**Inheritance**

**Inheritance** refers to an object being able to inherit methods and properties from a parent object (a Class in other OOP language, or a Function in JavaScript).

An **instance** is an implementation of a Function. In simple terms, it is a copy of a Function or object. For example:

// Girl is a constructor function because we will use new keyword to invoke it.

function Girl (typeOfGirl) {}

// name is an instance of Girl.

var name = new Girl ("Rama");

In the above example, name is an object that was created from the Girl constructor function. We say that the name object is an instance of the Girl object. Girl is both an object and a function, because functions are objects in JavaScript. name can have its own methods and properties and inherit methods and properties from the Girl object.

two universal principles—creating objects (especially from constructor Functions) and allowing objects to inherit properties and methods—are the main concepts with OOP in JavaScript.

**Inheriting properties**

// Let's create an object o from function f with its own properties a and b:

let f = function () {

this.a = 1;

this.b = 2;

}

let o = new f(); // {a: 1, b: 2}

// add properties in f function's prototype

f.prototype.b = 3;

f.prototype.c = 4;

// do not set the prototype f.prototype = {b:3,c:4}; this will break the prototype chain

// o.[[Prototype]] has properties b and c.

// o.[[Prototype]].[[Prototype]] is Object.prototype.

// Finally, o.[[Prototype]].[[Prototype]].[[Prototype]] is null.

// This is the end of the prototype chain, as null,

// by definition, has no [[Prototype]].

// Thus, the full prototype chain looks like:

// {a: 1, b: 2} ---> {b: 3, c: 4} ---> Object.prototype ---> null

console.log(o.a); // 1

// Is there an 'a' own property on o? Yes, and its value is 1.

console.log(o.b); // 2

// Is there a 'b' own property on o? Yes, and its value is 2.

// The prototype also has a 'b' property, but it's not visited.

// This is called "property shadowing."

console.log(o.c); // 4

// Is there a 'c' own property on o? No, check its prototype.

// Is there a 'c' own property on o.[[Prototype]]? Yes, its value is 4.

console.log(o.d); // undefined

// Is there a 'd' own property on o? No, check its prototype.

// Is there a 'd' own property on o.[[Prototype]]? No, check its prototype.

// o.[[Prototype]].[[Prototype]] is null, stop searching,

// no property found, return undefined.

**Inheriting Methods**

var o = {

a: 2,

m: function() {

return this.a + 1;

}

};

console.log(o.m()); // 3

// When calling o.m in this case, 'this' refers to o

var p = Object.create(o);

// p is an object that inherits from o

p.a = 4; // creates a property 'a' on p

console.log(p.m()); // 5

// when p.m is called, 'this' refers to p.

// So when p inherits the function m of o,

// 'this.a' means p.a, the property 'a' of p