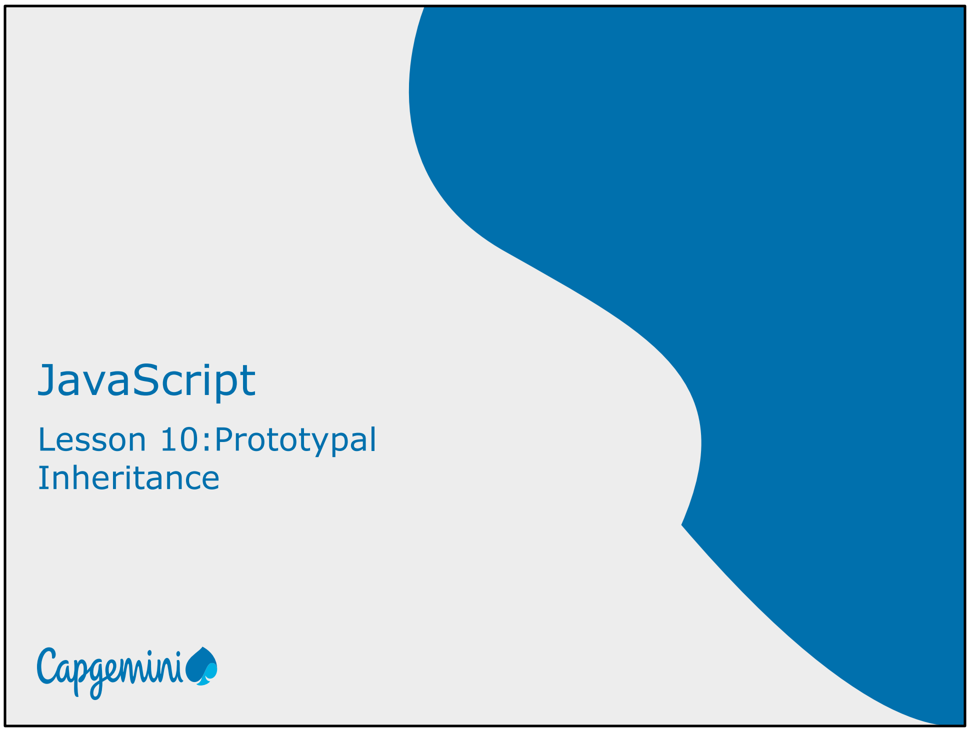


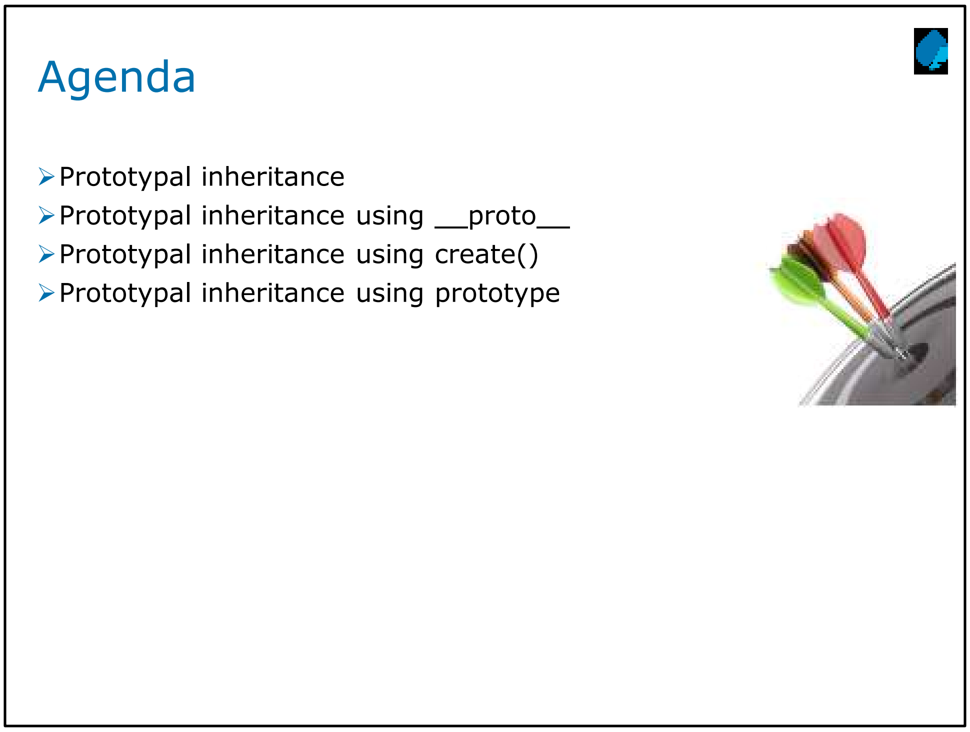
JavaScript

Lesson 10:Prototypal Inheritance

Capgemini

Objective

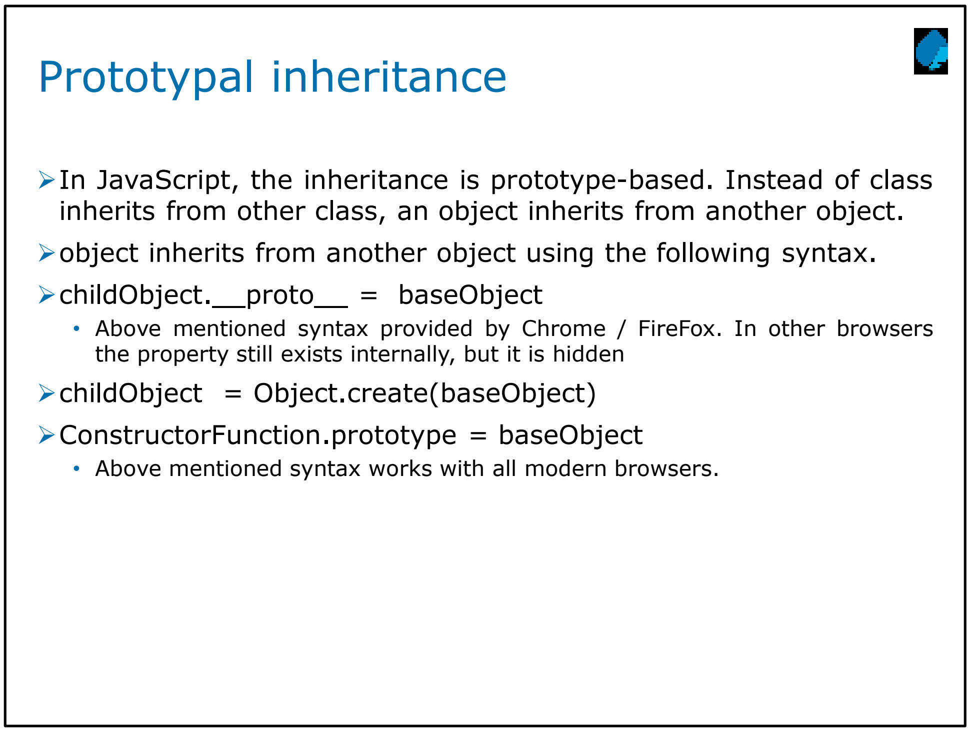
- At the end of this lesson participants will be able to –
  - Implement Inheritance using JavaScript



Agenda

- Prototypal inheritance
- Prototypal inheritance using __proto__
- Prototypal inheritance using create()
- Prototypal inheritance using prototype

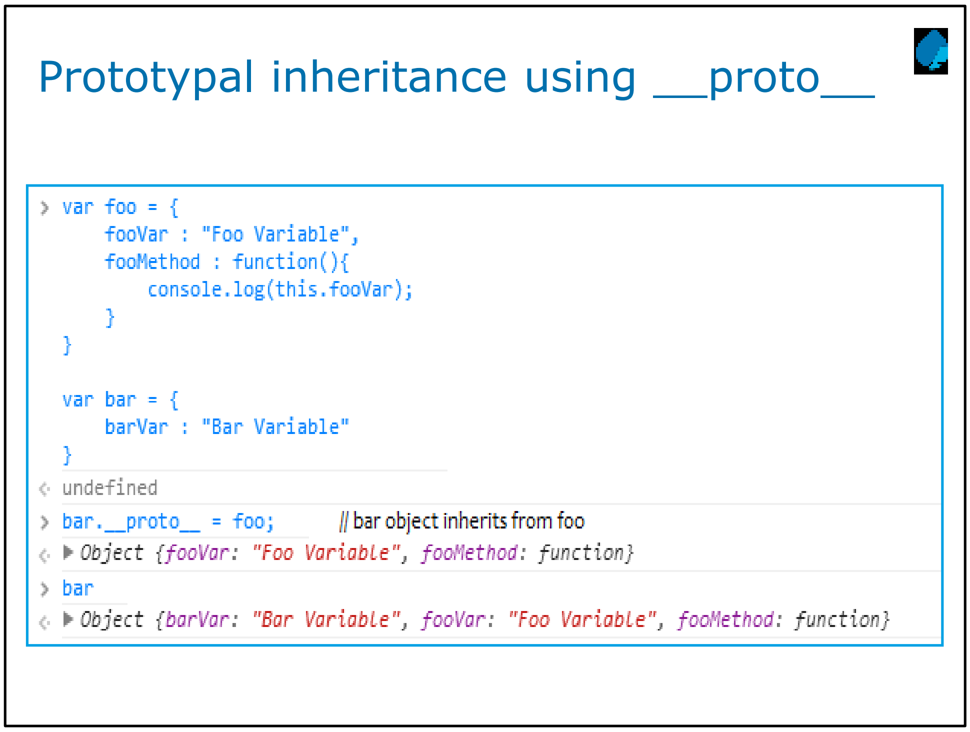
Add instructor notes here.



Prototypal inheritance

- In JavaScript, the inheritance is prototype-based. Instead of class inherits from other class, an object inherits from another object.
- object inherits from another object using the following syntax.
- childObject.__proto__ = baseObject
  - Above mentioned syntax provided by Chrome / FireFox. In other browsers the property still exists internally, but it is hidden
- childObject = Object.create(baseObject)
- ConstructorFunction.prototype = baseObject
  - Above mentioned syntax works with all modern browsers.

Add instructor notes here.



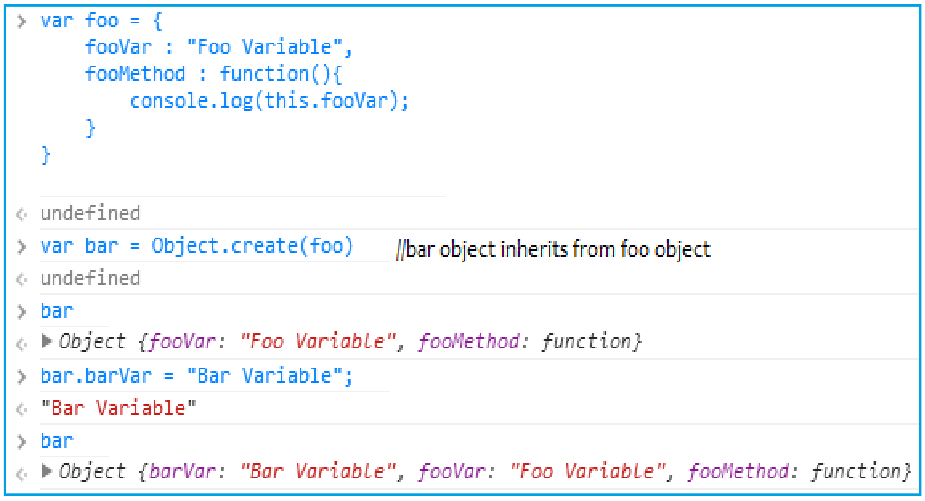
Prototypal inheritance using __proto__

```
> var foo = {
      fooVar : "Foo Variable",
      fooMethod : function(){
          console.log(this.fooVar);
      }
  }

  var bar = {
      barVar : "Bar Variable"
  }
< undefined
> bar.__proto__ = foo;        // bar object inherits from foo
< ▶ Object {fooVar: "Foo Variable", fooMethod: function}
> bar
< ▶ Object {barVar: "Bar Variable", fooVar: "Foo Variable", fooMethod: function}
```

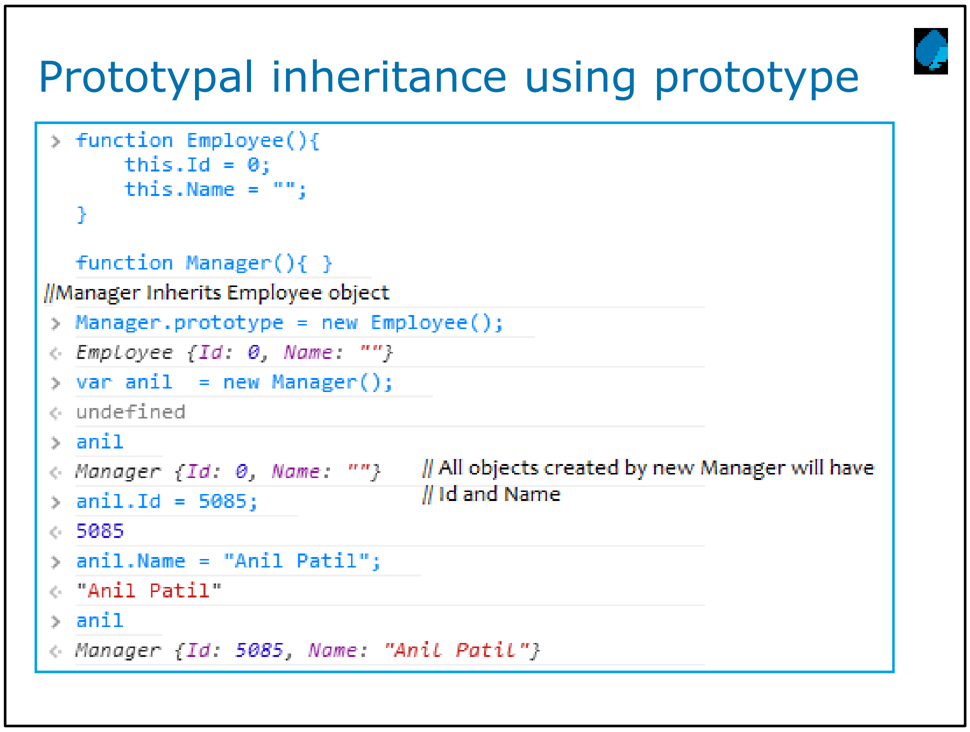
Add instructor notes here.

Prototypal inheritance using Object.create()



```
> var foo = {
      fooVar : "Foo Variable",
      fooMethod : function(){
          console.log(this.fooVar);
      }
  }
< undefined
> var bar = Object.create(foo)      //bar object inherits from foo object
< undefined
> bar
< ▶Object {fooVar: "Foo Variable", fooMethod: function}
> bar.barVar = "Bar Variable";
< "Bar Variable"
> bar
< ▶Object {barVar: "Bar Variable", fooVar: "Foo Variable", fooMethod: function}
```

Add instructor notes here.



Prototypal inheritance using prototype

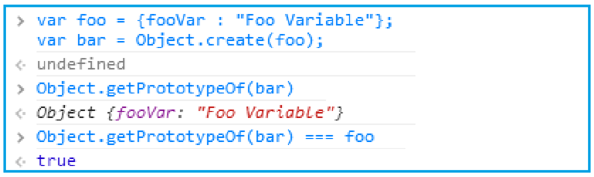
```
> function Employee(){
      this.Id = 0;
      this.Name = "";
  }

  function Manager(){ }
//Manager Inherits Employee object
> Manager.prototype = new Employee();
< Employee {Id: 0, Name: ""}
> var anil  = new Manager();
< undefined
> anil
< Manager {Id: 0, Name: ""}    // All objects created by new Manager will have
> anil.Id = 5085;              // Id and Name
< 5085
> anil.Name = "Anil Patil";
< "Anil Patil"
> anil
< Manager {Id: 5085, Name: "Anil Patil"}
```

Add instructor notes here.

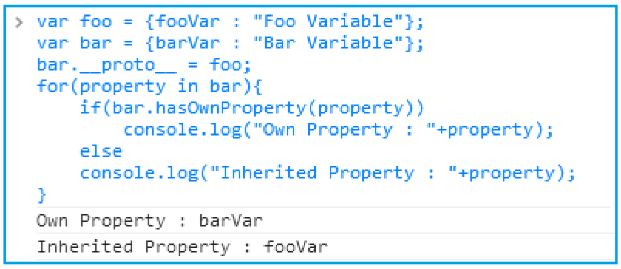
Prototypal inheritance

- Object.getPrototypeOf(obj) returns the value of obj.__proto__.



```
> var foo = {fooVar : "Foo Variable"};
  var bar = Object.create(foo);
<- undefined
> Object.getPrototypeOf(bar)
<- Object {fooVar: "Foo Variable"}
> Object.getPrototypeOf(bar) === foo
<- true
```

- for..in loop lists properties in the object and its prototype chain. obj.hasOwnProperty(prop) returns true if property belongs to that object.

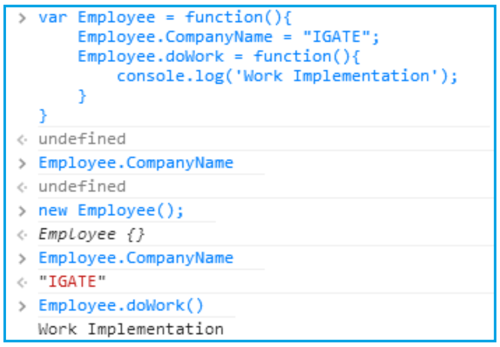


```
> var foo = {fooVar : "Foo Variable"};
  var bar = {barVar : "Bar Variable"};
  bar.__proto__ = foo;
  for(property in bar){
      if(bar.hasOwnProperty(property))
          console.log("Own Property : "+property);
      else
      console.log("Inherited Property : "+property);
  }
  Own Property : barVar
  Inherited Property : fooVar
```

Add instructor notes here.

Static variables and methods

- In JavaScript we can directly put data into function object which acts like Static member.
- Static Members need to be accessed directly by Object name, cannot be accessed by reference variable. Static members gets created when the first object gets created.



```
> var Employee = function(){
      Employee.CompanyName = "IGATE";
      Employee.doWork = function(){
          console.log('Work Implementation');
      }
  }
<· undefined
> Employee.CompanyName
<· undefined
> new Employee();
<· Employee {}
> Employee.CompanyName
<· "IGATE"
> Employee.doWork()
  Work Implementation
```

In JavaScript :

private variables are declared with the 'var' keyword inside the object, and can only be accessed by private functions and privileged methods.

private functions are declared inline inside the object's constructor (or alternatively may be defined via var functionName=function(){...}) and may only be called by privileged methods (including the object's constructor).

privileged methods are declared with this.methodName=function(){...} and may invoked by code external to the object.

public properties are declared with this.variableName and may be read/written from outside the object.

public methods are defined by Classname.prototype.methodName = function(){...} and may be called from outside the object.

prototype properties are defined by Classname.prototype.propertyName = someValue

static properties are defined by Classname.propertyName = someValue

Summary

- In this lesson we have learned about -
  - Prototypal inheritance
  - Prototypal inheritance using __proto__
  - Prototypal inheritance using create()
  - Prototypal inheritance using prototype