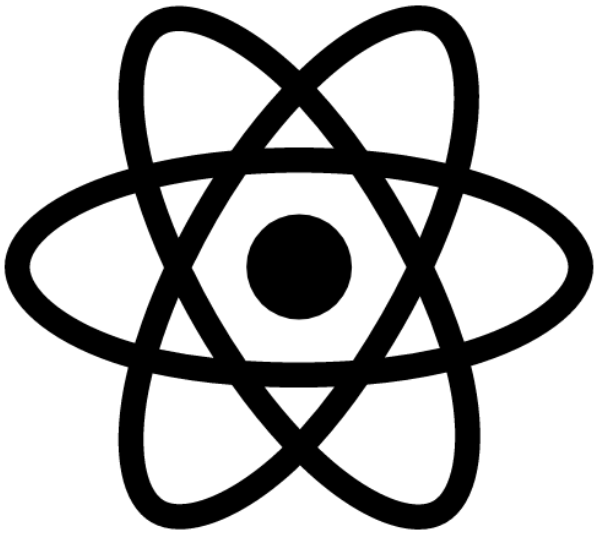

ES6

SENG 4640
Software Engineering for Web Apps
Winter 2023

Sina Keshvadi
Thompson Rivers University

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!

Functional Programming

- Imperative:

```
for (let i = 0; i < anArr.length; i++) {
 newArr[i] = anArr[i] * i;
}
```

- Functional:

```
newArr = anArr.map(function (val, ind) {
 return val * ind;
});
```

- Can write entire program as functions with no side-effects

```
anArr.filter(filterFunc).map(mapFunc).reduce(reduceFunc);
```

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);
};

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

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
```

For of

Old way - Iterator over an array

```
var a = [5, 6, 7];
var sum = 0;
for (var i = 0; i < a.length; i++) {
 sum += a[i];
}
```

New way - Iterate over arrays, strings, Map, Set, without using indexes.

```
let sum = 0;
for (ent of a) {
 sum += ent;
}
```

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 – 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 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
```

Dates

```
let date = new Date();
```

- Are special objects: typeof date == 'object'
- The number of milliseconds since midnight January 1, 1970 UTC
  - Timezone needed to convert.
  - Not good for fixed dates (e.g. birthdays)
- Many methods for returning and setting the data object. For example:
  - date.valueOf() = 1452359316314
  - date.toISOString() ='2016-01-09T17:08:36.314Z'
  - date.toLocaleString() = '1/9/2016, 9:08:36 AM'

Several ways to create a Date object:

```
const today = new Date();

const birthday = new Date("December 17, 1995 03:24:00");

// DISCOURAGED: may not work in all runtimes

const birthday2 = new Date("1995-12-17T03:24:00");

// This is ISO8601-compliant and will work reliably
// "T" indicates the beginning of the time element.

const birthday3 = new Date(1995, 11, 17);

// the month is 0-indexed

const birthday4 = new Date(1995, 11, 17, 3, 24, 0);

const birthday5 = new Date(628021800000);

// passing epoch timestamp
```

Formats of toString method return values

```
const date = new Date("2020-05-12T23:50:21.817Z");
date.toString();
// Tue May 12 2020 18:50:21 GMT-0500 (Central Daylight Time)
// Z is the UTC offset representation
date.toDateString(); // Tue May 12 2020
date.toTimeString(); // 18:50:21 GMT-0500 (Central Daylight Time)
date.toISOString(); // 2020-05-12T23:50:21.817Z
date.toUTCString(); // Tue, 12 May 2020 23:50:21 GMT
date.toJSON(); // 2020-05-12T23:50:21.817Z
date.toLocaleString(); // 5/12/2020, 6:50:21 PM
date.toLocaleDateString(); // 5/12/2020
date.toLocaleTimeString(); // 6:50:21 PM
```

More on Date - Reference

JavaScript: The Bad Parts

- Declaring variables on use
  - Workaround: Force declarations
  - let myVar = 100;
- Automatic semicolon insertion
  - Workaround: Enforce semicolons
- Type coercing equals: ==
  - Workaround: Always use ===,!== instead
  - ("" == "0") is false but (0 == "") is true
  - (false == '0') is true as is (null == undefined)
- with, eval
  - Workaround: Don't use

Some JavaScript idioms

- Assign a default value
  - `hostname = hostname || "localhost";`
  - `port = port || 80;`
- Access a possibly undefined object property
  - `let prop = obj && obj.propname;`

Summary

- ES6 provides simplified syntax and new libraries and functionality

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