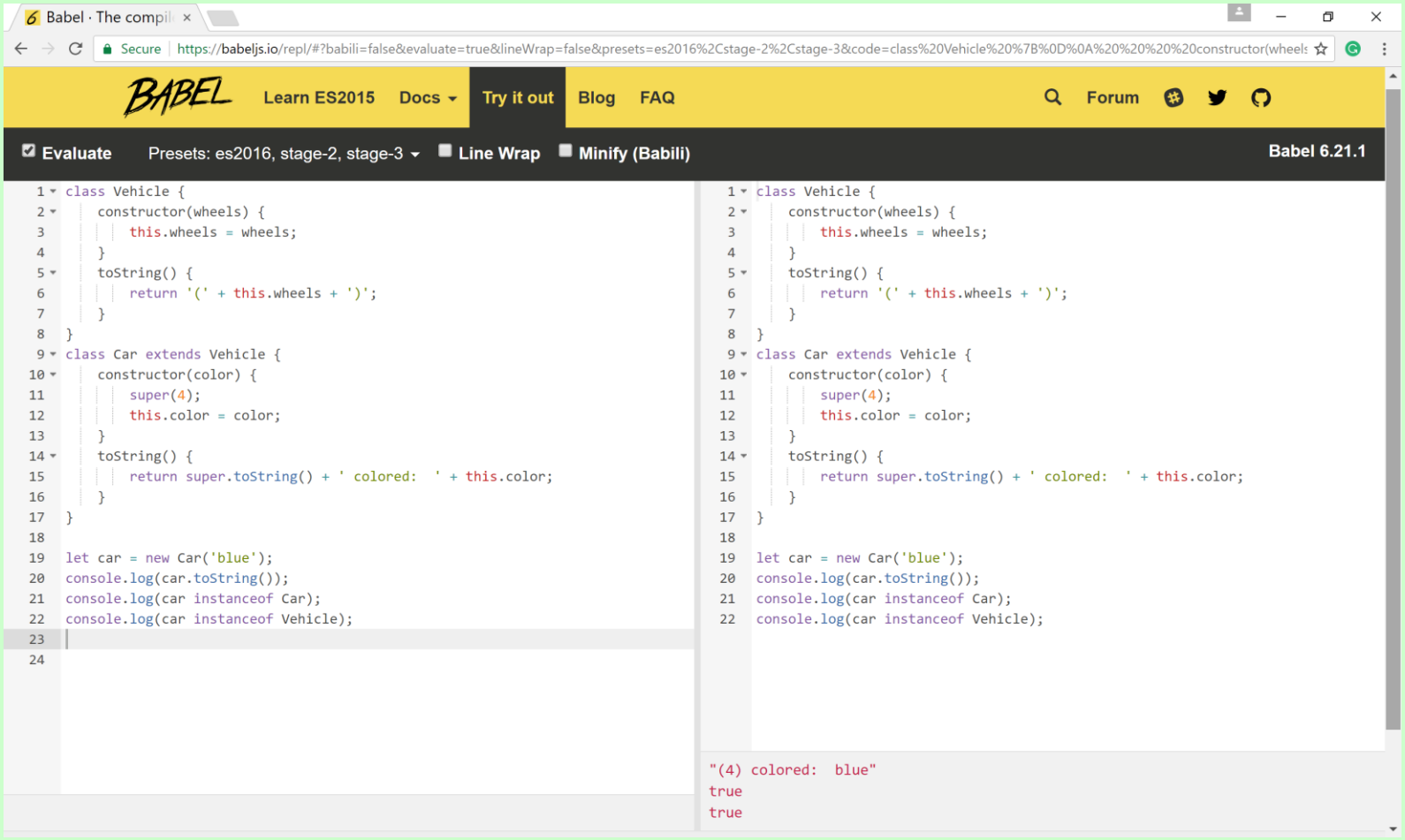
- In this example, the Car class extends the Vehicle class.
- The output is as follows:

  (4) in blue

  true

  true

Class Example on Babel



Class Example in Browser

```
<div id="output"></div>
<!-- Load Babel -->
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
<!-- Your custom script here -->
<script type="text/babel" data-presets="es2016,es2017,stage-2, stage-3">
class Vehicle {
constructor(wheels) {
this.wheels = wheels;
}
toString() {
return '(' + this.wheels + ')';
}
}
class Car extends Vehicle {
constructor(color) {
super(4);
this.color = color;
}
toString() {
return super.toString() + ' colored: ' + this.color;
}
}
let car = new Car('blue');
document.getElementById('output').innerHTML = car.toString() + "<br>";;
document.getElementById('output').innerHTML += (car instanceof Car) + "<br>";
document.getElementById('output').innerHTML += (car instanceof Vehicle);

</script>
```

Arrow functions

- Arrows are ***functions shorthand*** by the **=>** syntax.
- Used in Java and C#.
- Arrows are also very helpful because they share the same lexical this as their scope.
- They are mainly used in two forms.
  - One is using an expression body:

```
const squares = numbers.map(n => n * n);
```

- Another form is using a statement body:

```
numbers.forEach(n => {
  //anonymous function code
  if (n % 2 === 0) evens.push(n);
});
```

Arrow Functions

- **Lexical scoping** means whatever variables are in **scope** where you define a function from (as opposed to when you call it) are in **scope** in the function
- An example of using the shared lexical would be:

```
const author = {
  fullName: "Bob Alice",
  books: [],
  printBooks() {
    this.books.forEach(book => console.log(book + ' by ' +
this.fullName));
  }
};
```

- If used as a regular function, this would be the book object and not the author.

Default, Rest, and Spread

- Default, Rest, and Spread are three new features related to functions parameters.
- The default feature allows you to **set a default value to the function parameter**:

```
function add(x, y = 0) {
    return x + y;
}
add(1)
add(1,2)
```

- In this example, the value of y will be set to 0 if a value is not passed or is set to undefined.

Default, Rest, and Spread

- The Rest feature **allows you to pass an array as trailing arguments** as follows:

```
function userFriends(user, ...friends) {
  console.log(user + ' has ' + friends.length + ' friends');
}
userFriends('User', 'Bob', 'Alice');
```

Default, Rest, and Spread

- The Spread feature **turns an array into a call argument**:

```
function userTopFriends(firstFriend, secondFriend,
thirdFriends) {
  console.log(firstFriend);
  console.log(secondFriend);
  console.log(thirdFriends);
}

userTopFriends(...['Alice', 'Bob', 'Michelle']);
```

References

- Textbook
- https://leanpub.com/ecmascript2015es6guide/read
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import
- http://www.2ality.com/2015/08/getting-started-es6.html
- https://leanpub.com/setting-up-es6/read
- http://exploringjs.com/es2016-es2017/index.html
- http://es6-features.org/