Introduction to Object Oriented JavaScript

Inheritance

Capgemini

Lesson Objectives

Prototype paradigm

Prototypal inheritance

Prototypal inheritance using __proto__

Prototypal inheritance using create()

Prototypal inheritance using prototype

Prototype paradigm

Prototype-based programming is a style of object-oriented programming in which behavior reuse is performed via a process of reusing existing objects via delegation that serve as prototypes.

Prototype object oriented programming uses generalized objects, which can then be cloned and extended.

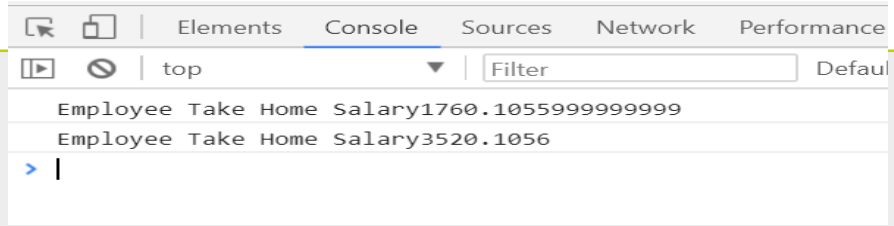
Prototype paradigm makes use of an object’s prototype property, which is considered to be the prototype upon which new objects of that type are created.

In Prototype , an empty constructor is used only to set up the name of the class.

All properties and methods are assigned directly to the prototype property.

Why-Prototype

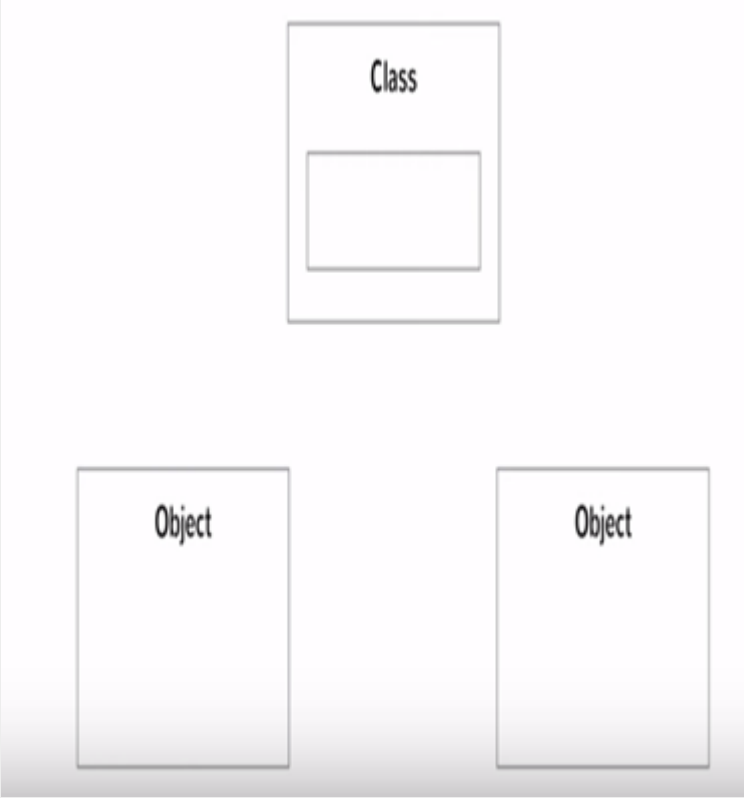
```
function createEmployee(empId,empName,empSalary,empDep){
    this.empId=empId;
    this.empName=empName;
    this.empSalary=empSalary;
    this.empDep=empDep;
    this.totalSalary;
    this.getTakeHomeSalary=function(){
        this.totalSalary=this.empSalary-(this.empSalary*0.12);
        console.log("Employee Take Home Salary"+this.totalSalary)
    }
}
var empone=new createEmployee(1001,'Rahul',2000.12,'JAVA');
empone.getTakeHomeSalary();
var empTwo=new createEmployee(1002,'vikash',4000.12,'.Net');
empTwo.getTakeHomeSalary();
```



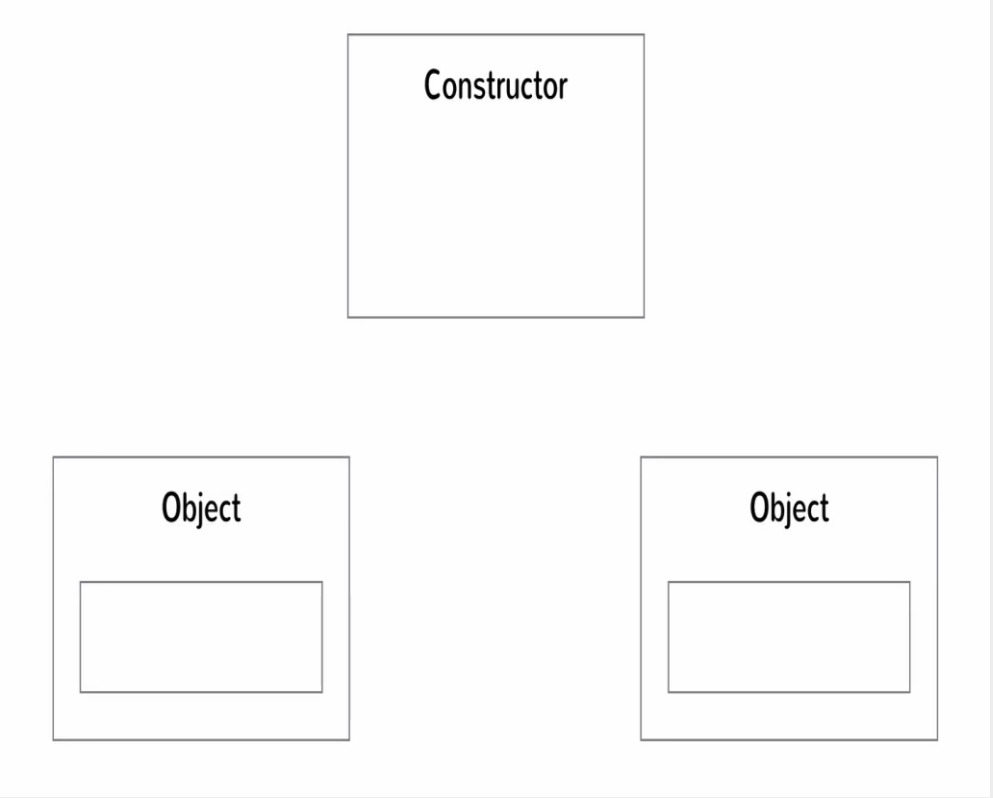
Why-Prototype

In Javascript no concept of classes

Only one copy
in class & object



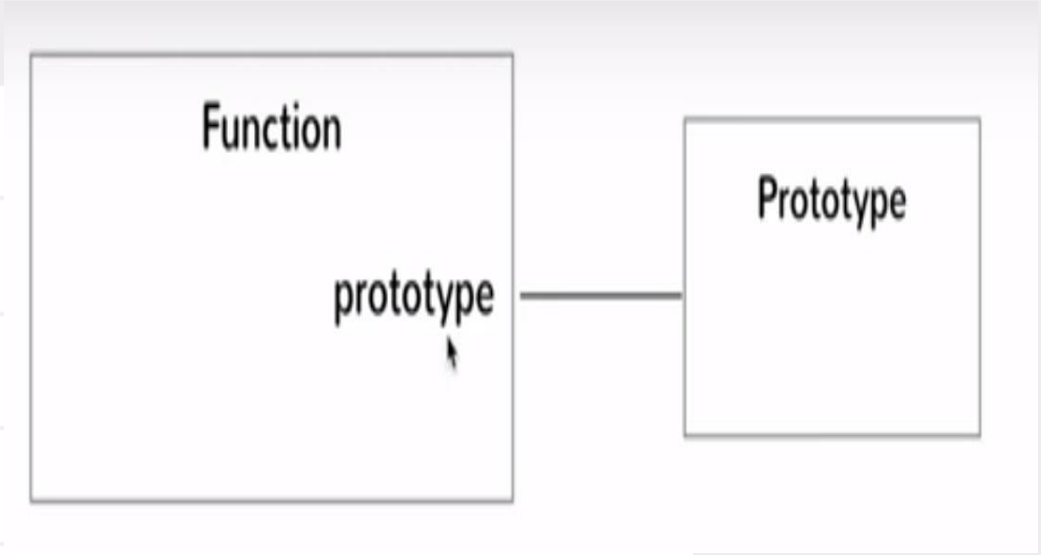
Each Object has own copy
in constructor & object



Prototype paradigm

Each Javascript function create 2 Object
  function object
  prototype object

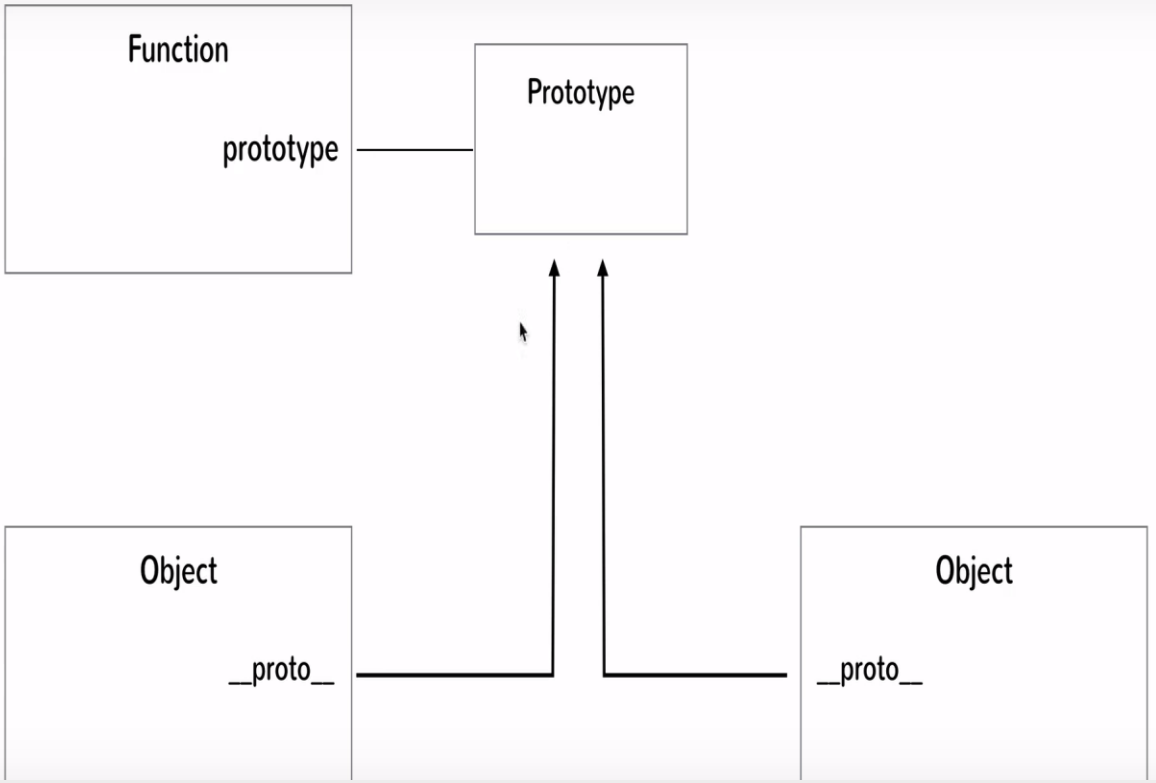
```
> function foo(){}
<· undefined
> function bar(){}
<· undefined
> foo
<· ƒ foo(){}
> bar
<· ƒ bar(){}
> foo.prototype
<· ▶ {constructor: ƒ}
> bar.prototype
<· ▶ {constructor: ƒ}
>
```



Prototype paradigm

Now the any Objects will refer to _proto not function

```
> function foo(){}
<· undefined
> var myObj=new foo();
<· undefined
> myObj
<· ▼ foo {} i
    ▼ __proto__:
      ▶ constructor: ƒ foo()
      ▶ __proto__: Object
>
```



Prototype paradigm

Now Check the two-- by help of _proto_

```
> function foo(){}
< undefined
> foo();
< undefined
> foo.prototype.test="this is Prototype"
< "this is Prototype"
> var newObj=new foo();
< undefined
> newObj.__proto__.test
< "this is Prototype"
> newObj.__proto__.test===foo.prototype.test
< true
>
```

Prototype paradigm

Prototype Example

```
function Employee(empId,empName,empSalary,empDep){
      this.empId=empId;
      this.empName=empName;
      this.empSalary=empSalary;
      this.empDep=empDep;
      this.totalSalary;
      Employee.prototype.getTakeHomeSalary=function(){
            this.totalSalary=this.empSalary-(this.empSalary*0.12);
            console.log("Employee Take Home Salary"+this.totalSalary)
      }
}
Employee.prototype.greet=function(){
console.log("WELCOME to PROTOTYPE");}
var emp=new Employee(1001,"Abcd",8888,"Java");
emp.getTakeHomeSalary();
var empOne=new Employee(1002,"bcd",98888,".Net");
empOne.getTakeHomeSalary();
empOne.greet();
```

Inheritance-What in Javascript

JavaScript is a **prototype-based language**, meaning object properties and methods can be shared through generalized objects that have the ability to be cloned and extended.

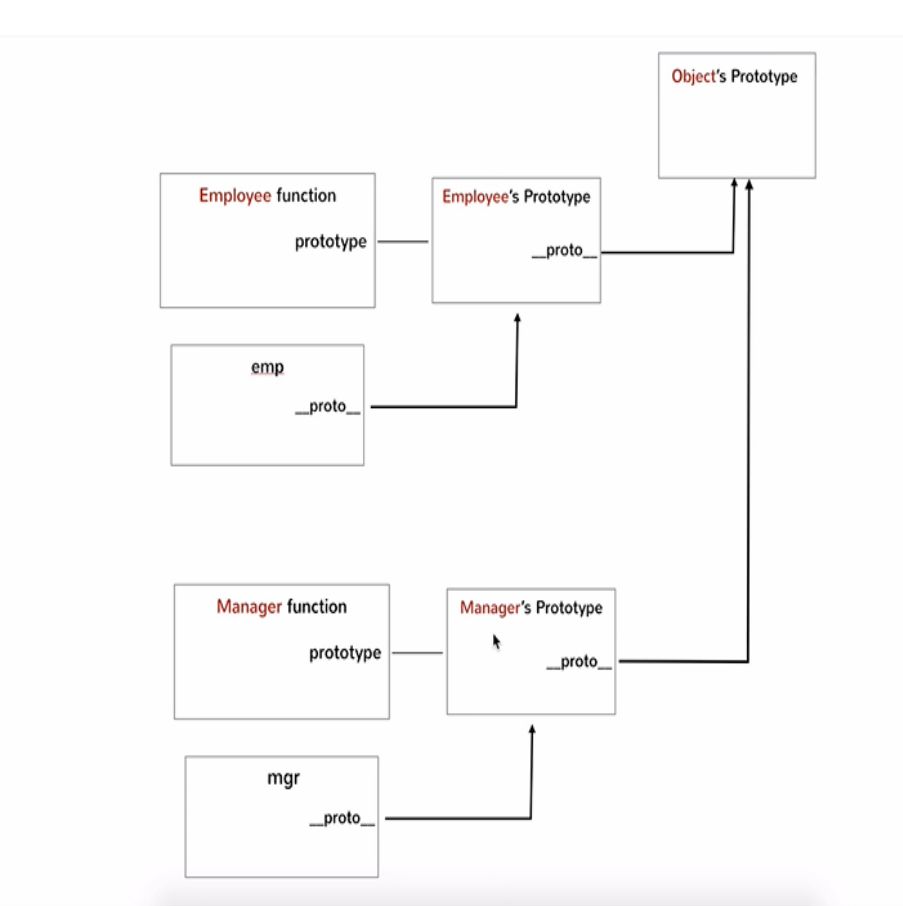
We can do inheritance by

Inheritance –By Using _proto_

By Using Object.create()

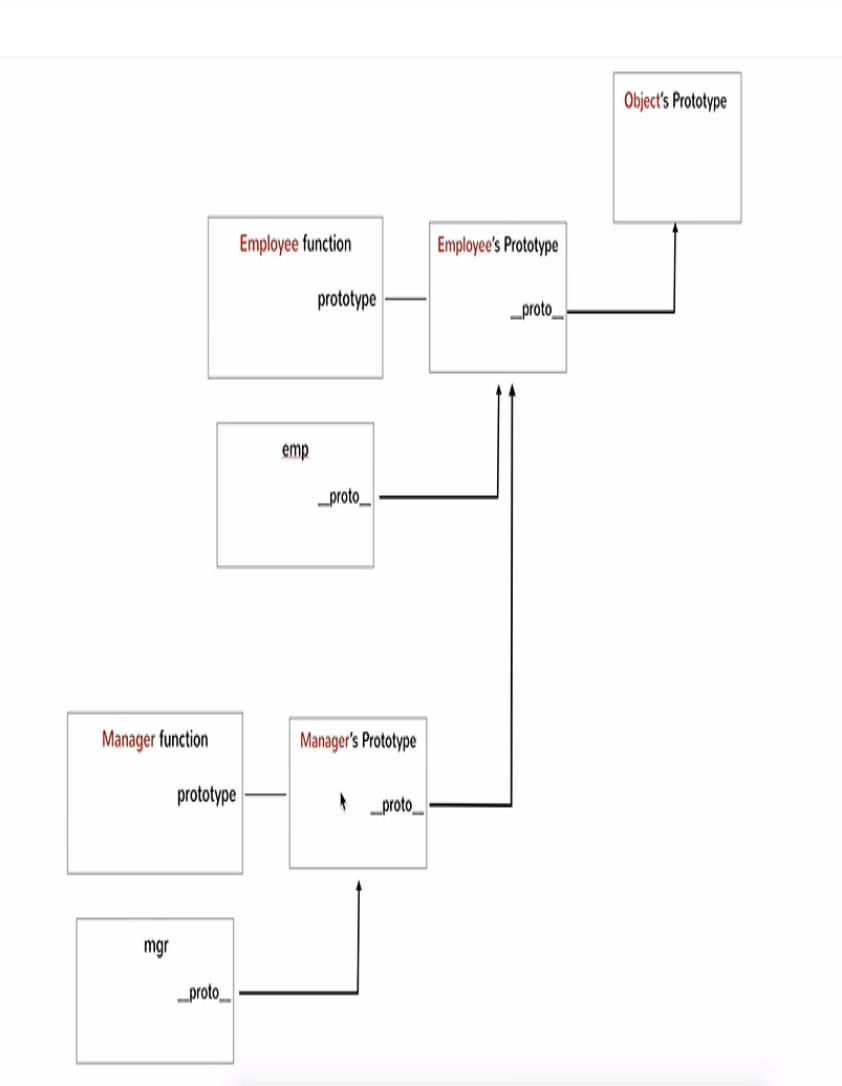
Inheritance -Why

According to the diagram we create 2 function & try to access other member such as dep want to access “name” member of employee




```
function Employee(name){
        this.name=name;
}

Employee.prototype.getName=function(){
        return this.name
}

function Department(name,manager){
        this.name=name;
        this.manager=manager;
}

Department.prototype.getDepName=
        function(){
                return this.name
        }
var emp=new Employee("Abcd");
var dep=new Department("sales","BCDE");
console.log(emp.getName());
console.log(dep.getDepName());

//but if we want to aceess dep.getName()
```

Inheritance –By Using _proto_




```
function Employee(name){
        this.name=name;
}

Employee.prototype.getName=
        function(){
                return this.name
        }

function Department(name,manager){
        this.name=name;
        this.manager=manager;
}

Department.prototype.getDepName=
        function(){
                return this.name
}

var emp=new Employee("Abcd");
var dep=new Department("sales","BCDE");
console.log(emp.getName());
console.log(dep.getDepName());

//but if we want to aceess dep.getName()

dep._proto_=emp;  //dep inherit from emp
console.log(dep._proto_.getName());
```

Inheritance –By Using Object.create()

```
function Employee(name){
        this.name=name;
}

Employee.prototype.getName=function(){return this.name}

function Department(name,manager){
        this.name=name;
        this.manager=manager;
}

Department.prototype.getDepName=function(){
                        return this.name
                }

var emp=new Employee("Bcd");
var dep=Object.create(emp);
console.log(dep.getName());
```

Inheritance –By using prototype

```
function Employee(name){
            this.name=name;
}

Employee.prototype.getName=function(){
                                    return this.name
                        }

function Department(manager){

            this.manager=manager;
}

Department.prototype.getMagagerName=function(){
                        return this.manager
            }

Department.prototype=new Employee("CDE");

var dep=new Department();

console.log(dep.getName());
```

Demo

Demo1
Demo2
Demo3
Demo4

Lab

Lab 2

Summary

In this lesson we have learned about -

Prototype paradigm

Prototypal inheritance

Prototypal inheritance using __proto__

Prototypal inheritance using create()

Prototypal inheritance using prototype