Web Programming 03/14

* More on JavaScript
* ~~Git Basics~~

More on JavaScript

- More on Types / Variables / Objects
- More on Functions
- Revisiting Ajax, DOM manipulation & Event listening
- ES-6 New Stuffs
- ~~Git Basics~~

More on Types

- Recall: 所有 JavaScript 的數字都是用 double 來表示，而基本上的算術邏輯操作就跟 C/C++ 一樣。但一些只有整數才有意義的操作要怎麼解釋呢？
  - Shift (<<, >>) operator: 將數字轉成最多 32-bit 的 signed int 來進行操作
- Numeric values or string?
  - “30” + 8 ==> “308”
  - “30” - 8 ==> 22, “30” * 8 ==> 240, “30” / 8 = 3.75
  - “30” << 8 ==> 7680, “30 >> 8 ==> 0
- parseInt(string, base) / / base = [2,36]
  - parseInt(“0xF”) ==> parseInt(“0xF”, 16) ==> 15
  - parseInt(“15*3”, 10) ==> parseInt(“15”, 10) ==> 15
  - parseInt(021, 8) ==> parseInt(“017”, 8) ==> 15
  - parseInt(“321”, 2) ==> NaN
  - See also: parseFloat(“string”);

More on Objects・Constructing Objects

- Recall: 這兩種寫法效果是一樣的
  - function Student(name, score) {
    this.name = name;
    this.score = score;
    };
  - var Student = function (name, score) {
    this.name = name;
    this.score = score;
    };
- "Student" as a constructor
  - var ric = new Student("Ric", 100);
  - var mary = new Student("Mary", 20);
  - ric.age = 48; // this property “age” won’t go into “mary”

More on Objects・Properties and Values

- 使用 .propertyName 或 [“propertyName”] 來存取 property 的 value
  // using the Student example in the previous page
  - ric.name; // “Ric”
  - ric."name"; // ERROR!!
  - ric[name]; // undefined
  - ric["name"]; // “Ric”
  - ric[0]; // undefined

More on Objects・Properties and Values

- 使用 [] 來存取 object 的 property 有一個好處是可以 “動態的設定” property name
  - var a;
    if (someExpression) { a = "A"; }
    else { a = "B"; }
    / / assume someExpression is false
    ric[a] = 70;
    console.log(ric.A); / / undefined
    console.log(ric.B); / / 70

More on Objects ◦ Properties and Values

- However, property name can almost be anything…
  [e.g.] var car = { manyCars: {a: "Saab", **"b"**: "Jeep"}, **7**: "Mazda",
  "": "An empty string", **"!"**: "Bang!" };
  - car.manyCars.a; / / "Saab" car.manyCars."a"; / / ERROR!!
  - car.manyCars.b; / / "Jeep" car.manyCars."b"; / / ERROR!!
  - car.manyCars[a]; / / undefined car.manyCars["a"]; / / "Saab"
  - car.manyCars[b]; / / undefined car.manyCars["b"]; / / "Jeep"
  - car."7"; / / ERROR!! car.7; / / ERROR!!
    car["7"] = "Mazda"; car[7]; / / "Mazda"
  - car.""; / / ERROR!! car[""]; / / "An empty string"
  - car."!"; / / ERROR!! car.!; / / ERROR!! car["!"]; / / "Bang!"

More on Objects・Constructing Objects

- 使用內建 Object type
  - var ric = new Object({
    name: "Ric",
    score: 100,
    sayHi: function() { alert("Hi, I am " + this.name + "."); },
    });
- 使用 Object.create()　// IE8 不支援 (就算了... 誤)
  - var ric = { name: "Ric",
    sayHi: function() { alert("Hi, I am " + this.name + "."); } };
  - var mary = Object.create(ric);
  - console.log(mary);　　console.log(mary.name);
  - mary.name = "Mary";　mary.sayHi();
  - // More about Inheritance and Object.create() later

More on Objects・Constructing Objects

- 使用 **constructor** property — points to the constructor function.
  - var Student = function (name) { this.name = name; }
  - var ric = new Student("Ric");
  - ric.constructor; // return the anonymous function
  - ric.constructor.name; // “Student”
  - ric instanceof Student; // true
  - var mary = new ric.**constructor**("Mary");
  - console.log(mary);
  - console.log(ric.**__proto__** === Student.**prototype**); // true

More on Objects ◦ Prototype and Inheritance

- Recall: JavaScript is a "prototype-based language"
- Each object has a **prototype** object, which acts as a template object that it inherits methods and properties from.
  - Note: Nearly all objects in JavaScript are instances of **Object**, and a typical object inherits properties (including methods) from **Object.prototype**
- var Person = function(name) { this.name = name; };
  var person1 = new Person("Ric");

Prototype chain of "person1"

More on Objects ◦ Prototype and Inheritance

[e.g.] var Student = function (name) { this.name = name; }
var ric = new Student("Ric");

- Modify prototype
  - Student.**prototype**.sayHi
    = function() { alert("Hi, I am " + this.name + "."); }
  - ric.sayHi(); // Hi, I am Ric.
- Suggested practice:
  - 將 properties 寫在 constructor function 裡面
  - 將 methods 寫在 constructor function 外面
    (i.e. 利用 prototype)

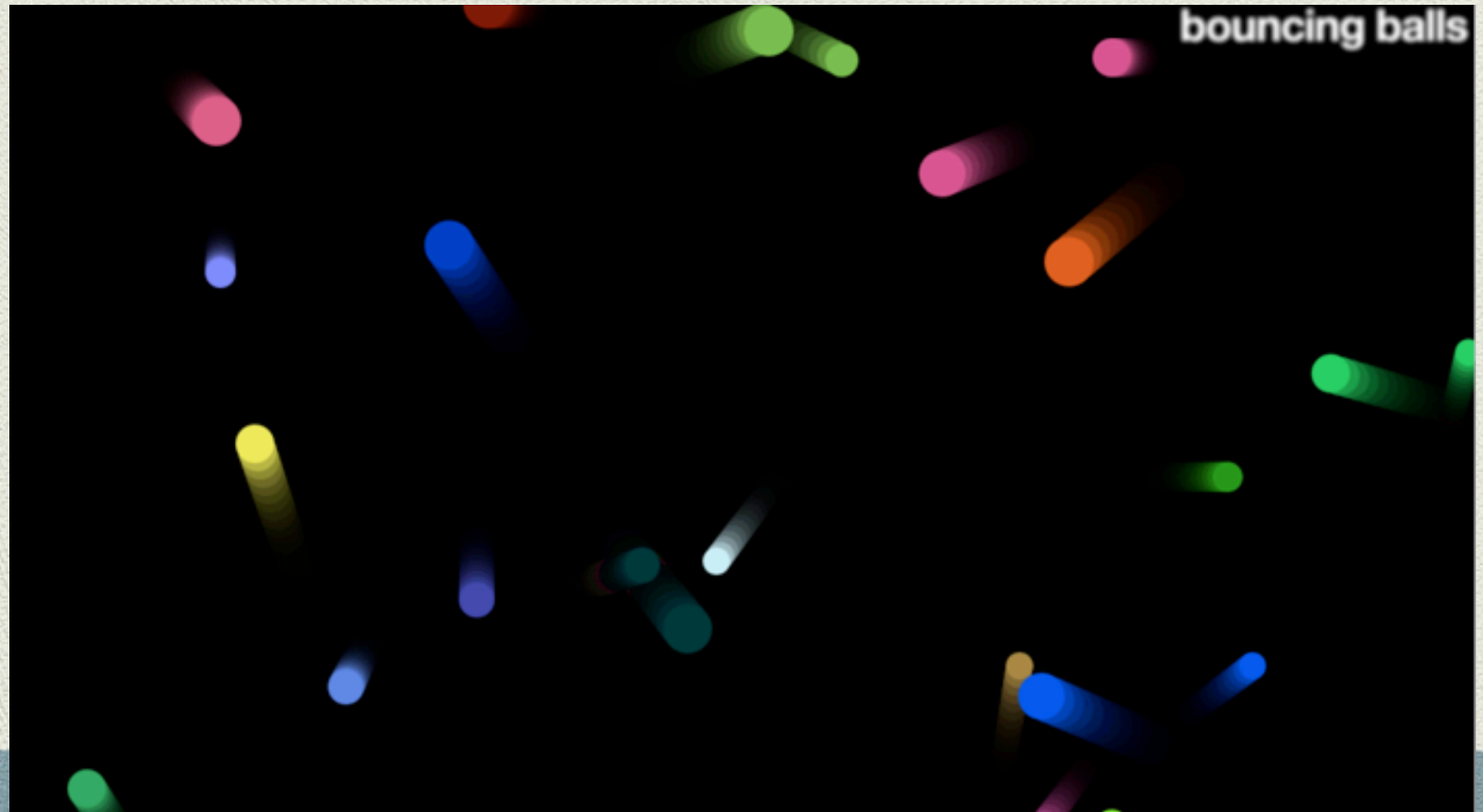
More on Objects ◦ Prototype and Inheritance

[e.g.] function Person(name) { this.name = name; }
Person.prototype.sayHi = function() { alert("Hi, I am " + this.name + "."); }

- Creating a class Teacher that inherits Person
  - function Teacher(name, subject) {
    Person.call(this, name);
    this.subject = subject;
    }
  - Teacher.prototype.sayHi; // undefined
  - Teacher.prototype = Object.create(Person.prototype);
  - Teacher.prototype.constructor; // Oops! It becomes “Person”
  - Teacher.prototype.constructor = Teacher; // Done!!
- Overloading Person’s sayHi() function
  - Teacher.prototype.sayHi = function() { alert("I teach " + this.subject + "."); }

JavaScript Practice #1 ◦ Bouncing Ball

- Download the code “bouncingBall.example.tgz” from Ceiba
- Open “index.html” on web browser. What do you see?
  - Why isn’t “<h1>bouncing balls</h1>” not shown?
- Our goal is:



JavaScript Practice #1 ◦ Bouncing Ball

- We will use Canvas API for this practice.
  First, let's create a Canvas object in the script:
  - var canvas = document.querySelector("canvas");
  - var ctx = canvas.getContext("2d");

// "ctx" is the object that directly represents the drawing
// area of the canvas and allows us to draw 2D shapes on it.

- var width = canvas.width = window.innerWidth;
- var height = canvas.height = window.innerHeight;
- ctx.fillStyle = "rgba(0, 0, 0, 0.25)";
- ctx.fillRect(0, 0, width, height);
- Why isn't it black (i.e. rgb = (0,0,0))?

Ref: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_building_practice

JavaScript Practice #1 ◦ Bouncing Ball

- Now, let's define a Ball object
  - function Ball(x, y, vx, vy, color, r) { … };
  - Use "Math.random()" and "Math.floor()" to create random effect.
    => Create a "random(…)" function for reusability

- To draw the ball, let's add a method to the Ball prototype
  - Ball.prototype.draw = function() { … }
  - **Start drawing a path**: ctx.beginPath();
  - **Define the fill style**: ctx.fillStyle = this.color;
  - **Draw a circle (i.e. an arc with degree = 2π)**:
    ctx.arc(this.x, this.y, this.r, 0, 2 * Math.PI);
  - **Fill the circle**: ctx.fill();

Ref: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_building_practice

JavaScript Practice #1 ◦ Bouncing Ball

- Let's draw many balls…
  - Create a "balls" array
  - Using "for" loop to create Ball objects (use "let" and "const")

- To move the ball, let's add a method to the Ball prototype
  - Ball.prototype.move = function() { … }
  - Remember to check the window boundary

- To detect the collision, let's add another method to the Ball prototype
  - Ball.prototype.checkCollision() { … }
  - Use "Math.sqrt(…)" to compute square root
  - When collision occurs, do something to the balls (e.g. change colors)

Ref: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_building_practice

JavaScript Practice #1 ◦ Discussions

- How do we create “trailing effect”?
- What does “alpha” value 0.25 do in this example?
- How can we make it more fun by adding some interactions?

Ref: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_building_practice

JavaScript Practice #2
◦ Inheritance and Event Handling

- Let's Add some new requirements to the bouncing ball example…

1. Create a "evil ball" that will eat the regular balls (when touched)
2. Use the keyboard to control the "evil ball"
3. When a regular ball is eaten, it should disappear! Optionally, you should generate a new regular ball in a random place
4. Keep the score for how many balls the evil one has eaten

JavaScript Practice #2
◦ Inheritance and Event Handling

1. Create a "evil ball" that will eat the regular balls (when touched)
   - Change Ball to a base object, and create EvilBall and RegBall to inherit it
   - Modify/Add "move()" and "checkCollision()" accordingly

2. Use the keyboard to control the "evil ball"
   - Add event listener to the keyboard
     ==> EvilBall.prototype.checkKey() { window.onkeydown = function(e) { … }
   - [Note] Pay special attention to "this"!!
     ==> "this" for "checkKey() vs. "this" for the event handler

3. When a regular ball is eaten, it should disappear! Optionally, you should generate a new regular ball in a random place
   - Modify/Add "move()", "checkCollision()" or "draw()" accordingly

4. Keep the score for how many balls the evil one has eaten
   - Add a <p> in HTML to display score, and add an event listener to it

More on Functions ◦ Function.prototype.call()

- [Recall] When a function is called, “this” in the function usually refers to the function object itself.
  - What can we do if we want to call a function by another object (i.e. not in the inheritance chain), but we want to pass this object to the function as its “this”?

- function someFunc() { console.log(this.name + " 到此一遊"); }
  var ric = { name: "Ric", score: "100",
  hi: function() { someFunc.call(this); } }
  ric.hi(); // Ric 到此一遊
  var mary = { name: "Mary", gender: "unknown" }
  someFunc.call(mary); // Mary 到此一遊

More on Functions ◦ Function.prototype.call()

- **Using call() to chain constructors for an object**
  - function Product(name, price) { this.name = name; this.price = price; }
  - function Food(name, price) {
  - Product.call(this, name, price);
  - this.category = 'food';
  - }
  - function Toy(name, price) {
  - Product.call(this, name, price);
  - this.category = 'toy';
  - }
  - var cheese = new Food('feta', 5);
  - var fun = new Toy('robot', 40);

More on Functions ◦ Function.prototype.call()

- **Using call() to invoke an anonymous function**
  - var animals = [{ species: 'Lion', name: 'King' },
    { species: 'Whale', name: 'Fail' }];
  - for (var i = 0; i < animals.length; i++) {
  - (function(i) {
  - this.print = function() {
  - console.log('#' + i + ' ' + this.species + ': ' + this.name);
  - }
  - this.print();
  - }).call(animals[i], i);
  - }

More on Functions ∘ Function.prototype.call()

- **Using call() to invoke a function and specifying the context for 'this'**

```
function greet() {
    var reply = [this.animal, 'typically sleep between',
                 this.sleepDuration].join(' ');
    console.log(reply);
}
var obj = {
    animal: 'cats', sleepDuration: '12 and 16 hours'
};
greet.call(obj);  // cats typically sleep between 12 and 16 hours
```

More on Functions ◦ Function.prototype.call()

- **Using call() to invoke a function and without specifying the first argument**
  - // If the first argument of call() is not passed,
    // the value of this is bound to the global object.
  - var sData = 'Wisen';
  - function display(){
  - console.log('sData value is %s ', this.sData);
  - }
  - display.call(); // sData value is Wisen

More on Functions・Function.prototype.apply()

- 跟 **call()** 很像，只是 **apply()** 的第二個 argument (可省) 是一個 array, 而不是 list of arguments
  - // If the first argument is not null,
    // it refers to the global object.
- var numbers = [5, 6, 2, 3, 7];
- var max = Math.max.apply(null, numbers);
- // This about equal to Math.max(numbers[0], ...)
- // or Math.max(5, 6, ...)
- var min = Math.min.apply(null, numbers);

More on Functions ◦ Closure

- Consider this example:
  - function init() {
  - var name = 'Mozilla'; // name is a local variable
  - function displayName() { // displayName() is inner, a closure
  - alert(name); // use variable declared in the parent function
  - }
  - displayName();
  - }
  - init(); // no returned value

More on Functions ◦ Closure

- If we revise this example…
  - function makeFunc() {
  - var name = 'Mozilla'; / / name is a local variable
  - function displayName() { / / displayName() is inner, a closure
  - alert(name); / / use variable declared in the parent function
  - }
  - **return** displayName();
  - }
  - **var myFunc** = makeFunc(); / / function object
  - myFunc(); / / Able to display a local variable in “makeFunc()”

More on Functions ∘ Closure

- Summary: The inner function, a closure (i.e. displayName()), maintains a reference to the lexical scope of the local variable. Therefore, when the inner function is returned by the outer function (i.e. makeFunc), the referenced caller (i.e. myFunc) will be able to operate on the local variable.

More on Functions

◦ Closure (A Practical Example)

- Consider we have a dynamic webpage —
  - 12
  - 14
  - 16
  - ————-
  - document.getElementById('size-12').onclick = size12;
  - document.getElementById('size-14').onclick = size14;
  - document.getElementById('size-16').onclick = size16;
- Should “size12”, “size14”, “size16” be 3 different functions?

More on Functions

◦ Closure (A Practical Example)

- Using closure —
  - function makeSizer(size) {
  - return function() {
  - document.body.style.fontSize = size + 'px';
  - };
  - }
  - var size12 = makeSizer(12);
  - var size14 = makeSizer(14);
  - var size16 = makeSizer(16);
- Find more examples on: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

More on Functions ◦ Arguments Object

- The arguments object is an Array-like object corresponding to the arguments passed to a function.
  - function func1(a, b, c) {
  - console.log(arguments[0]);
  - console.log(arguments[1]);
  - console.log(arguments[2]);
  - }
  - func1(1, 2, 3); // 1, 2, 3

More on Functions ◦ Default and Rest Arguments

- In JavaScript, default arguments are optional for any parameter in the list.
  - function func1(a = 1, b, c = 0) { … }
- When calling a function, if there is a missing argument, “undefined” will be assumed.
- The rest parameter syntax allows us to represent an indefinite number of arguments as an array.
  - function fun1(...args) { console.log(theArgs.length); }
    fun1(5, 6, 7); / / 3
  - function f(...[a, b, c]) { return a + b + c;}
    f(1) / / NaN (b and c are undefined)
    f(1, 2, 3) / / 6
    f(1, 2, 3, 4) / / 6 (the fourth parameter is not destructured)

More on Functions • Getter

- The **get** syntax binds an object property to a function that will be called when that property is looked up.
  - var obj = {
  - log: ['example','test'],
  - get latest() {
  - if (this.log.length == 0) return undefined;
  - return this.log[this.log.length - 1];
  - }
  - }
  - console.log(obj.latest); / / "test".
  - delete obj.latest; / / obj.latest becomes undefined

More on Functions • Setter

- The **set** syntax binds an object property to a function to be called when there is an attempt to set that property.
  - var language = {
  - set current(name) { this.log.push(name); }, log: []
  - }
  - language.current = 'EN';
  - console.log(language.log); / / ['EN']
  - language.current = 'FA';
  - console.log(language.log); / / ['EN', 'FA']
  - delete language.current; / / becomes undefined

New JavaScript • ES 2015 and later

- ECMAScript (or ES) is a trademarked scripting-language spec by Ecma International.
  - JavaScript has remained the best-known implementation of ECMAScript since the standard was first published in 1997.

- ES-3 was published in 1999, and ES-4, a more ambitious version and originally targeted for 2008, was abandoned due to some political disputes (especially between Mozilla and Microsoft)
- ES-5, a compromised, less ambitious version, was published in 2009
- ES-6 (2015), added several significant new syntax such as class, arrow function, destructuring, generator, let+const… ==> We will cover these later
- ES-7(2016), ES-8(2017), introduced exponentiation operator (**), await/ async, etc.

ES-6 ◦ Arrow Functions

- An arrow function expression has a shorter syntax than a function expression and does not have its own **this**, **arguments**, **super**, or **new.target**. These function expressions are best suited for non-method functions, and they cannot be used as constructors.

- Syntax
  - (param1, param2, …, paramN) => { statements }
  - (param1, param2, …, paramN) => expression
  - // equivalent to: => { return expression; }

  - // Parentheses are optional when there's only one parameter name:
  - singleParam => { statements }

  - // No parameter case.
  - () => { statements }

ES-6 ◦ Arrow Functions

- More Arrow Function Syntax
  - // Parenthesize the body of function to return an object literal expression:
  - params => ({foo: bar})

  - // Rest parameters and default parameters are supported
  - (param1, param2, ...rest) => { statements }
  - (param1 = defaultValue1, param2, …, paramN = defaultValueN) => {
  - statements }

  - // Destructuring assignments within the parameter list is also supported
  - var f = ([a, b] = [1, 2], {x: c} = {x: a + b}) => a + b + c; f(); // 6

ES-6 ◦ Classes

- JavaScript classes, introduced in ECMAScript 2015, are primarily **syntactical sugar** over JavaScript's existing prototype-based inheritance. The class syntax does not introduce a new object-oriented inheritance model to JavaScript.

ES-6 ◦ Classes

- Class declaration
  - class Rectangle {
    constructor(height, width) {
    this.height = height;
    this.width = width;
    }
    }
  - var Rectangle = class { …. }
  - var Rectangle = class Rectangle { …. }
- No class hoisting!!
  - var p = new Rectangle(); / / Error!!
  - class Rectangle {}

ES-6 ∘ Class Methods

- class Rectangle {
  constructor(height, width) { this.height = height; this.width = width; }
  // Getter
  get area() { return this.calcArea(); }
  // Method
  calcArea() { return this.height * this.width; }
  }
- const square = new Rectangle(10, 10);
- console.log(square.area); // 100

ES-6 ∘ Static Class Methods

- Static methods are called without instantiating their class and cannot be called through a class instance. Static methods are often used to create utility functions for an application.
- class Point {
  constructor(x, y) { this.x = x; this.y = y; }
  static distance(a, b) { const dx = a.x - b.x; const dy = a.y - b.y;
  return Math.hypot(dx, dy); }
  }
- const p1 = new Point(5, 5);
- const p2 = new Point(10, 10);
- console.log(Point.distance(p1, p2)); / / 7.071067811865475

ES-6 • Sub Classing with extends

- class Animal {
  constructor(name) { this.name = name; }
  speak() { console.log(this.name + ' makes a noise.’); }
  }
- class Dog **extends** Animal {
  speak() { console.log(this.name + ' barks.’); }
  }
- var d = new Dog('Mitzie');
- d.speak(); / / Mitzie barks.

ES-6 • Sub Classing with extends

```
// extend traditional function-based "classes"
function Animal (name) { this.name = name; }
Animal.prototype.speak = function () {
    console.log(this.name + ' makes a noise.’);
}
class Dog extends Animal {
    speak() { console.log(this.name + ' barks.’); }
}
var d = new Dog('Mitzie');
d.speak(); // Mitzie barks.
```

ES-6 ◦ Super class calls with super

- class Cat {
  constructor(name) { this.name = name; }
  speak() { console.log(this.name + ' makes a noise.’); }
  }
- class Lion extends Cat {
  speak() {
  super.speak();
  console.log(this.name + ' roars.');
  }
  }
- var l = new Lion('Fuzzy');
- l.speak();
- // Fuzzy makes a noise.
- // Fuzzy roars.

After Class・Week #1

- **Homework assignment**
  1. 完成講義 p18-19 的 JS Practice #2
  2. 完成交換 wireframe 拿到的網頁設計
  3. 完成 “TODO List” 網頁設計 (spec will be posted on Ceiba)
- **Deadline**: 5pm, 03/28(Wed).
- 以上網頁，again, 請用 native html/JS 完成，但可以使用別人的 CSS style file (不要用任何的模組或是框架)

After Class・Week #1

- **Readings /Other actions**
  - **安裝 Git, 申請 Github 帳號**
    - 請安裝 command line, 避免使用 GUI 工具：https://git-scm.com/book/zh-tw/v2 (chap 1.4 開始)
- **Git tutorial**
  - (30 mins video) https://www.youtube.com/watch?v=HVsySz-h9r4
  - Codeschool Try Git: https://www.codeschool.com/courses/try-git
- **安裝 ESLint (for 你習慣的編輯器)**
- **React (ref provided later)**
- **Basic node.js, express (ref provided later)**