ES6

Dr. Jose Annunziato

VARIABLES

Dr. Jose Annunziato

JavaScripts 6 (ES6) Variables

- Three different ways to declare variables

```
var x = 123 // variable hoisting, can use before declaring
let y = 234 // error if used before declaring
const PI = 3.141593 // set value only at declaration
```

- Best practice
  - Declare variables at top of scope before using
  - Prefer **let** and **const** over **var**

JavaScript Object Notation (JSON)

```
let someObject = {
 anObjectProperty: {
   anotherObjectProp: {q: 111, w: 222},
   anotherArrayProp: [321, 432, 543]
 },
 aNumberArrayProp: [1, 2, 3],
 anObjectArrayProp: [
   {a: 123, b: 234},   {a: 321, b: 432}]}

console.log(someObject.anObjectProperty
     .anotherArrayProp[2]) // ==> 543
```

Variables can be scoped in code blocks

```
var i, x, a = [1, 2, 3]
for (i = 0;
    i < a.length;
    i++) {
 x = a[i];
}
```

ES5

```
let a = [1, 2, 3]
for (let i = 0;
    i < a.length;
    i++) {
 let x = a[i];
}
```

ES6

Scoped variables and closures

```
var callbacks = []
for (var i = 0; i <= 2; i++) {
 (function (i) {
   callbacks[i] = function() {
     return i * 2 }
 })(i);
}
callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```

```
let callbacks = []
for (let i = 0; i <= 2; i++) {

 callbacks[i] = function () {
     return i * 2 }
}

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

Variables can be scoped in code blocks

```
(function () {
 var foo = function ()
      { return 1 }
 foo() === 1
 (function () {
   var foo = function ()
      { return 2 }
   foo() === 2
 })();
 foo() === 1
})()
```

```
{
 function foo ()
      { return 1 }
 foo() === 1
 {
   function foo ()
      { return 2 }
   foo() === 2
 }
 foo() === 1
}
```

FUNCTIONS

Dr. Jose Annunziato

Arrow Functions

ES5

```
function addEs5(a, b) {
 console.log(a, b);
 return a + b;
}
```

ES6

```
const addEs6 = (a, b) => {
 console.log(a, b);
 return a + b;
};
```

Single Line *Implied Return*

- ES5

```
function addEs5(a, b) {
 return a + b;
}
```

- ES6

```
const addEs6 = (a, b) => a + b;
// return is optional if one line body return
```

Single Argument Optional Parens

- ES5

```
function squareEs5 (b) {
 return b * b
}
```

- ES6

```
const squareEs6 = b => b * b
// parenthesis is optional if one argument
```

This keyword

- ES5 variant 1

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

- ES5 variant 2

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

- ES6 this behaves same as other modern languages

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

Default Parameters

```
const f = (x, y = 7, z = 42) => {
    return x + y + z;
}
f(1) === 50;
```

ARRAYS

Dr. Jose Annunziato

Filtering even/odd numbers

Use ***filter*** to compute a subset of array

```
let all = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
let even = all.filter((i) => { return i % 2 === 0 })
let odd  = all.filter(i => i % 2 !== 0)
```

Note syntax difference/equivalence!

Return ***true*** to keep, ***false*** to skip item

Filter with Old Function Notation

ES6 Arrow functions are less verbose

```
let all = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

ES6:

```
let even = all.filter(i => i % 2 === 0)
```

ES5:

```
let even = all.filter(function (i) {return i % 2 === 0})
```

Careful if you need additional lines!

- ES6:

```
let even = all.filter(i => {
    console.log(i)
    return i % 2 === 0})
```

- ES5:

```
let even = all.filter(function (i) {
    console.log(i)
    return i % 2 === 0})
```

Implicit returns only work if it's one line. You'll need explicit return if body has more than one line

Mapping

- Use ***map*** to collate results into new array

```
let all = [1, 2, 3, 4]
let square = all.map(i => i * i)
// square = [1, 4, 9, 16]
```

Finding & Filtering

```
let array = [ 1, 3, 4, 2, 5 ]
array.find(x => x > 3)          // ⇒ 4
array.findIndex(x => x > 3)     // ⇒ 2
array.filter(x => x > 3)        // ⇒ [4, 5]
```

STRING INTERPOLATION

Jose Annunziato

Use interpolation instead of concatenation

```
const customer = { name: "Alice" }
const card = { amount: 7, product: "Bar", unitprice: 42 }
let message = `Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?`
console.log(message)
```

DESTRUCTURING ASSIGNMENT

Dr. Jose Annunziato

Array Matching

- Flexible Array destructuring into variables during assignment
- ES5:

```
var list = [ 1, 2, 3 ];
var a = list[0], b = list[2];
var tmp = a; a = b; b = tmp;
```

- ES6:

```
const list = [ 1, 2, 3 ]
let [ a, , b ] = list          // a = list[0]; b = list[2]
[ b, a ] = [ a, b ]            // swapping a and b
```

Object Matching, Shorthand Notation

- Flexible Object destructuring into variables during assignment
- ES5

```
const tmp = {a: "123", b: "234", c: "345", d: "456"}
var a  = tmp.a;
var c = tmp.c;
```

- ES6

```
let { a, c } = tmp
```

Parameter Matching

- Consider the following ES5

```
function h (arg) {
 var name = arg.name
 var val  = arg.val
 console.log(arg.name, arg.val)
}
h({ name: "bar", val: 42 })
```

Parameter Matching

- In ES6 use shorthand parameter matching

```
const h = ({ name, val }) => console.log(name, val)
h({ name: "bar", val: 42 })
```

Order is irrelevant here

OR

```
const h = ({ val, name }) => console.log(name, val)
h({ name: "bar", val: 42 })
```

Renaming matched parameters

```
const h = ({ name : n, val : v }) => {
    console.log(n, v)
}
h({ name: "bar", val: 42 })
```

Also works with arrays

```
function f ([ name, val ]) {
    console.log(name, val)
}
f([ "bar", 42 ])
```

Property Shorthand

- If property names are same as values:

```
var x = 0, y = 0;
obj = { x: x, y: y };
```

- Use shorthand instead

```
obj = { x, y };
```

MODULES

Dr. Jose Annunziato

Exporting and Importing

- Support for exporting/importing values from/to modules without global namespace pollution

```
// 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 & Wildcard

- Marking a value as the default exported value and mass-mixin of values

```
// lib/mathplusplus.js
export * from "lib/math"
export var e = 2.71828182846
export default (x) => Math.exp(x)
// someApp.js
import exp, { pi, e } from "lib/mathplusplus"
console.log("e^{π} = " + exp(pi))
```

CLASSES

Dr. Jose Annunziato

Class Definition

- Intuitive, OOP-style and boilerplate-free classes

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

Class Inheritance

```
class Rectangle extends Shape {
 constructor (id, x, y, width, height) {
   super(id, x, y)  // must be first line in constructor
   this.width  = width
   this.height = height
 }}
class Circle extends Shape {
 constructor (id, x, y, radius) {
   super(id, x, y)
   this.radius = radius
 }}
```

Getter/Setter

```
class Rectangle {
 constructor (width, height) {
   this._width  = width
   this._height = height}
 set width  (width)    { this._width = width             }
 get width  ()         { return this._width               }
 set height (height)   { this._height = height            }
 get height ()         { return this._height              }
 get area   ()         { return this._width * this._height }}
var r = new Rectangle(50, 20)       r.area === 1000
```

Static Members

```
class Rectangle extends Shape {
 // ...
 static defaultRectangle () {
   return new Rectangle("default", 0, 0, 100, 100)
 }
}
var defRectangle = Rectangle.defaultRectangle()
```

Base Class Access

```
class Shape {
 toString () {
   return `Shape(${this.id})` }}
class Rectangle extends Shape {
 constructor (id, x, y, width, height) {
   super(id, x, y) }
 toString () {
   return "Rectangle > " + super.toString()}}
```