Video 3.8

ES6

Chris Murphy

What is ES6?

- **ES6** stands for ECMAScript 6
- ECMAScript is the ”proper” name for JavaScript
- ES6 is the newest JavaScript Specification, released in 2015

What can you do with ES6?

- In ES6, you can…
  - Define constants
  - Use simpler notations for function declarations
  - Build classes
  - Refactor code into modules
  - Store data in Sets, Maps, and Typed Arrays
  - Copy objects in one line of code
  - … and much more!

ES6 – Arrow Functions

- New syntax for defining functions using arrows

- ES5 Syntax:

```
var arr = [1,2,3,4,5];
var square = function (n) {
   return n*n;
};
arr.forEach( function(v, i) {
   arr[i] = square(v);
});
```

ES6 – Arrow Functions

- New syntax for defining functions using arrows

- ES5 Syntax:

```
var arr = [1,2,3,4,5];
var square = function (n) {
   return n*n;
};
arr.forEach( function(v, i) {
   arr[i] = square(v);
});
```

ES6 – Arrow Functions

- New syntax for defining functions using arrows

- ES5 Syntax:

```
var arr = [1,2,3,4,5];
var square = function (n) {
   return n*n;
};
arr.forEach( function(v, i) {
   arr[i] = square(v);
});
```

ES6 – Arrow Functions

- New syntax for defining functions using arrows

- ES5 Syntax:

```
var arr = [1,2,3,4,5];
var square = function (n) {
   return n*n;
};
arr.forEach( function(v, i) {
   arr[i] = square(v);
});
```

- ES6 Syntax:

```
let arr = [1,2,3,4,5];
let square = n => {
   return n*n;
};
arr.forEach( (v, i) => {
   arr[i] = square(v);
});
```

ES6 – Arrow Functions

- New syntax for defining functions using arrows

- ES5 Syntax:

```
var arr = [1,2,3,4,5];
var square = function (n) {
   return n*n;
};
arr.forEach( function(v, i) {
   arr[i] = square(v);
});
```

- ES6 Syntax:

```
let arr = [1,2,3,4,5];
let square = n => {
   return n*n;
};
arr.forEach( (v, i) => {
   arr[i] = square(v);
});
```

ES6 – Arrow Functions

- New syntax for defining functions using arrows

- ES5 Syntax:

```
var arr = [1,2,3,4,5];
var square = function (n) {
   return n*n;
};
arr.forEach( function(v, i) {
   arr[i] = square(v);
});
```

- ES6 Syntax:

```
let arr = [1,2,3,4,5];
let square = n => {
   return n*n;
};
arr.forEach( (v, i) => {
   arr[i] = square(v);
});
```

ES6 – Arrow Functions

- New syntax for defining functions using arrows

- ES5 Syntax:

```
var arr = [1,2,3,4,5];
var square = function (n) {
   return n*n;
};
arr.forEach( function(v, i) {
   arr[i] = square(v);
});
```

- ES6 Syntax:

```
let arr = [1,2,3,4,5];
let square = n => {
   return n*n;
};
arr.forEach( (v, i) => {
   arr[i] = square(v);
});
```

ES6 – Arrow Functions

- New syntax for defining functions using arrows

- ES5 Syntax:

```
var arr = [1,2,3,4,5];
var square = function (n) {
   return n*n;
};
arr.forEach( function(v, i) {
   arr[i] = square(v);
});
```

- ES6 Syntax:

```
let arr = [1,2,3,4,5];
let square = n => {
   return n*n;
};
arr.forEach( (v, i) => {
   arr[i] = square(v);
});
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};

console.log(pow(3));
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};

console.log(pow(3));
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};

console.log(pow(3));
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};

console.log(pow(3));          // 9
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};

console.log(pow(3));         // 9

console.log(pow(3,3));
```

ES6 – Default Parameter Values

- The "=" symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};

console.log(pow(3));         // 9

console.log(pow(3,3));
```

ES6 – Default Parameter Values

- The “=“ symbol can be used to assign default values to function parameters

```
function pow (base, power = 2) {
   return Math.pow(base, power);
};

console.log(pow(3));          // 9

console.log(pow(3,3));        // 27
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

- ES6 Syntax:

```
var person = { name: "Lydia" };

var msg = `Dear ${person.name},
           How are you?`
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

- ES6 Syntax:

```
var person = { name: "Lydia" };

var msg = `Dear ${person.name},
           How are you?`
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

- ES6 Syntax:

```
var person = { name: "Lydia" };

var msg = `Dear ${person.name},
           How are you?`
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

- ES6 Syntax:

```
var person = { name: "Lydia" };

var msg = `Dear ${person.name},
           How are you?`
```

ES6 – Template Literals

- Can define a template for rendering strings

- ES5 Syntax:

```
var person = { name: "Lydia" };

var msg = "Dear " + person.name + ",\n" + "How are you? ";
```

- ES6 Syntax:

```
var person = { name: "Lydia" };

var msg = `Dear ${person.name},
           How are you?`
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
   this.height = height;
   this.width = width;
}

Rectangle.prototype.area = function () {
    return this.height * this.width;
}
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
    this.height = height;
    this.width = width;
}

Rectangle.prototype.area = function () {
    return this.height * this.width;
}
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
    this.height = height;
    this.width = width;
}

Rectangle.prototype.area = function () {
    return this.height * this.width;
}
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
   this.height = height;
   this.width = width;
}

Rectangle.prototype.area = function () {
    return this.height * this.width;
}
```

- ES6 Syntax:

```
class Rectangle {
   constructor (height, width) {
      this.height = height;
      this.width = width;
   }
   area () {
      return this.height * this.width;
   }
}
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
   this.height = height;
   this.width = width;
}

Rectangle.prototype.area = function () {
    return this.height * this.width;
}
```

- ES6 Syntax:

```
class Rectangle {
   constructor (height, width) {
      this.height = height;
      this.width = width;
   }
   area () {
      return this.height * this.width;
   }
}
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
    this.height = height;
    this.width = width;
}

Rectangle.prototype.area = function () {
     return this.height * this.width;
}
```

- ES6 Syntax:

```
class Rectangle {
    constructor (height, width) {
        this.height = height;
        this.width = width;
    }
    area () {
        return this.height * this.width;
    }
}
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
   this.height = height;
   this.width = width;
}

Rectangle.prototype.area = function () {
    return this.height * this.width;
}
```

- ES6 Syntax:

```
class Rectangle {
   constructor (height, width) {
      this.height = height;
      this.width = width;
   }
   area () {
      return this.height * this.width;
   }
}
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
   this.height = height;
   this.width = width;
}

Rectangle.prototype.area = function () {
    return this.height * this.width;
}
```

- ES6 Syntax:

```
class Rectangle {
   constructor (height, width) {
      this.height = height;
      this.width = width;
   }
   area () {
      return this.height * this.width;
   }
}
```

ES6 – Classes

- Instead of building prototypes, ES6 allows classes to be directly defined in more traditional OOP style
- ES5 Syntax:

```
var Rectangle = function (height, width) {
   this.height = height;
   this.width = width;
}

Rectangle.prototype.area = function () {
    return this.height * this.width;
}
```

- ES6 Syntax:

```
class Rectangle {
   constructor (height, width) {
      this.height = height;
      this.width = width;
   }
   area () {
      return this.height * this.width;
   }
}
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Sets

- ES6 introduces a **`Set`** class
- Elements are distinct and maintain order

```
let s = new Set();

s.add("alligator"); // s = {"alligator"}
s.add("dolphin");   // s = {"alligator", "dolphin"}
s.add("fox");       // s = {"alligator", "dolphin", "fox"}
s.add("alligator"); // s = {"alligator", "dolphin", "fox"}

s.has("alligator"); // true

s.delete("alligator"); // s = {"dolphin", "fox"}

for (let v of s.values())
    console.log(v);    // prints each value in order
```

ES6 Data Structures: Maps

- ES6 also introduces a **`Map`** class
- A **`Set`** of keys is mapped to corresponding values

```
let m = new Map();

m.set("dog", "rover"); // {"dog" => "rover"}
m.set("cat", "felix"); // {"dog" => "rover", "cat" => "felix"}

m.get("cat");    // "felix"
m.get("mouse");  // undefined


for (let [key, val] of m.entries())
    console.log(key + ": " + val); // prints keys and values
```

ES6 Data Structures: Maps

- ES6 also introduces a **`Map`** class
- A **`Set`** of keys is mapped to corresponding values

```
let m = new Map();

m.set("dog", "rover"); // {"dog" => "rover"}
m.set("cat", "felix"); // {"dog" => "rover", "cat" => "felix"}

m.get("cat");    // "felix"
m.get("mouse");  // undefined


for (let [key, val] of m.entries())
    console.log(key + ": " + val); // prints keys and values
```

ES6 Data Structures: Maps

- ES6 also introduces a **`Map`** class
- A **`Set`** of keys is mapped to corresponding values

```
let m = new Map();

m.set("dog", "rover"); // {"dog" => "rover"}
m.set("cat", "felix"); // {"dog" => "rover", "cat" => "felix"}

m.get("cat");    // "felix"
m.get("mouse");  // undefined


for (let [key, val] of m.entries())
    console.log(key + ": " + val); // prints keys and values
```

ES6 Data Structures: Maps

- ES6 also introduces a **`Map`** class
- A **`Set`** of keys is mapped to corresponding values

```
let m = new Map();

m.set("dog", "rover"); // {"dog" => "rover"}
m.set("cat", "felix"); // {"dog" => "rover", "cat" => "felix"}

m.get("cat");    // "felix"
m.get("mouse");  // undefined


for (let [key, val] of m.entries())
    console.log(key + ": " + val); // prints keys and values
```

ES6 Data Structures: Maps

- ES6 also introduces a **`Map`** class
- A **`Set`** of keys is mapped to corresponding values

```
let m = new Map();

m.set("dog", "rover"); // {"dog" => "rover"}
m.set("cat", "felix"); // {"dog" => "rover", "cat" => "felix"}

m.get("cat");    // "felix"
m.get("mouse");  // undefined


for (let [key, val] of m.entries())
    console.log(key + ": " + val); // prints keys and values
```

ES6 Data Structures: Maps

- ES6 also introduces a **`Map`** class
- A **`Set`** of keys is mapped to corresponding values

```
let m = new Map();

m.set("dog", "rover"); // {"dog" => "rover"}
m.set("cat", "felix"); // {"dog" => "rover", "cat" => "felix"}

m.get("cat");    // "felix"
m.get("mouse");  // undefined


for (let [key, val] of m.entries())
    console.log(key + ": " + val); // prints keys and values
```

ES6 Data Structures: Maps

- ES6 also introduces a **`Map`** class
- A **`Set`** of keys is mapped to corresponding values

```
let m = new Map();

m.set("dog", "rover"); // {"dog" => "rover"}
m.set("cat", "felix"); // {"dog" => "rover", "cat" => "felix"}

m.get("cat");    // "felix"
m.get("mouse");  // undefined


for (let [key, val] of m.entries())
    console.log(key + ": " + val); // prints keys and values
```

ES6 Data Structures: Maps

- ES6 also introduces a **`Map`** class
- A **`Set`** of keys is mapped to corresponding values

```
let m = new Map();

m.set("dog", "rover"); // {"dog" => "rover"}
m.set("cat", "felix"); // {"dog" => "rover", "cat" => "felix"}

m.get("cat");    // "felix"
m.get("mouse");  // undefined


for (let [key, val] of m.entries())
    console.log(key + ": " + val); // prints keys and values
```

Summary

- ES6 provides simplified syntax and new libraries and functionality

- We will use ES6 notation in the remaining lessons in the course