//Example:

function Shape(pos) {

if (pos!=undefined) this.pos=pos;

else pos=[0,0];

}

Shape.prototype.getPos= function(){

return this.pos;

}

function Rectangle(pos,len, wid) {

this.pos=pos;

this.len=len;

this.wid=wid;

}

// Let Rectangle inherit Shape

Rectangle.prototype=new Shape();

Rectangle.prototype.constructor=Rectangle;

Rectangle.prototype.getArea= function(){

return this.len\*this.wid;

}

function Cuboid(pos,len, wid, hei) {

this.pos=pos;

this.len=len;

this.wid=wid;

this.hei=hei;

}

// Let Cuboid inherit Rectangle

Cuboid.prototype=new Rectangle();

Cuboid.prototype.constructor=Cuboid;

Cuboid.prototype.getVolume= function(){

return this.getArea()\*this.hei;

}

var cuboid= new Cuboid([4,3],10,5,7);

console.log("The cuboid posistion is: "+cuboid.getPos());

console.log("The cuboid total area is: "+cuboid.getArea());

console.log("The cuboid volume is: "+cuboid.getVolume());

var rectangle= new Rectangle([41,23],110,25);

console.log("The rectangle posistion is: "+rectangle.getPos());

console.log("The rectangle total area is: "+rectangle.getArea());

//Example2:

/\*

We can rewrite this to include encapsulation technologies, which means to create private member

variables in objects and use the accessors to modify or retrieve them.\*/

function Shape(pos) {

this.constructor.prototype.getPos= function(){ return pos; }

this.constructor.prototype.setPos= function(p){ return pos=p; }

}

function Rectangle(pos, len, wid) {

this.setPos(pos);

this.constructor.prototype.getLen= function(){ return len; }

this.constructor.prototype.getWid= function(){ return wid; }

this.constructor.prototype.setLen= function(l){ len=l; }

this.constructor.prototype.setWid= function(w){ wid=w; }

this.constructor.prototype.getArea=function(){ return wid\*len;}

}

function Cuboid(pos,len, wid, hei) {

this.setPos(pos);

this.setLen(len);

this.setWid(wid);

this.constructor.prototype.getHei= function(){ return hei; }

this.constructor.prototype.getVolume=function(){

return this.getArea()\*hei;

}

}

// create Inheritance before creating any object

Rectangle.prototype= new Shape();

Rectangle.prototype.constructor=Rectangle;

Cuboid.prototype= new Rectangle();

Cuboid.prototype.constructor=Cuboid;

var cuboid= new Cuboid([4,3],10,5,7);

console.log("The cuboid posistion is: "+cuboid.getPos());

console.log("The cuboid total area is: "+cuboid.getArea());

console.log("The cuboid volume is: "+cuboid.getVolume());

var rectangle= new Rectangle([41,23],110,25);

console.log("The rectangle posistion is: "+rectangle.getPos());

console.log("The rectangle total area is: "+rectangle.getArea());

//Example 3:

/\*

It is important to create the inheritance before you create the objects as a constructor function uses the prototype object during the initiation of the new object.

This means that you can not create any inheritance in constuctor function like this:

this.constructor.prototype=new Shape(); --- NOT POSSIBLE WRONG WAY

this.constructor.prototype.constructor=Rectangle; --- NOT POSSIBLE WRONG WAY

In such a chain of inheritance between objects, an object created by a function object is not

only an instance of this, but also an instance of the function objects, which was used to

create the inherited objects.

Javascript has a 'instanceof' operator we can use to test such things.

If you insert, the below script, at end of the above script you will see that an instance

of a Cupoid is also an instance of a Rectangle and a Shape as well.

==================================================================

document.write("Is cuboid instanceof Shape - returns: "+

(cuboid instanceof Shape) +"<br>");

document.write("Is cuboid instanceof Rectangle - returns: "+

(cuboid instanceof Rectangle) +"<br>");

\*/

//============================================================================================

//Example 4: Create an inheritance through copying.

/\* Another way to create inheritance between objects is to copy the properties from the object

you want to inherit from.

The properties that are copied will be added to the prototype object to the function object.

The advantage here is that you do not need to recreate the constructor property of the

prototype objects again, and you can do the copy function inside the constructor function.

The disadvantage of this method is that objects are not an instance of the object it

inherits.\*/

function copyProp(toObj, fromObj) {

for (obj in fromObj) {

toObj[obj]=fromObj[obj];

}

}

function Shape(pos) {

this.constructor.prototype.getPos= function(){ return pos; }

this.constructor.prototype.setPos= function(p){ return pos=p; }

}

function Rectangle(pos, len, wid) {

copyProp(this.constructor.prototype, new Shape(pos));

this.constructor.prototype.getLen= function(){ return len; }

this.constructor.prototype.getWid= function(){ return wid; }

this.constructor.prototype.getArea=function(){ return wid\*len;}

}

function Cuboid(pos,len, wid, hei) {

copyProp(this.constructor.prototype, new Rectangle(pos,len, wid));

this.constructor.prototype.getHei= function(){ return hei; }

this.constructor.prototype.getVolume=function(){

return this.getArea()\*hei;

}

}

var cuboid= new Cuboid([4,3],10,5,7);

console.log("The cuboid posistion is: "+cuboid.getPos());

console.log("The cuboid total area is: "+cuboid.getArea());

console.log("The cuboid volume is: "+cuboid.getVolume());

var rectangle= new Rectangle([41,23],110,25);

console.log("The rectangle posistion is: "+rectangle.getPos());

console.log("The rectangle total area is: "+rectangle.getArea());

console.log("Is cuboid instanceof Shape - returns: "+

(cuboid instanceof Shape));

console.log("Is cuboid instanceof Rectangle - returns: "+

(cuboid instanceof Rectangle));

Example: