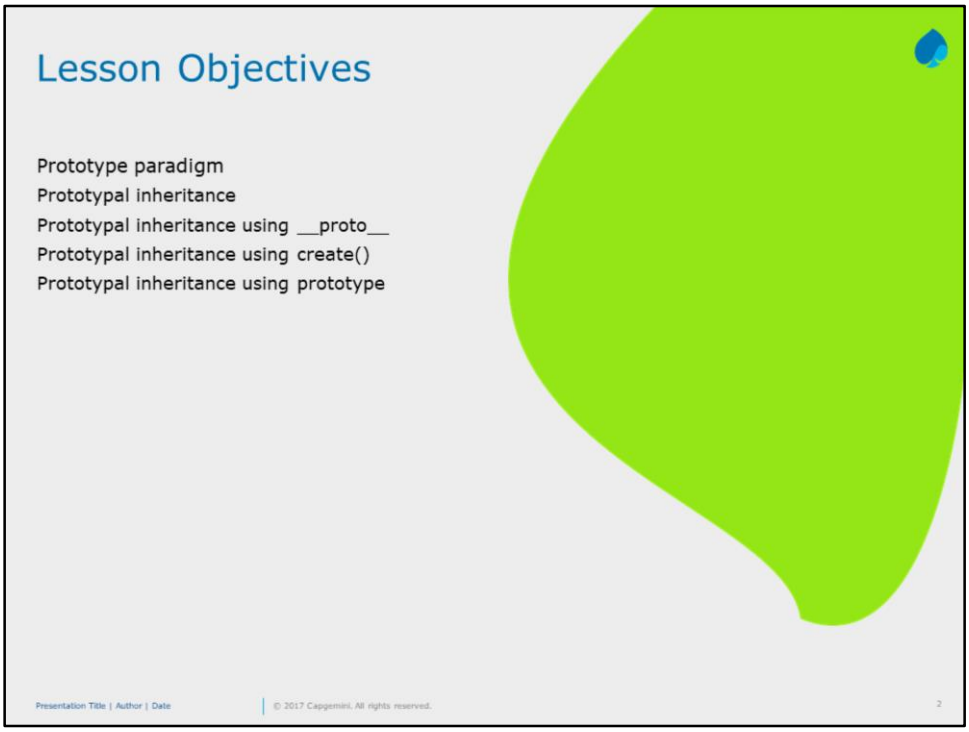
Instructor Notes:

Add instructor notes here.

Instructor Notes:

Add instructor notes here.



Following contents would be covered:

1.1 : What are Web services

1.1.1 Web service components and architecture

1.1.2 How do Web services work

1.2: HTTP and SOAP messages

1.3: Overview of JAX – WS and JAX – RS

Instructor Notes:

2.1: Prototype paradigm

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.

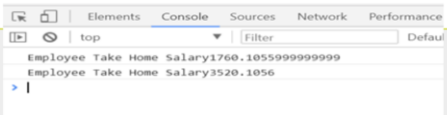
All properties and methods are assigned directly to the prototype property.

Instructor Notes:

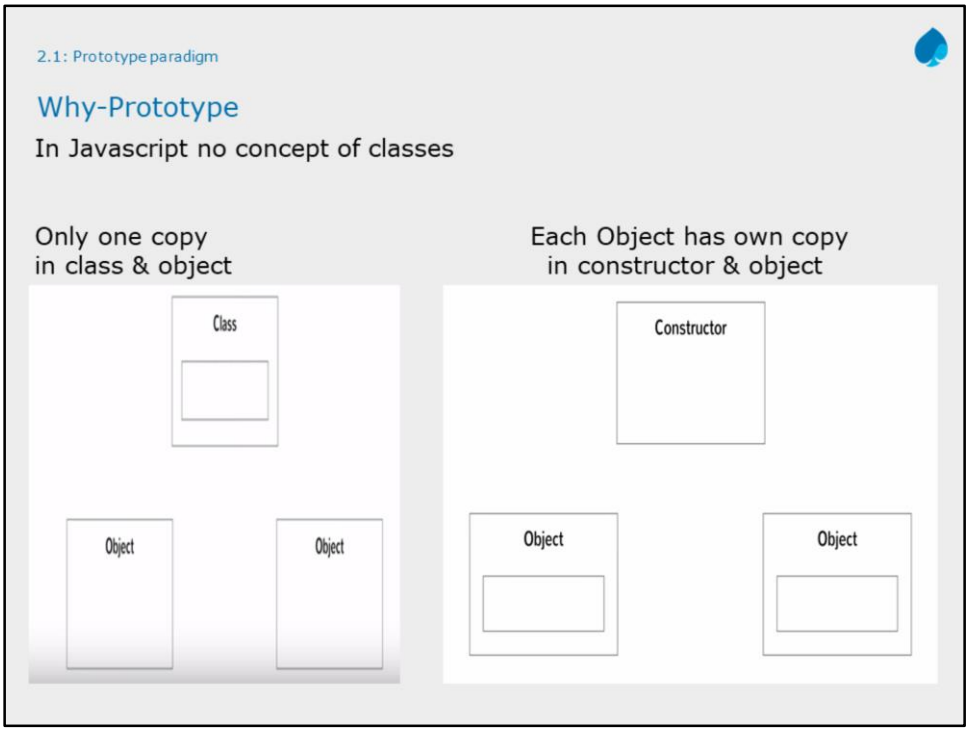
2.1: Prototype paradigm

Why-Prototype

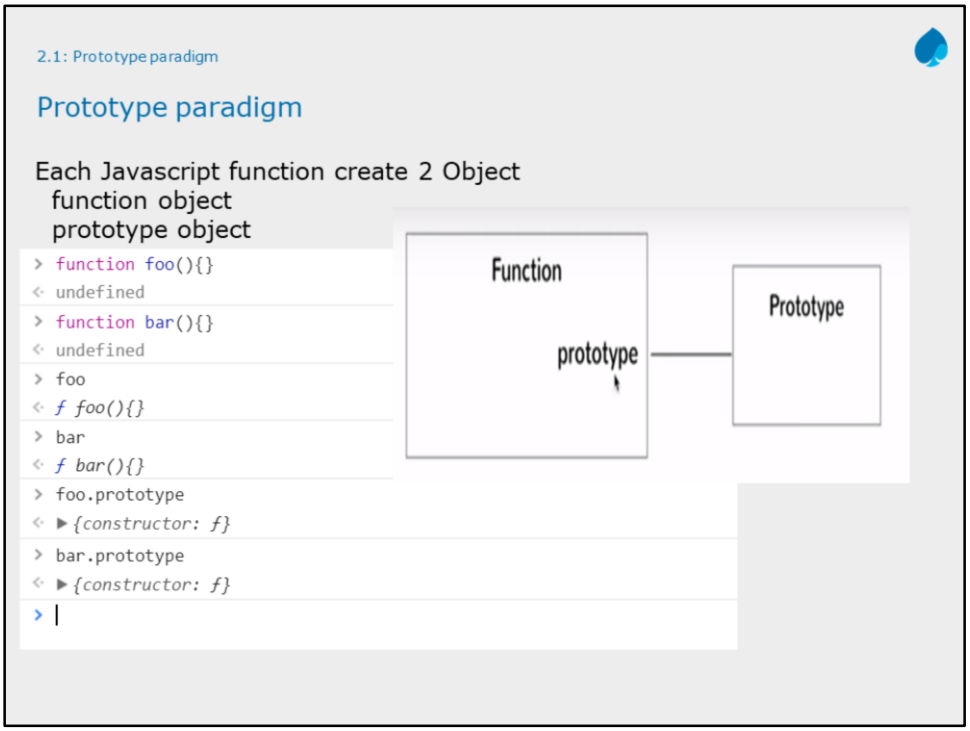
```
function createEmployee(empId,empName,empSalary,empDep){
    this.empId=empId;
    this.empName=empName;
    this.empSalary=empSalary;
    this.empDep=empDep;
    this.totalSalary;
    this.getTakeHomeSalary=function(){
        this.totalSalary=this.empSalary-(this.empSalary*0.12);
        console.log("Employee 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();
```



Instructor Notes:

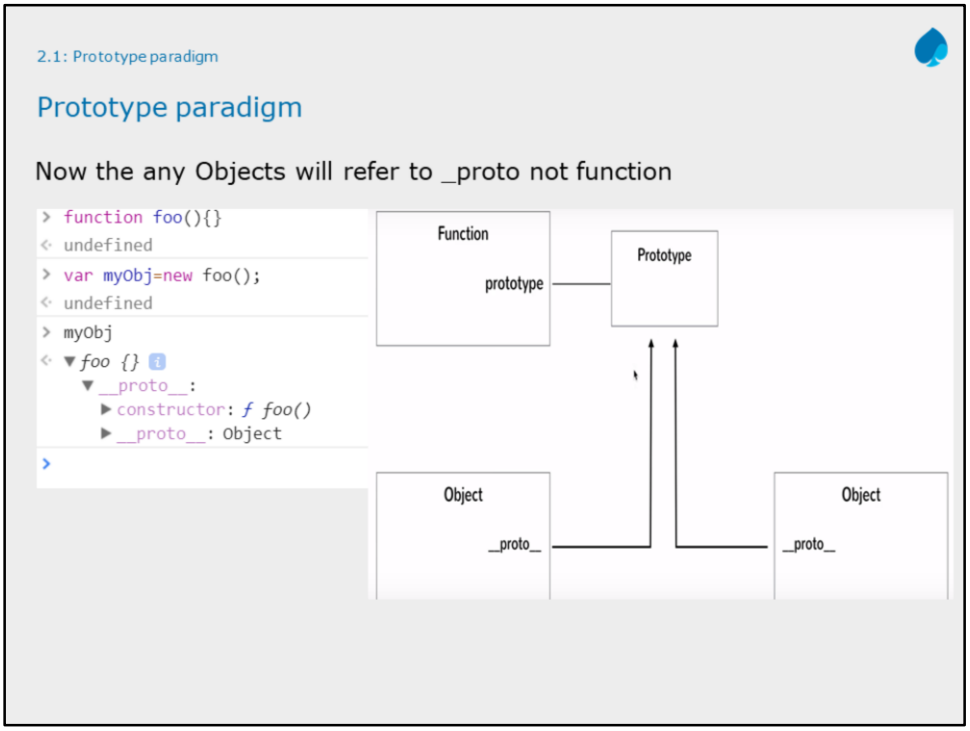


Instructor Notes:



When you attempt to access a property or method of an object, JavaScript will first search on the object itself, and if it is not found, it will search the object's [[Prototype]]. If after consulting both the object and its [[Prototype]] still no match is found, JavaScript will check the prototype of the linked object, and continue searching until the end of the prototype chain is reached.
At the end of the prototype chain is Object. prototype. All objects inherit the properties and methods of Object. Any attempt to search beyond the end of the chain results in null.

Instructor Notes:



In our example, x is an empty object that inherits from Object. x can use any property or method that Object has, such as toString().
This prototype chain is only one link long. x -> Object. We know this, because if we try to chain two [[Prototype]] properties together, it will be null.
x.__proto__.__proto__;
Output
null

Instructor Notes:

2.1: Prototype paradigm

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
> |
```

We can test this by creating a new array.
var y = [];
we could also write it as an array constructor,
var y = new Array().
If we take a look at the [[Prototype]] of the new y array, we will see that it has more properties and methods than the x object. It has inherited everything from Array.prototype.
y.__proto__; [constructor: *ƒ*, concat: *ƒ*, pop: *ƒ*, push: *ƒ*, …]

Other Way
var person2 = Object.create(person1);
What create() actually does is to create a new object from a specified prototype object. Here, person2 is being created using person1 's prototype as a prototype object.
person2.__proto__

Instructor Notes:

2.1: Prototype paradigm

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();
```

Instructor Notes:

2.1:Prototypal inheritance

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.

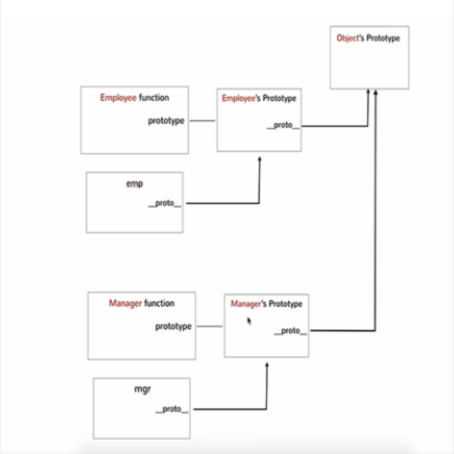
We can do inheritance by
Inheritance –By Using _proto_
By Using Object.create()

Instructor Notes:

2.1:Prototypal inheritance

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



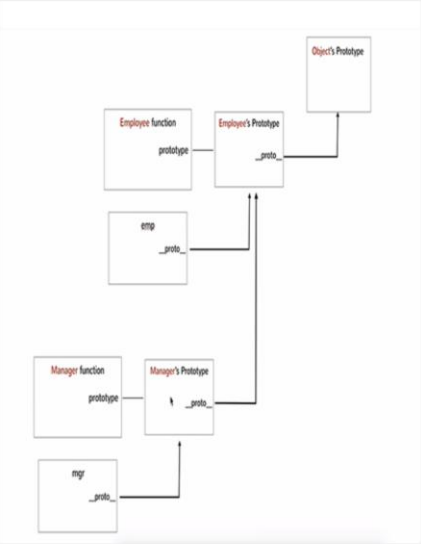

```
function Employee(name){
        this.name=name;
}

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()
```

This chain is now referring to Object.prototype. We can test the internal [[Prototype]] against the prototype property of the constructor function to see that they are referring to the same thing.
y.__proto__ === **Array**.prototype; // true y.__proto__.__proto__ === **Object**.prototype; // true We can also use the isPrototypeOf() method to accomplish this.
Array.prototype.isPrototypeOf(y); // true
Object.prototype.isPrototypeOf(**Array**); // true We can use the instanceof operator to test whether the prototype property of a constructor appears anywhere within an object's prototype chain.
y **instanceof Array**; // true

Instructor Notes:

2.2: Prototypal inheritance

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());
```

all JavaScript objects have a hidden, internal [[Prototype]] property (which may be exposed through __proto__ in some browsers). Objects can be extended and will inherit the properties and methods on [[Prototype]] of their constructor. These prototypes can be chained, and each additional object will inherit everything throughout the chain. The chain ends with the Object.prototype.

Instructor Notes:

2.2:Prototypal inheritance

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());
```

Instructor Notes:

2.2:Prototypal inheritance

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());
```

Instructor Notes:

Add instructor notes here.

Demo

Demo1
Demo2
Demo3
Demo4

Add the notes here.

Instructor Notes:

Add instructor notes here.

Lab

Lab 2

Add the notes here.

Instructor Notes:

Add instructor notes here.

Summary

In this lesson we have learned about -

Prototype paradigm
Prototypal inheritance
Prototypal inheritance using __proto__
Prototypal inheritance using create()
Prototypal inheritance using prototype