**Object.prototype.proto**

**Dog.prototype = Animal.prototype.**

We don't do this because that means any function added to Dog would also be added to Animal.

We want : **Dog.prototype.\_\_proto\_\_ === Animal.prototype**

This way when an attribute is not found on Dog.prototype and the interpreter looks at whatever is set equal to Dog.prototype.\_\_proto\_\_, it will look at Animal.prototype. Dog objects will then have access to methods defined on the Animal prototype.

**\_\_proto\_\_** is **bad** practice because it was just standardized in ES2015 and is not supported across all browsers

**Object.setPrototypeOf**

Object.setPrototypeOf(Dog.prototype, Animal.prototype)

Like \_\_proto\_\_, however, using Object.setPrototypeOf comes with a performance hit because it alters an object's prototype.

**Object.create**

Dog.prototype = Object.create(Animal.prototype); // Dog now inherits from Animal

Dog.prototype.constructor = Dog // otherwise instances of Dog will have 'instance.constructor === Animal'

Object.create returns an entirely new object with its \_\_proto\_\_ set to whatever argument is passed to Object.create. We then set the object returned by Object.create to be the prototype of the child constructor function.

function Dog (name, coatColor) {

// call super-constructor function on \*\*the current `Dog` instance\*\*.

Animal.call(this, name);

// `Dog`-specific initialization

this.coatColor = coatColor;}

**The (old) Surrogate trick**

// A surrogate will be used to construct `Dog.prototype`.

// A `Surrogate` instance should delegate to `Animal.prototype`.

function Surrogate () {};

Surrogate.prototype = Animal.prototype;

// Set `Dog.prototype` to a `Surrogate` instance.

Dog.prototype = new Surrogate();

// Make sure that instances of Dog have instance.constructor === Dog (rather than Animal)

Dog.prototype.constructor = Dog;

**Inheritance in ES2015**

Class Dog extends Animal is the syntactic-sugar equivalent of using Object.create.

class Bicycle {

constructor(color, model) {

this.color = color;

this.model = model;

}

action() {

return "rolls along";

}

}

class RaceBicycle extends Bicycle {

constructor(color, model, gears) {

super(color, model);

this.gears = gears;

}

action() {

const oldAction = super.action();

return `${oldAction} at a blistering pace!`

}

}

**Module Bundlers**

const Animal = require('./animal.js');

module.exports = Lion;

npm install -g webpack webpack-cli

webpack app.js -o bundle.js --mode=development

<script src="bundle.js"></script> in html file.