PAGE NO:
DATE: / /

Javascript Objects and Prototypes in-depth. (related to Objects but not class based so diff from other language)

Unit 01:-
Creating Objects

Unit 02:-
Function execution and the this reference

Unit 03:-
Prototypes (important, like a template.
Note:- JS don't have class there are tools/schtaut that build help something that resemble class But no. core class as in language of Java

Note:-

The ES6 version has a class keyword that give class like Syntax. But you ~~dg~~ still don't get all the typical class features. Its Syntax Sugar.

PAGE NO :
DATE : / /

Object Basics

↳ You can variable with primitive like number, string, Boolean or something call bundle/ multiple values called Object
(collection)
↓
can be further obj/ primitive / function / array

↳ An Object can be created in multiple way simplest being inline

```
var myObj = {};
Console.log (myObj);
myObj.foo = "value";
Console.log (myObj.foo); //"value"
myObj.foo = 100;
```

PAGE NO :
DATE : / /

↳ Other way (inline With Value Set)

```
Var myObj = {
    "foo" : "value",
    "age" : 30,
    "address" : {
        "street" : "123 JS",
        "city" : "JS"
    },
    "Pincode" : 12345.
};
```

↳ Can access using [] or . (we dissuss in Other playlist)

```
myObj.foo
myObj["foo"]
```

PAGE NO :
DATE : / /

Creating Objects

```
Var empl = {};
 Empl.firstName = "Michael";
 Empl.lastName = "Scott";
 Empl.gender = "M";
Empl.designation = "Regional Manager";
```

```
Var emp2 = {};
emp2.firstName = "Dwight";
Emp2.lastName = "Schrute";
emp2.gender = "M";
emp2.designation = "Assistant to the regional
                    Manager";
```

What if we have 100 employee? this will become tedious.

↳ What we want is write Once use mult time providing useful information

PAGE NO :
DATE : / /

```
function CreateEmployeeObject(){
    var newObject = {};
    newObject.firstName = "Mikael";
    newObject.lastName = "Scott";
    newObject.gender = "M";
    newObject.designation = "Regional Manager";
    return newObject;
};
```

↑ Here value are hardcoded, everytime called newObject will be created with same hard coded value.

```
function CreateEmployeeObject(firstname,
                              lastName, gender, designation)
{
    var newObject = {};
    newObject.firstName = firstName;
    newObject.lastName = lastName;
    newObject.gender = gender;
    newObject.designation = designation;
};  return newObject;
```

PAGE NO
DATE : / /

Var emp3 = CreateEmployeeObject ("Jim", "Halpert", "M", "Sales Representative");

Constructor Function

function CreateEmployeeObject (——) {

Var newObject = {};

Common When Creating Obj

return newObject;

}

this is common, JS Created Shortcut to Skip this Code

Var emp3 = new CreateEmpObjFunc (——);

↓

I know you want to Create new Employee Object Just Code rest part

PAGE NO :
DATE : / /

```
function CreateEmpObj(firstName, lastName,
                      gender, designation) {
    // Create new obj, you can access it as this
    // Varg this = {};
    this.firstName = firstName;
    this.lastName = lastName;
    this.gender = gender;
    this.designation = designation;
    //return this
}
```

done under the hood.

```
var emp3 = new CreateEmpObj("Jim", "Halpert",
                            "M", "Sales repre");
```

PAGE NO
DATE : / /

Difference between regular function and Constructor.

inline

```
Var bicycle = {
    "cadence" : 50,
    "speed" : 20,
    "gear" : 4
};
```

```
function CreateBicycle (cadence, speed, gear) {
    Var newBicycle = {};
    newBicycle.cadence = cadence;
    newBicycle.speed = speed;
    newBicycle.gear = gear;
    return newBicycle;
}

Var bicycle1 = CreateBicycle (50, 20, 4);
Var bicycle2 = CreateBicycle (20, 5, 1);
```

PAGE NO:
DATE: / /

```
function bicycleConstructor(cadence, speed, gear){
    // var this = {};   ← not actually but all practical purposes use can assume
    this.cadence = cadence;
    this.speed = speed;
    this.gear = gear;

    // return this;
}

var bicycle3 = new bicycleConstructor(50, 20, 4);
```

Note:- There is nothing about the function itself that say it should be called in constructor mode. Its not prop of func but way you call it.

```
var bicycle3 = bicycleConstructor(50, 20, 4);
```

Here it will create number globally in regular mode

Convention:- To use regular case, rather than camel case. ✓ function Bicycle(---){}

bicycleConstructor ✗ not good practice

You are basically calling function in JS to create Object.

no one is stopping you to call a function in either mode. (regular or constructor mode).

PAGE NO :
DATE : / /

Switching function types and calls

```
funct CreateBicycle (candence, speed, gear){
    Var newBicycle = {};
    ========
    return newBicycle;
}

Function Bicycle (candence, speed, gear){
    [redacted]
    this.cadence = cadrce;
    this.gear = gear;
    this.speed = speed;
    [redacted]
}
```

✓ Var bicycle1 = new CreateBicycle (50,20,4);

↳ add | var this = {}; ← start
| return newBicycle;
| return this; ← end

✓ Var bicycle2 = CreateBicycle (50,20,4);

✓ `var bicycle3 = new Bicycle(50, 20, 4);`

✗ `var bicycle4 = Bicycle(50, 20, 4);` → code this refers to global (window object)

↑ undefined

add member speed, cadence, gear to window object

Note Calling a constructor function without the new keyword doesn't work! No new Object gets created, and return value is undefined.

Note (Conclusion)

Never mix them, use right mode.

PAGE NO :
DATE : / /

Function Execution Types.

1. Regular
2. Constructor mode
3. In Context of Object
4. ? (Call/bind) etc

(window object)

```
//method 1
function foo() {
    console.log("Hello");
}
foo();

//method 2   [there is diff in (1) and (2) will be clear ahead i.e this]
var obj = {};
obj.foo = function () {
    console.log("Hello");
};
obj.foo(); //[In Context of Object]
```

PAGE NO
DATE : / /

// method 3 (constructor)

new foo();

// method 4. ← Will discuss it later

you need to understand about Execution Context and this Reference

PAGE NO:
DATE: / /

The this argument Values

When a function is called its called in particular context. Js also has execution context. Context depend on Way function is called.

One Variable of this execution Context is 'this'.

There are two default arguments to every function call: arguments and this

Once the function call is identified the value of this is very predictable.

1.

```
function foo() {
    console.log("Hello");
    console.log(this);
}

foo();
// Hello
// (Window Obj)
```

Method 1: Calling standalone functions directly

this reference: The global Object

PAGE NO.
DATE: / /

2.

```
var Obj = {};
Obj.prop = "This is the Object itself!";
Obj.foo = function () {
    Console.log("Hello");
    Console.log(this);
}
obj.foo();
```

// Hello
// Obj Object

```
{"prop": "This is the Object itself!",
 "foo": Obj.foo function()
}
```

Method 2: Calling function as property of an Object reference

this reference: The Object reference

3.

function foo() {
  C.L("Hello"); C.L(this);
}

3. new foo(); # Hello
empty new Obj.

Method 3: Calling Standalone functions using "new" Keyword

this reference — The newly Created Object

—x—

Working on Objects with this Reference

function Bicycle(cadence, speed, gear, tirePressure
{
  this.cadence = cadence;
  this.speed = speed;
  this.gear = gear;
  this.tirepressure = tirepressure;
  this.inflateTires = function() {
    this.tirepressure += 3;
  }
}

this due to new

this due to way its called.

another obj ● can be diff. ● Watch video end.

var bicycle1 = new Bicycle(50, 20, 4, 25);
bicycle1.inflateTires();

based on way call we know execution context

Console.log(bicycle1.tirepressure);

PAGE NO:
DATE: / /

Works for multiple Object as well

```
Var bicycle2 = new Bicycle(50, 20, 4, 30);
bicycle2.inflateTires();
C.L(bicycle2.tirePressure)//33
```

What if I want to give this funct to some other Obj BicycleMechanic, given a bicycle.

Problem

Scope chain based on declaration
Execution Context (this) based on call.

PAGE NO :
DATE : / /

Using the call function

```
function Bicycle (cadence, speed, gear, tirePressure){
    this.cadence = cadence;
    this.speed = speed;
    this.gear = gear;
    this.tirePressure = tirePressure;
    this.inflateTires = function (){
        this.tirePressure += 3;
    }
}
var bicycle1 = new Bicycle(50,20,4,25);
bicycle1.inflateTires(); // 25 → 28
function Mechanic (name) {
    this.name = name;
}
var mike = new Mechanic("mike");
mike.inflateTires = bicycle1.inflateTires;
mike.inflateTires(); // its call in context of mike obj.
                     // this.tirepressure would be undefined
```

Problem

PAGE NO
DATE : /

How if I can somehow say what this refers to.

#4th way to call function

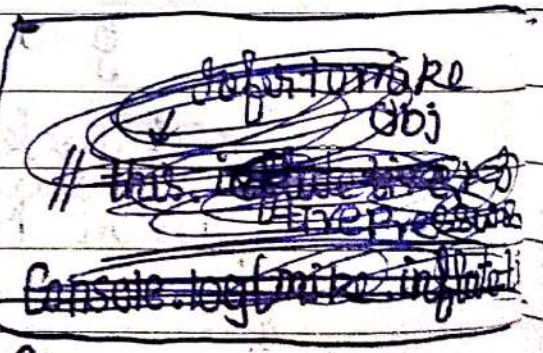
function foo() { this.abc = "def" }

this is bound to Object passed as argument

foo.call({});

you have a choice.

Modifying old Code.



mike.inflateTires();

//this will refer to mike

// this.tirePressure += 3

//Console.log(mike.tirePressure); //NaN

// 28-31

mike.inflateTires.call(bicycle1); //this refers to bicycle1

Console.log(bicycle1.tirePressure);

// 31

PAGE NO:

```
var bicycle2 = new Bicycle (50, 20, 4, 33);
mike.inflateTires.call (bicycle2);
Console.log (bicycle2.tirePressure); 33 → 36
                                    // 36.
```

PAGE NO
DATE : / /

When Constructor's aren't good enough

prototypes is a Concept in JS that lets you build objects using a template or blueprint a lot of languages have a Concept of using blueprint to Create Objects and they use classes. But JS does not have Concept of classes What it does have is the prototype Concept.

(not exactly equivalent to classes)

You Can't Create multiple Obj using prototype but Can Create behaviors that effect multiple objects by using Prototypes.

Will make Sense Slowly.

PAGE NO :
DATE : / /

Javascript Objects don't have "own" methods. They just have properties and any property could be a function

```
function Bicycle (cadence, speed, gear, tirePressure){
    this.cadence = cadence;
    this.speed = speed;
    this.gear = gear;
    this.tirePressure = tirePressure;
    this.inflateTires = function (){
        this.tirepressure += 3;
    }
}
```

→ when this will take seep space for each object as its object property

```
var bicycle1 = new Bicycle (50, 20, 4, 25);
```

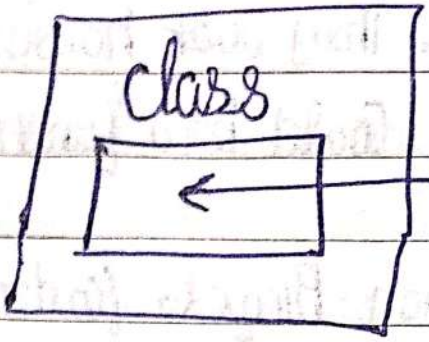
each (underlined) is property of Object and is created each time from scratch whenca new Object of type Bicycle is created.

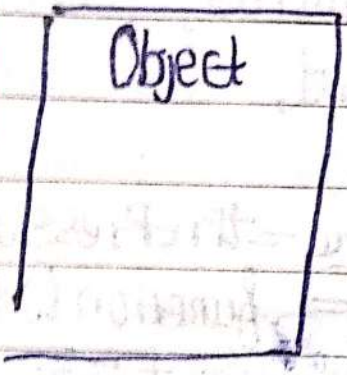
```
var bicycle2 = new Braycle (50, 20, 4, 29);
```

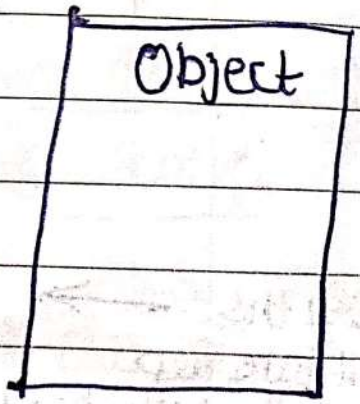
What if we create 10 Object, inflateTires will be prop created for each Object seperately.

PAGE NO
DATE : /

Objects with Classes (other lang)

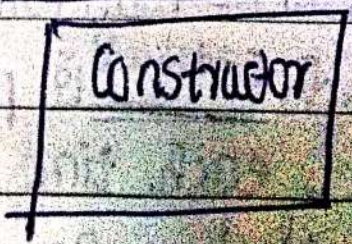


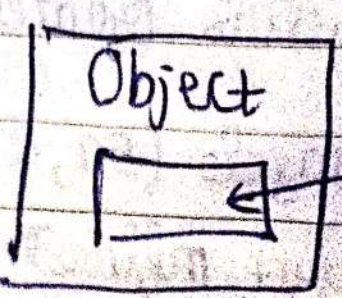


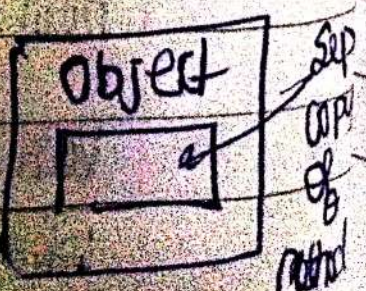


What we have coded now takes lot of mem

Constructor Functions







PAGE NO :
DATE : / /

There should be a efficient way to do it in JS. Way to maintain a single copy and every Object should use it.

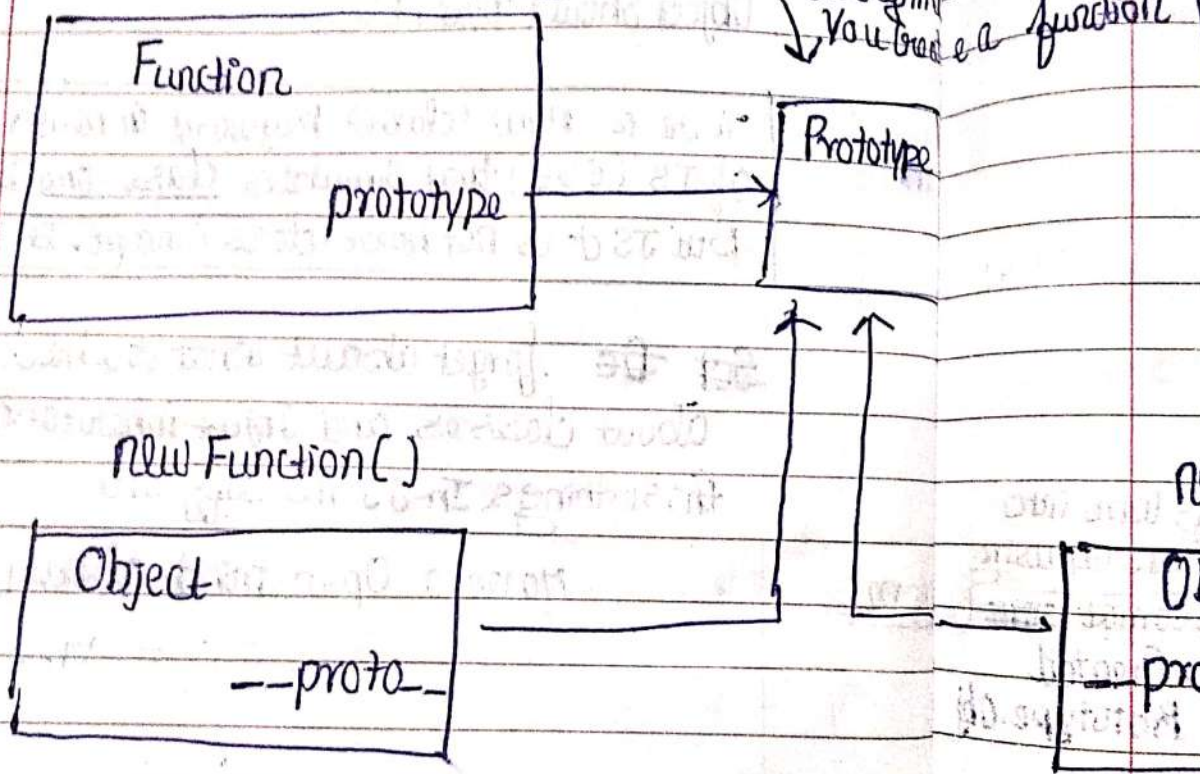
There is new 'class' keyword in newer version of JS (ES6) that simulates class-like behaviours. But JS does not have class concept. It just syntax sugar.

~~Sol~~ ~~Do~~ forget about what you have known about classes and Object inheritance and all those things. In JS its different

Have a Open mind about JS.

PAGE NO
DATE : / /

Introducing the prototype → what happen, next video why? Created everytime you create a function

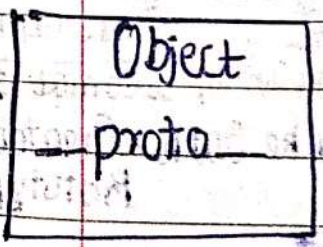


Here — 2 obj Creat
When you
function

function
it self

Here
2 obj

new Function()



PAGE NO :
DATE : / /

Here → >> Function foo() {}

2 Obj Created
When you create
function

>> foo
function foo() {
}

function itself

Prototype object

Here → >> function bar() {}
2 Obj
bar fun — bar Prototype

ew Function()

bject
oto

How to access Prototype of resp fun.

>> foo.prototype
// Object return

>> bar.prototype
// Object return

PAGE NO
DATE : / /

→ This happens for every Function

```
Function greet() {
    Console.log("Hello");
}
```

greet();
undef ← return
//Hello

greet
//ƒ function greet()

greet.prototype
//. Object

☆☆
→ Prototype always come into Play when you Create objusing func else Sit Silently// automat
return i.e greet func ke sath ~ Created Prototype.obj

Why we are talking about it in Creating Obj. Yes it is related Continue with next lines of code

(Just don't look at what written Object())

PAGE NO: Foo()
DATE: when you get its impl
Return from console dependent

~~for~~ foo();
→ undefined.
// nothing executed as empty body

new Foo();
→ object
// now some lines are injected and what is returned is an object.

var myObj = new Foo();

myObj
→ Object { ... __proto__: Object }

~~myObj~~ myObj.__proto__ == foo.Prototype

var myObj2 = new Foo();

myObj2.__proto__ == myObj1.__proto__
// true

PAGE NO
DATE : / /

property lookup with prototypes

Function foo() {}
→ undefined

foo.prototype
→ Object {...}

var newFooObj = new Foo();
→ undefined

newFooObj
→ Object {...}

foo.prototype.test = "This is the prototype Object of foo";
→ "This is the prototype Object of foo"

foo.prototype.test
→ "This is the prototype Object of foo"

PAGE NO:
DATE: / /

newFooObj.__proto__.test

→ "This is the prototype Object of foo"

foo.prototype == newFooObj.__proto__

⇒ true

Know prototype was introduced to something like temp/blueprint to Object. What we want is a central location where we define the behaviours for bunch of similar objects. These Object have behaviour of their own, if not they refer to the central place. They lookup behaviour.

newFooObj.hello

→ undefined.

→ First look at that Object if it has it it give it to you directly

newFooObj.hello = "test"

newFooObj.hello

→ "test"

PAGE NO
DATE : /

delete newFooObj.hello
→ true

newFooObj.hello
→ undefined

→ newFooObj.__proto__
Object{ test: "This is the Prototype Object of foo" }

↑ does not have hello property

→ newFooObj.__proto__.hello = "This value is from prototype"

→ newFooObj.__proto__.hello = "/foo.prototype/ from newFooObj.hello"

all return "This value is from prototype"

✡✡
It does extra step
→ If it does not have a property called hello
It goes to __proto__ and checks there
If found return
else go to its __proto__
until it reaches null
If nothing found returns undefined

PAGE NO:
DATE: / /

newFooObj.test
→ "This is the prototype Object of foo".

→ do newFooObj has member test
Ans. No
↳ then its check at newFooObj.__proto__ do you have test Property
Ans. yes
it returns that

// If I add this property on newFooObj guess what happen ---

newFooObj.test = 10;

// engine will first look for test prop on newFooObj and finds it and not lookup further

newFooObj.test
→ 10

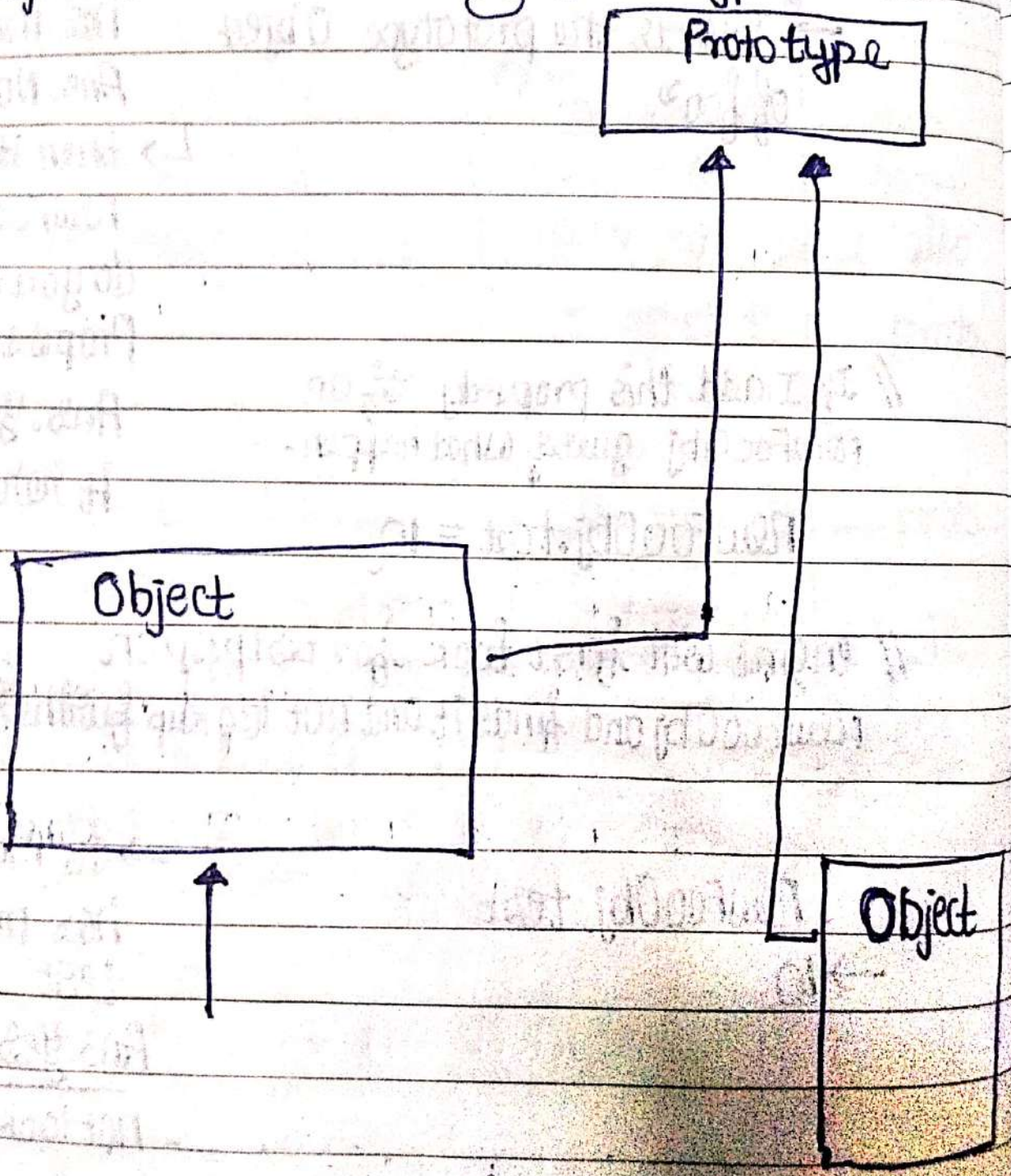
→ do newFooObj has member test
Ans. yes (return)
not lookup at __proto__

This is fancy but what is need of having it?

PAGE NO

DATE : / /

Object behaviours using prototypes



PAGE NO :
DATE : / /

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

var empl = new Employee ("Jim");

→ undef

empl

Object { name: "Jim" .. }

var emp2 = new Employee ("Pam");

emp2

Object { name: "Pam" }

Employee.prototype

Object { , 1 more. }

↙ no sep copy for each emp obj - Thanks to Prototype

Employee.prototype.playPranks = function () {

Console.log ("Prank played!");

};

Employee.prototype.playPranks ();

→ Undefined

Prank played!

PAGE NO
DATE: / /

empl.playPranks(); //due to lookup
→ undefined
Prank played!

emp2.playPranks(); //due to lookup
→ undefined
Prank played!

Only one central storage [Prototype] contain property playPranks, not each employee Object. No matter how many Employee Object, only 1 times its maintained in Prototype.
Emp

Reason why JS has Prototype Concept

Var emp3 = new Employee("Dwight");

emp3.playPranks(); //due to lookup
→ undef
Prank played!

PAGE NO :
DATE : / /

This is dynamic in nature (V/S other language) different
lookup happen at runtime.

emp2.greet() ⟵ !! TypeError:
emp2.greet is
not a function

Employee.prototype.greet = function () {
Console.log ("Hello");
}

emp2.greet();
→ undefined
Hello

emp3.greet();
→ undefined
Hello

PAGE NO :
DATE : / /

Object links With Prototypes

function foo(){}
→ undef

foo.prototype
→ Object { , 1more... }

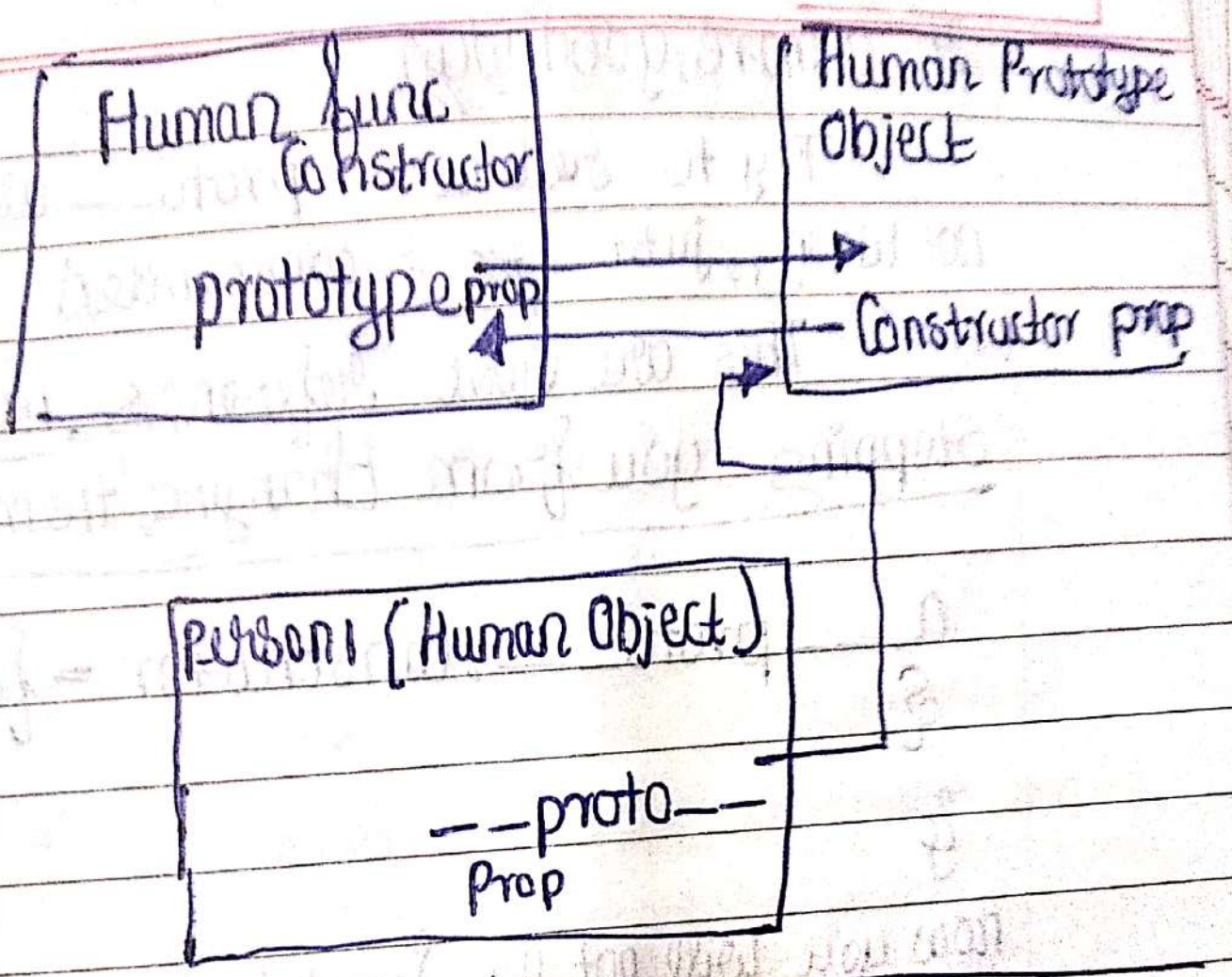
var a = new foo();
→ undefined

a
→ Object { }

The double-underscores are referred to as "dunder" ; So this property is called "dunder Proto"

var proto = foo.prototype;

proto.constructor [not Copy, Just References]
→ function foo()



If you don't know what created a

a.__proto__.constructor
→ function foo()

Var b = new a.__proto__.constructor()
→ undefb
→ b
→ Object { }

PAGE NO
DATE : / /

~~It's a more good way~~

Try to Reduce __proto__ use, no its not wrong, just ≠ Recommended.

This are Just References, no one is Stopping you from changing them

```
a.__proto__.constructor = function bar
{

}
```

now you will not get funct Const that Created Object. Ron anyone can change stuff.

```
Var C = new a.__proto__.constructor()
→undefined
```

```
C
→ Object { }   // Created using bar function Constructor
```

c.__proto__.constructor
→ Function Foo()
← Are same using for
a.__proto__.constructor
→ function Foo()

✡✡ They are not reliable

The Object function

↳ In Java runtime environments just like you have a global window object you also have some global functions and one of them is called Object. name of function is Object but type is function [Object Info].

✡✡✡ Change Perception

↳ Anyway since functions are just objects in JS. this global function which is called Object is actually an Object as well

PAGE NO
DATE : / /

So good title would be

~~global function~~

The Object function Object [one of many global functs]

——— X ———

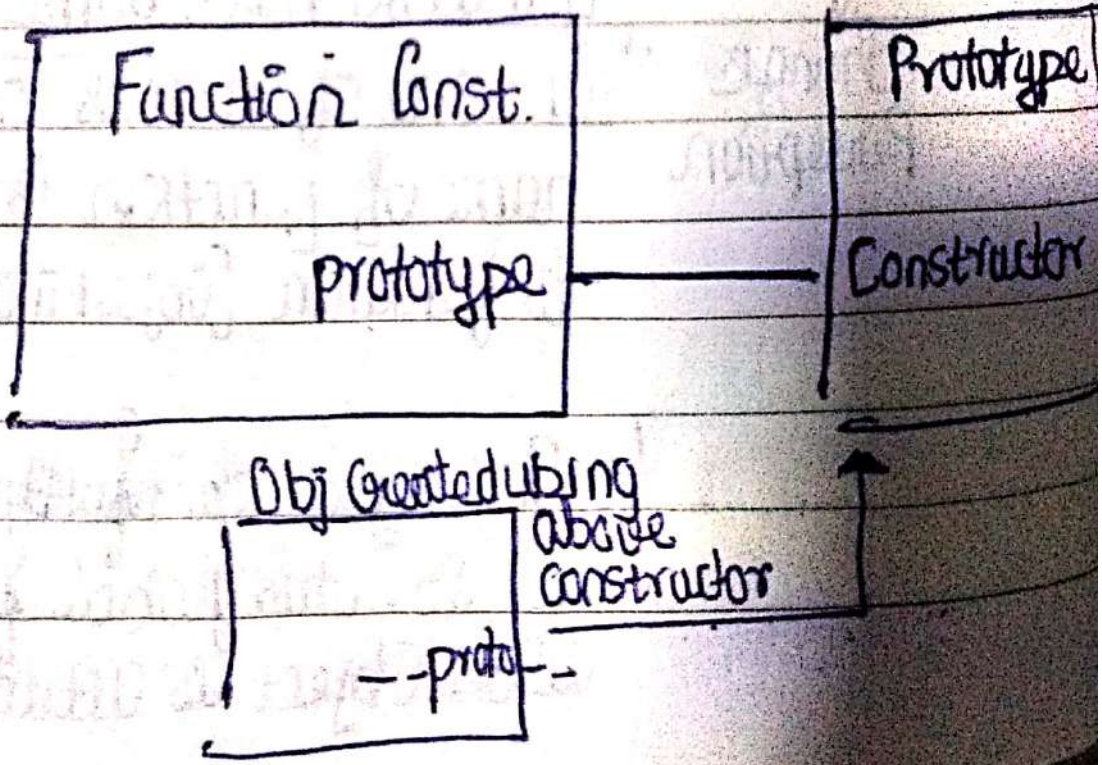
Object ← one of global function

→ function Object ()

Object ()

→ Object & 3

Quick recap



PAGE NO:
DATE: / /

To get the function that constructed this object O.

O.__proto__.constructor

How you create Object in JS.?

```
Var Simple = {};
```

internally they are same

```
Var obj = new Object();
```

another way using global function Object.

Simple
→ Object { }

different Object

obj
→ object { }

obj.__proto__ == Object.prototype → true
Simple.__proto__.constructor
→ function Object()

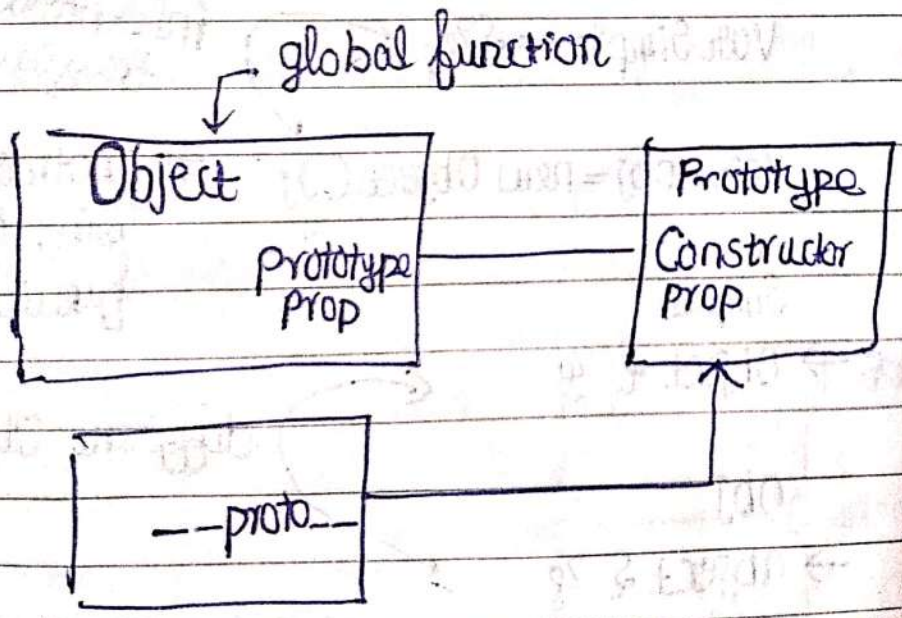
PAGE NO.
DATE: / /

Simple.__proto__ == Obj.__proto__
→ true

Simple.__proto__ = Object.prototype
→ true

var Simple = {} was equivalent to var Simple = new Object();

global function Object



Takeaways 1. Existing of global function Object
2. Whenever you are creating an Object if you are not using specific constructor function you are automatically using new on global Object function

PAGE NO:

DATE: / /

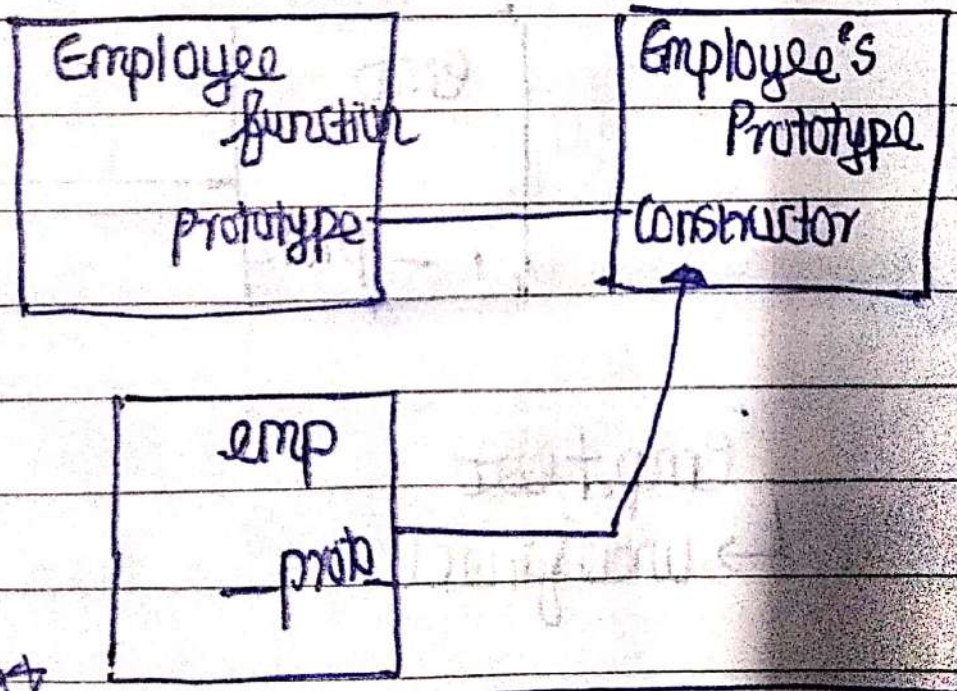
The Prototype Object

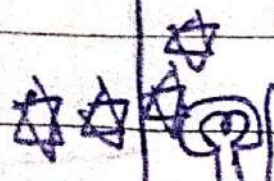
↳ you are nearly close to end

```
function Employee() {}

Var emp = new Employee();
→ undefined
```

bject

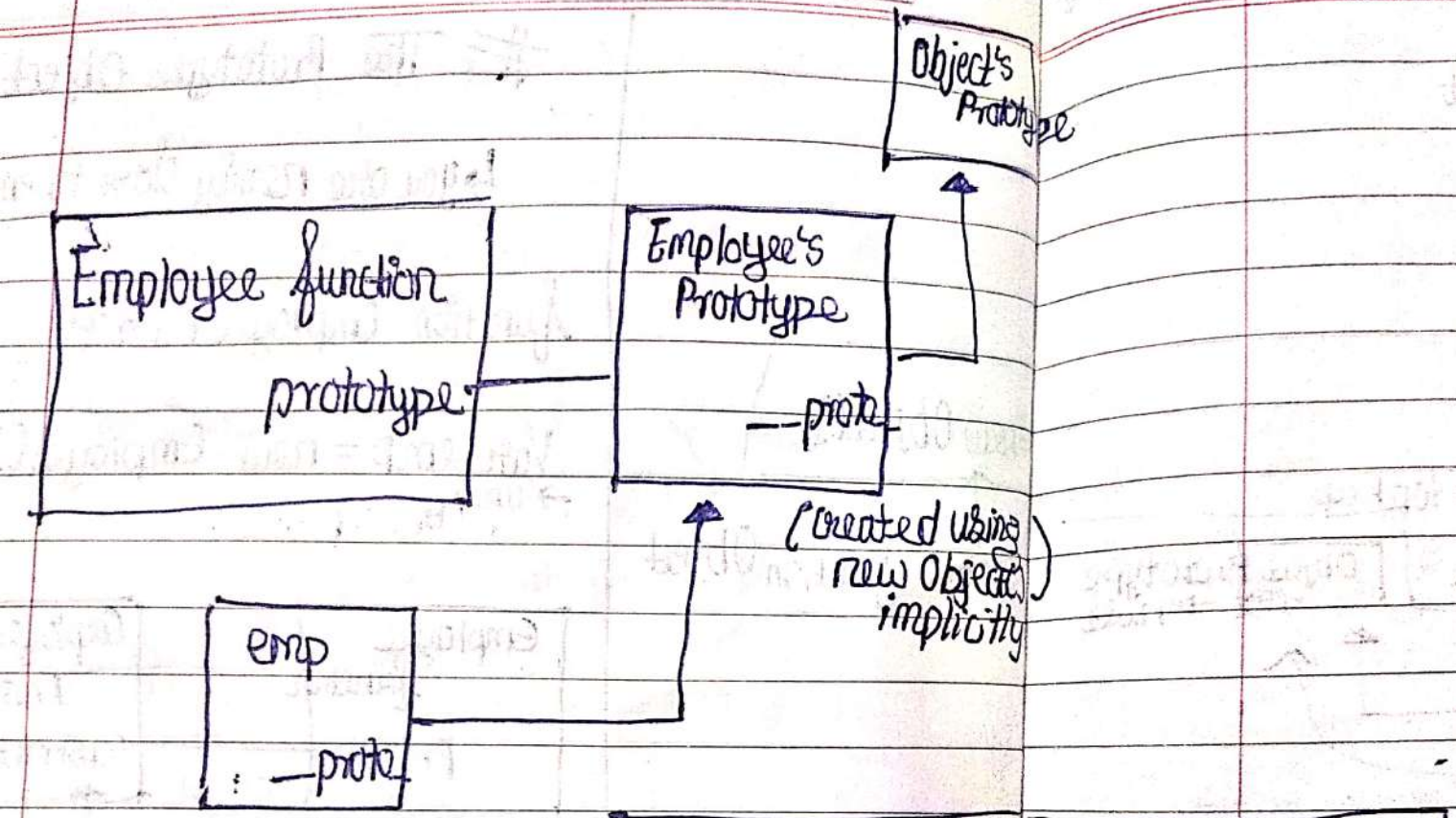




The automatically created prototype object is created using "new Object()".

Complete idea

PAGE NO
DATE : / /



emp.test
↳ undefined

// we look lookup
Check in emp
then Employee Prototype (__proto__)
then Object Prototype (__proto__)

emp.prop = "Employee";

PAGE NO:
DATE: / /

emp.prop
→ "Employee";

emp.__proto__.parentProp =
"parent of Employee";
→ "Parent of Employee"

emp.parentProp // lookup
→ "Parent of Employee"

emp.__proto__.__proto__ == Object.
Prototype
→ true

notice its consequence ahead

Object.prototype.grandparentProp =
"Grandparent of Employee";

emp.grandparentProp // @lookup
→ "Grandparent of Employee";

Object.prototype.grandParentProp = "Grand Parent Of Employee"

↑

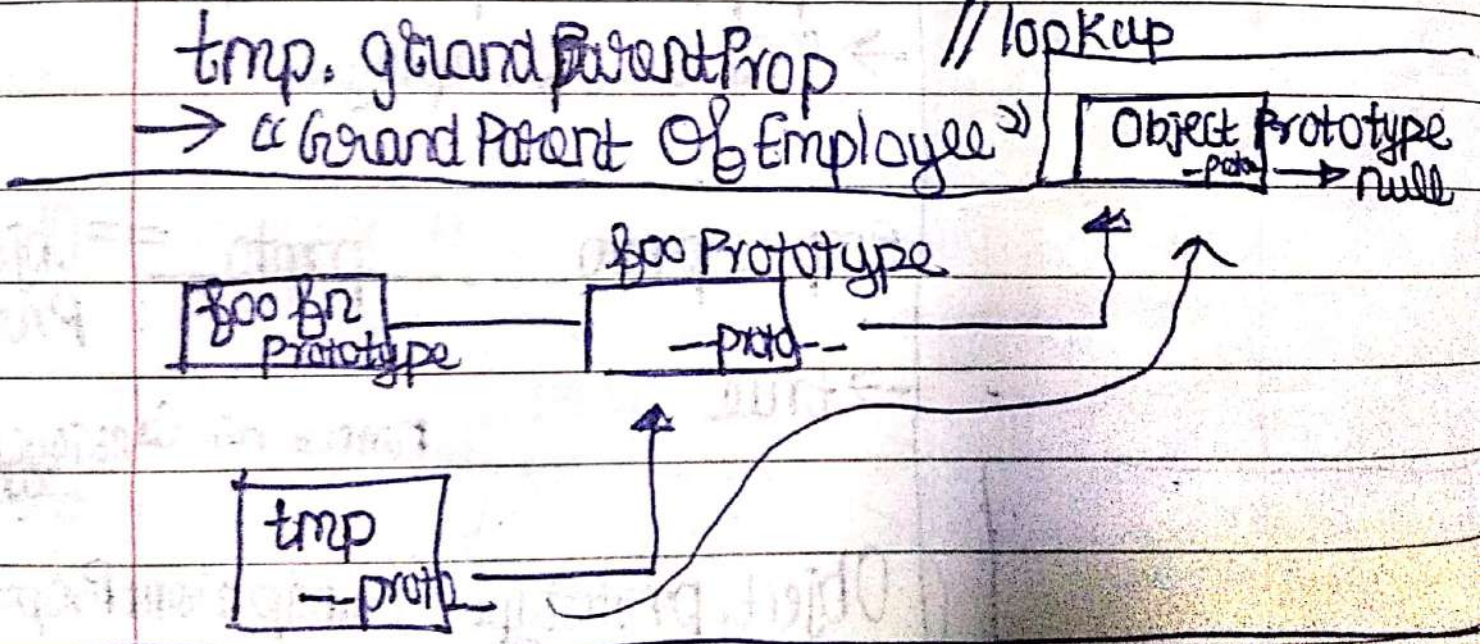
Now every Object will have it

function foo() { }

Var tmp = new foo();

tmp.grandParentProp // lookup

→ "Grand Parent Of Employee"



Indication Of what happen

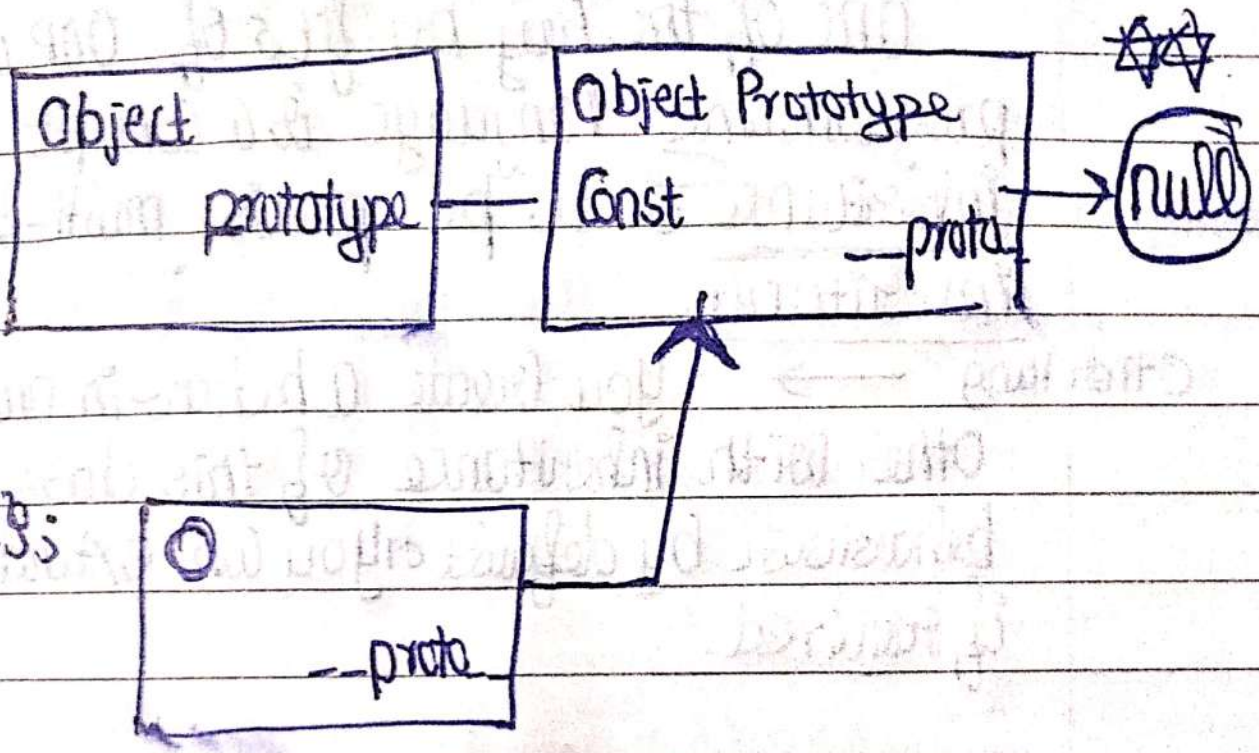
Note:- Object Prototype is also Object What will be __proto__ Of it.

Should lookup be infinite ??

PAGE NO:
DATE: / /

Answer:- No

Object Prototype is where it ends.



Once lookup reaches null you get undefined.

PAGE NO
DATE : / /

Inheritance in Javascript

One of the key benifits of OOP in any programming language is a concept called Inheritance or to be precise multi-level inheritance

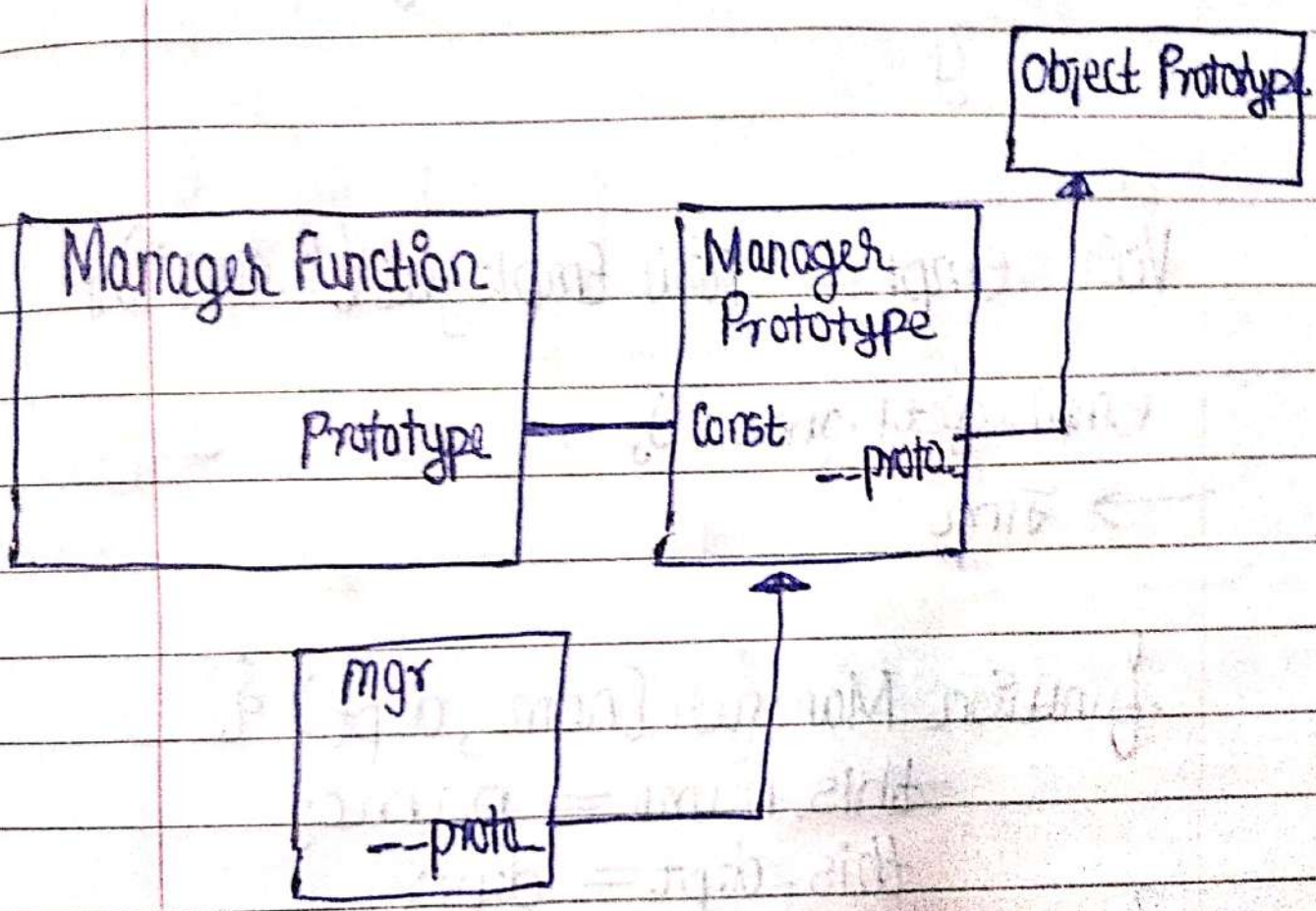
Other lang → You create a behave in one class and other with inheritance of this class get this behaviour by default or you can extend behavior if required.

Prototype help You to achieve something similar in JS.

We already saw its example everything on Object prototype was available to all Object.

PAGE NO :
DATE : / /

We already created Employee, Let Suppose we also want Manager which is also an Employee.



Lets Code this

~~Var~~ function Employee (name) {
this.name = name;
}

PAGE NO
DATE : / /

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

Var empl = new Employee("Jim");
empl.getName();
```

→ Jim

```
function Manager(name, dept) {
        this.name = name;
        this.dept = dept;
}

Manager.prototype.getDept = function()
    return this.dept;
}
```

PAGE NO :
DATE : / /

~~Manager~~

```
Var mgr = new Manager ("Michael", "Sales");
mgr.getDept();
↳ "Sales"
mgr.getName();
```

✗ TypeError: mgr.getName is not a function

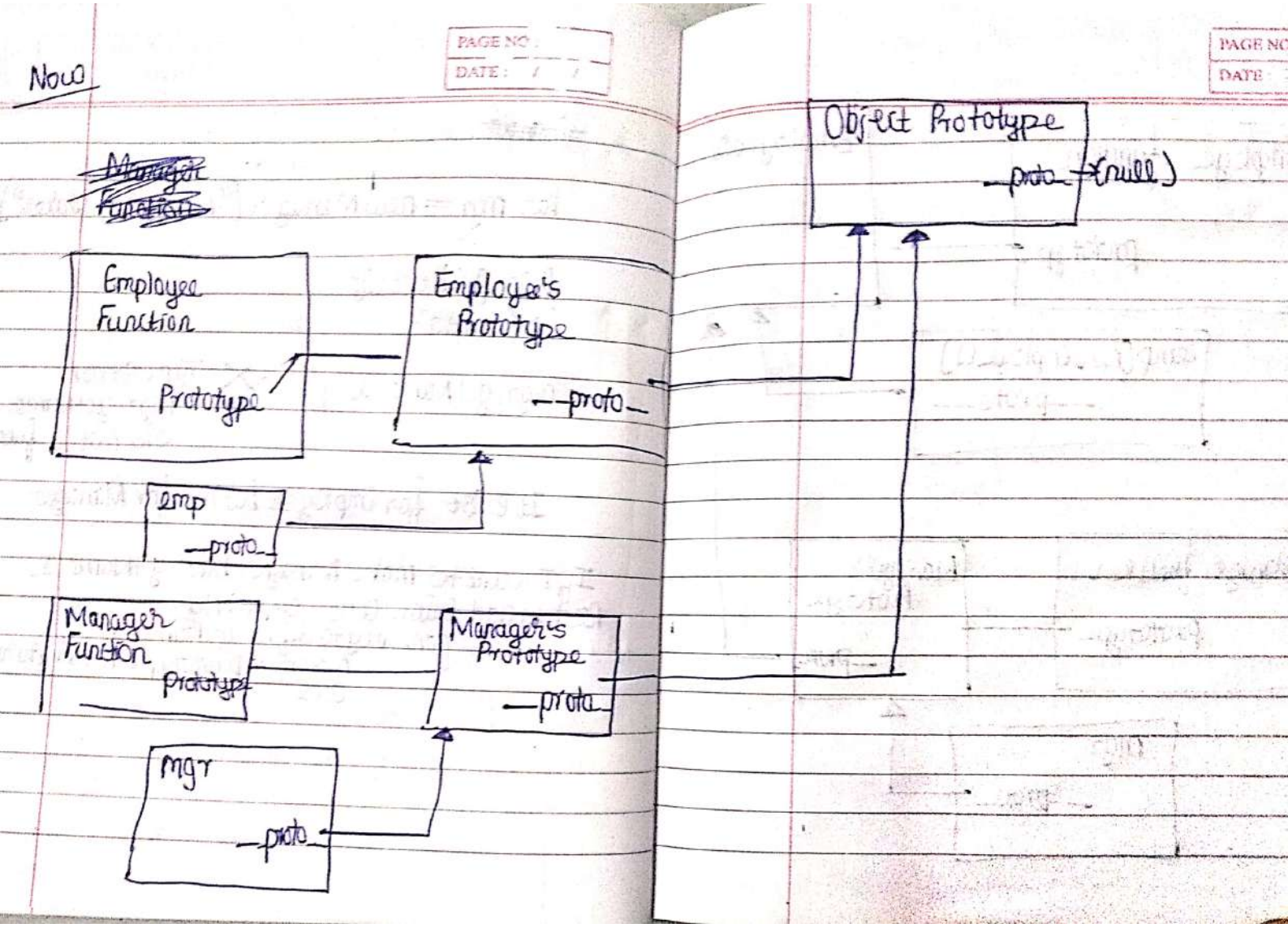
It exist for employee but not for Manager

If I want to make Manager have getName I need to add below code explicitly:

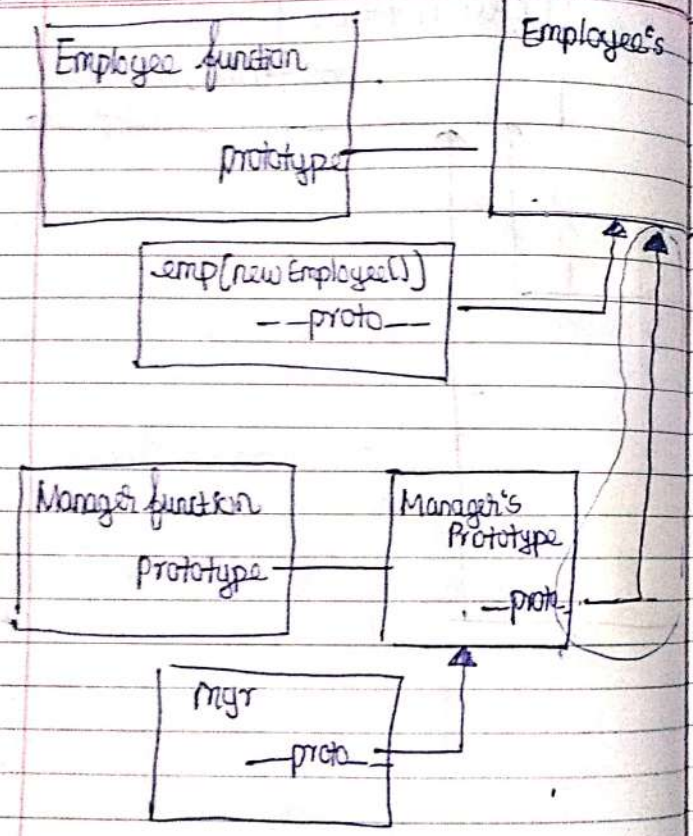
```
Manager.prototype = Object.create(Employee.prototype);
```

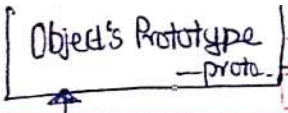
[one way to define getName on Manager.prototype but why to repeat code.]

Now



We Work





Code

I am not showing all return → show return

```
mgr.__proto__.__proto__ = Employee.prototype

mgr.getName();
↳ Michael

Employee.prototype.getSalary = function()
{ return 100;
}

emp1.getSalary();
→ 100

mgr.getSalary();
→ 100
```

Hopefully make sense

PAGE NO
DATE : /

You can go Crazy with it

Conclusion

Summary

Recommended → You don't know JS.