**OOJS EXERCISE**

**Q1. Create a hierarchy of person, employee and developers.**

Solution :

function Person(fName, lName) {

this.firstName = fName;

this.lastName = lName;

}

function Employee(fName, lName, eId) {

Person.call(this, fName, lName);

this.empId = eId;

}

Employee.prototype = Object.create(Person.prototype);

Employee.prototype.constructor = Employee;

// Employee.prototype.getEmpInfo = function () {

// return [this.empId, this.firstName, this.lastName];

// }

function Developer(fName,lName,eId,profile)

{

Employee.call(this,fName,lName,eId);

this.profile = profile;

}

Developer.prototype = Object.create(Employee.prototype);

Developer.constructor = Developer;

Developer.prototype.getEmpInfo = function() {

return [this.empId, this.firstName, this.lastName, this.profile];

}

var e1 = new Developer('Mudit', 'Garg', 10 , 'Trainee');

e1.getEmpInfo();

OUTPUT : [10, "Mudit", "Garg", "Trainee"]

**Q2. Given an array, say [1,2,3,4,5]. Print each element of an array after 3 secs.**

Solution :

var a = [1,2,3,4,5];

var i = 0;

function print()

{

if(i<a.length)

{

console.log(a[i]);

}

i++;

setTimeout(print, 3000);

}

print();

**Q3. Explain difference between Bind and Call (example).**

Solution :

Bind() :

It is used to fix the value of this to a specific object (called the context object).

It attaches ‘this’ to the function, but it needs to be called separately.

Call() :

Function.call takes a context parameter as the first argument: anything passed here is accessible using this inside the function body. It invokes the function immediately, no need to invoke separately as in bind.

Bind example :

var person = {

hello: function() {

console.log(this.name + " says hello ");

}

}

var p = {name: "Mudit Garg"};

var p2 = person.hello.bind(p);

p2();

Call() example :

var person = {

hello: function() {

console.log(this.name + " says hello ");

}

}

var p = {name: "Mudit Garg"};

var p2 = person.hello.call(p);

**Q4. Explain 3 properties of argument object.**

Solution :

Argument object is an in-built object which is used when number of parameters being thrown to a function are unknown;

This object can access any number of arguments like an array.

example : to find max string among parameters thrown to a function

function maxstring()

{

var max = " " ;

for( var i = 0 ; i < arguments.length ; i++)

{

if(max.length < arguments[i].length)

{

max = arguments[i];

}

}

console.log(max);

}

maxstring("one","two","three");

3 properties of arguments object :

arguments.callee : It refers to the currently executing function. In following example, arguments.callee in console.log will print the function body of executing function itself;

function maxstring()

{

var max = " " ;

for( var i = 0 ; i < arguments.length ; i++)

{

if(max.length < arguments[i].length)

{

max = arguments[i];

}

}

console.log(max);

console.log(arguments.callee);

}

maxstring("one","two","three");

arguments.caller : Refers to the function that called currently executing function. In following example, the function is called from window.

function maxstring()

{

var max = " " ;

for( var i = 0 ; i < arguments.length ; i++)

{

if(max.length < arguments[i].length)

{

max = arguments[i];

}

}

console.log(max);

if (arguments.caller == null)

console.log('I was called from the global scope.');

else

console.log(arguments.caller + ' called me!');

}

maxstring("one","two","three");

**Q4. Create a function which returns number of invocations and number of instances of a function.**

Solution :

var objcounter = 0;

var invcounter = 0;

function foo()

{

if(this==window)

{

invcounter++;

}

else{

objcounter++;

}

}

var obj1 = new foo();

var obj2= new foo();

var obj3 = new foo();

foo();

foo();

console.log(objcounter,invcounter);

**Q6. Create a counter using closures.**

Solution :

var c = 0;

function outer()

{

var inner = function()

{

++c;

return c;

}

return inner;

}

for(i = 0; i < 10 ; i++)

{

var a = outer();  
 a();

}