Instructor Notes:

Add instructor notes here.

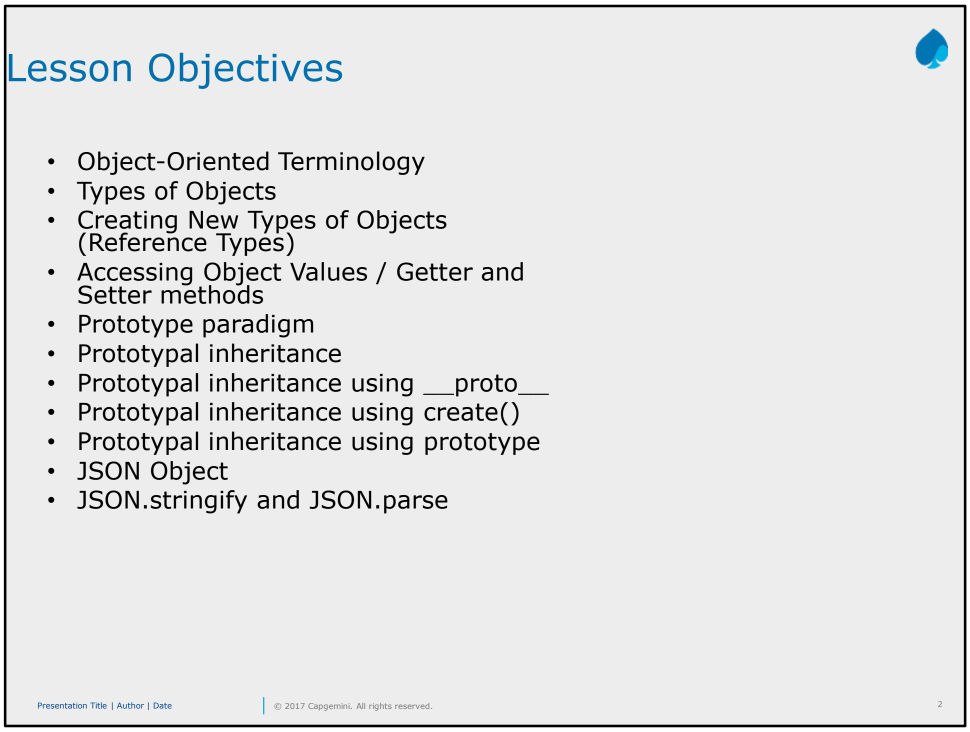
JavaScript ES6

Lesson 10-Advance JavaScript

Capgemini

Instructor Notes:

Add instructor notes here.



Lesson Objectives

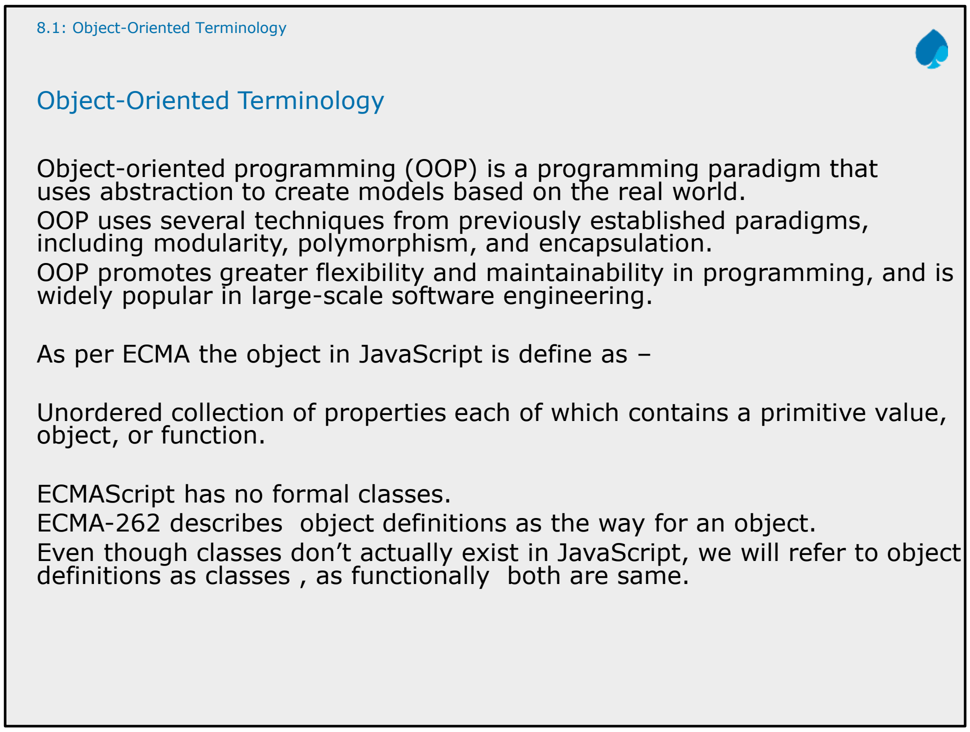
- Object-Oriented Terminology
- Types of Objects
- Creating New Types of Objects (Reference Types)
- Accessing Object Values / Getter and Setter methods
- Prototype paradigm
- Prototypal inheritance
- Prototypal inheritance using __proto__
- Prototypal inheritance using create()
- Prototypal inheritance using prototype
- JSON Object
- JSON.stringify and JSON.parse

Presentation Title | Author | Date

2

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:



8.1: Object-Oriented Terminology

Object-Oriented Terminology

Object-oriented programming (OOP) is a programming paradigm that uses abstraction to create models based on the real world.
OOP uses several techniques from previously established paradigms, including modularity, polymorphism, and encapsulation.
OOP promotes greater flexibility and maintainability in programming, and is widely popular in large-scale software engineering.

As per ECMA the object in JavaScript is define as –

Unordered collection of properties each of which contains a primitive value, object, or function.

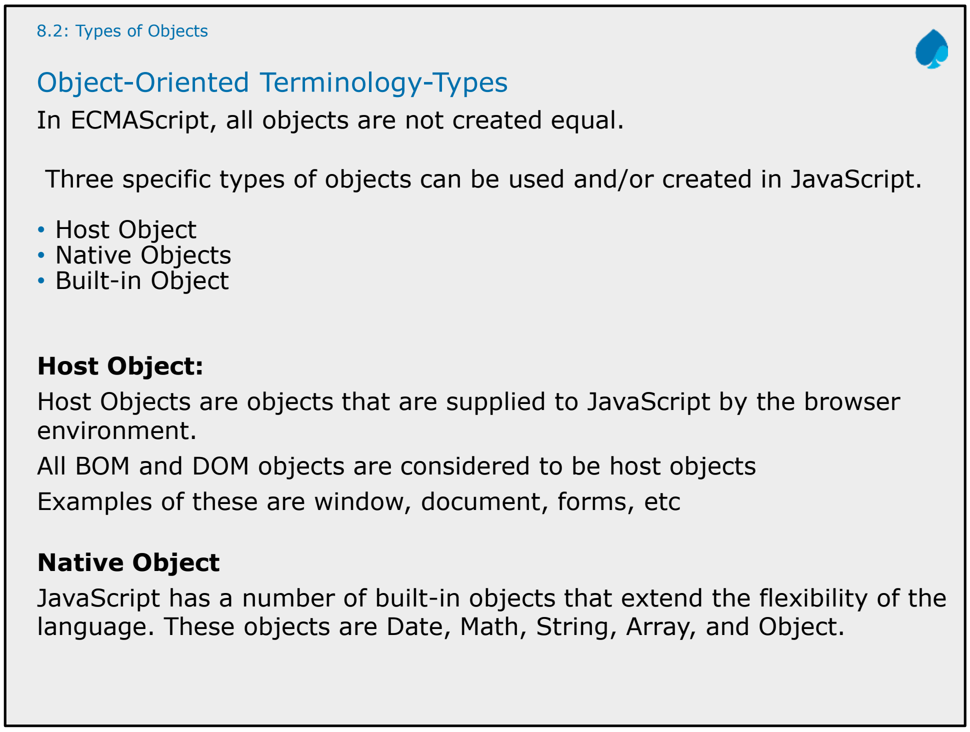
ECMAScript has no formal classes.
ECMA-262 describes object definitions as the way for an object.
Even though classes don't actually exist in JavaScript, we will refer to object definitions as classes , as functionally both are same.

Ecma International is an industry association founded in 1961 and dedicated to the standardization of Information and Communication Technology (ICT) and Consumer Electronics

ECMAScript is the scripting language standardized by Ecma International in the ECMA-262 specification and ISO/IEC 16262. The language is widely used for client-side scripting on the web, in the form of several well-known implementations such as JavaScript, JScript and ActionScript.

Terminology
Namespace A container which lets developers bundle all functionality under a unique, application-specific name.Class Defines the object's characteristics. A class is a template definition of an object's properties and methods.Object An instance of a class.Property An object characteristic, such as color.Method An object capability, such as walk. It is a subroutine or function associated with a class.Constructor A method called at the moment an object is instantiated. It usually has the same name as the class containing it.Inheritance A class can inherit characteristics from another class.Encapsulation A method of bundling the data and methods that use the data.Abstraction The conjunction of an object's complex inheritance, methods, and properties must adequately reflect a reality model.Polymorphism Poly means "*many*" and morphism means "*forms*". Different classes might define the same method or property.

Instructor Notes:



8.2: Types of Objects

Object-Oriented Terminology-Types

In ECMAScript, all objects are not created equal.

Three specific types of objects can be used and/or created in JavaScript.

- Host Object
- Native Objects
- Built-in Object

Host Object:

Host Objects are objects that are supplied to JavaScript by the browser environment.

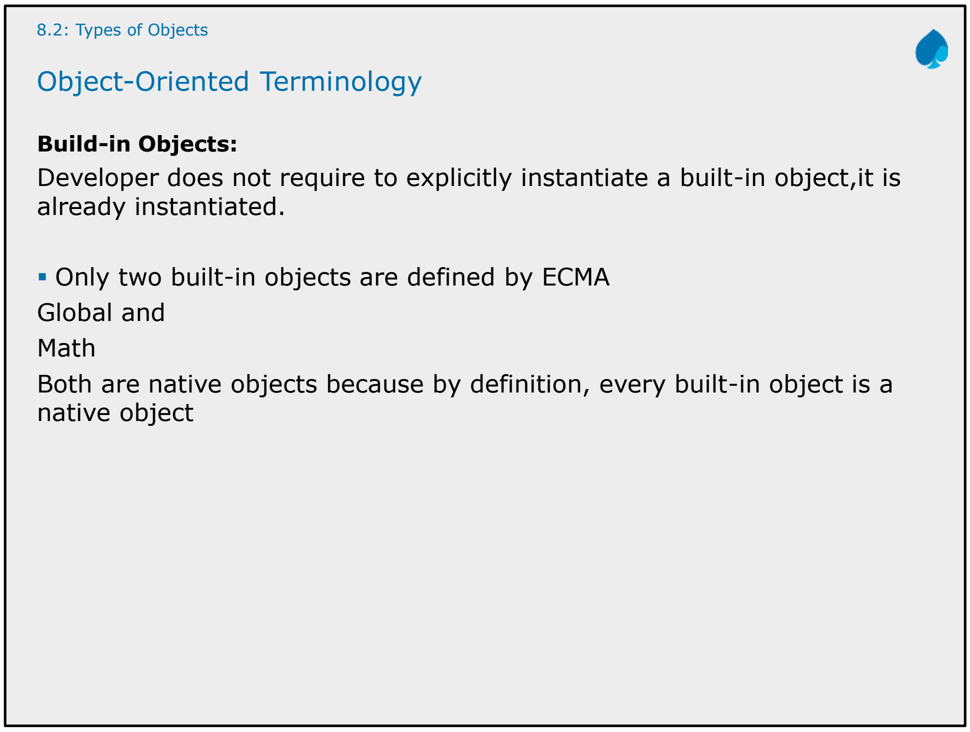
All BOM and DOM objects are considered to be host objects

Examples of these are window, document, forms, etc

Native Object

JavaScript has a number of built-in objects that extend the flexibility of the language. These objects are Date, Math, String, Array, and Object.

Instructor Notes:



8.2: Types of Objects

Object-Oriented Terminology

Build-in Objects:

Developer does not require to explicitly instantiate a built-in object,it is already instantiated.

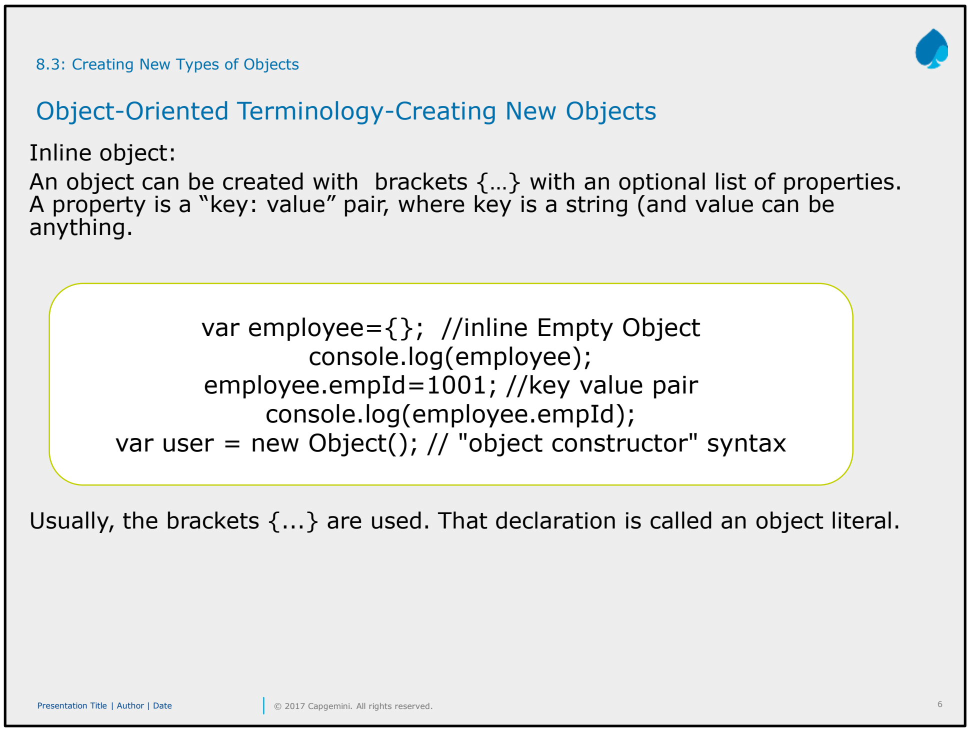
- Only two built-in objects are defined by ECMA

Global and

Math

Both are native objects because by definition, every built-in object is a native object

Instructor Notes:



8.3: Creating New Types of Objects

Object-Oriented Terminology-Creating New Objects

Inline object:

An object can be created with brackets {…} with an optional list of properties. A property is a "key: value" pair, where key is a string (and value can be anything.

```
var employee={};  //inline Empty Object
console.log(employee);
employee.empId=1001; //key value pair
console.log(employee.empId);
var user = new Object(); // "object constructor" syntax
```

Usually, the brackets {...} are used. That declaration is called an object literal.

Presentation Title | Author | Date © 2017 Capgemini. All rights reserved. 6

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction

A web service is a collection of open protocols and standards (promotes interoperability between clients / servers without the need of proprietary or trademark software)

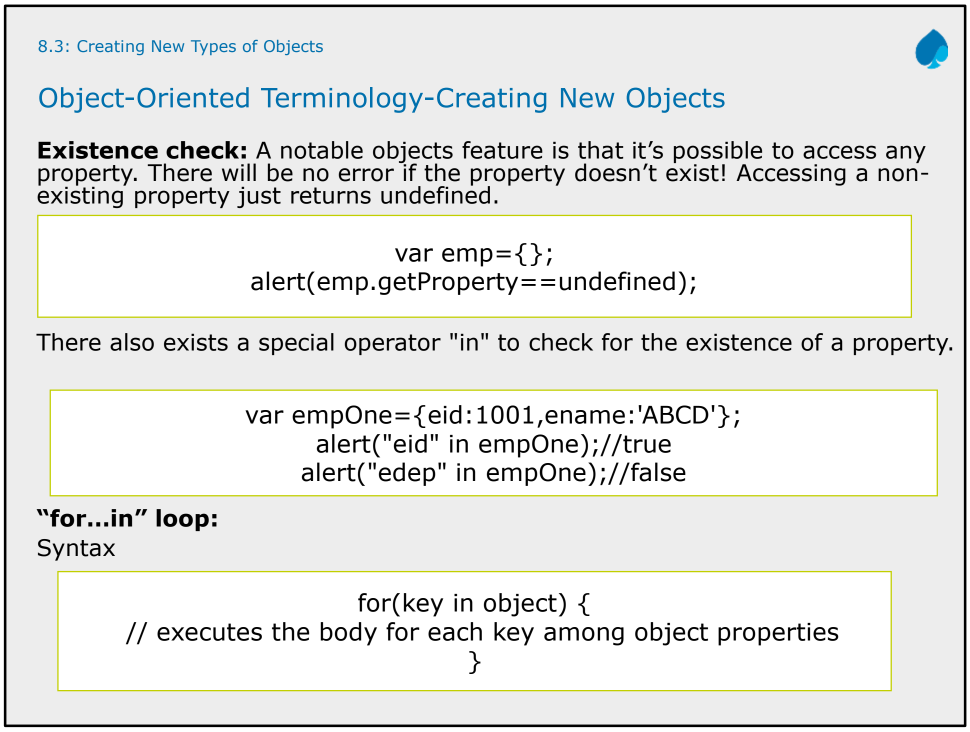
Software applications written in various programming languages and running on various platforms can use web services to exchange data. For example, interoperability between Java and Python, or Windows and Linux applications can be facilitated through web services.

Web services has the ability to go through firewalls.

Web services are available anytime, anywhere and on any device.

Web services can be used, if clients are scattered across the web.

Instructor Notes:



8.3: Creating New Types of Objects

Object-Oriented Terminology-Creating New Objects

Existence check: A notable objects feature is that it’s possible to access any property. There will be no error if the property doesn’t exist! Accessing a non-existing property just returns undefined.

```
var emp={};
alert(emp.getProperty==undefined);
```

There also exists a special operator "in" to check for the existence of a property.

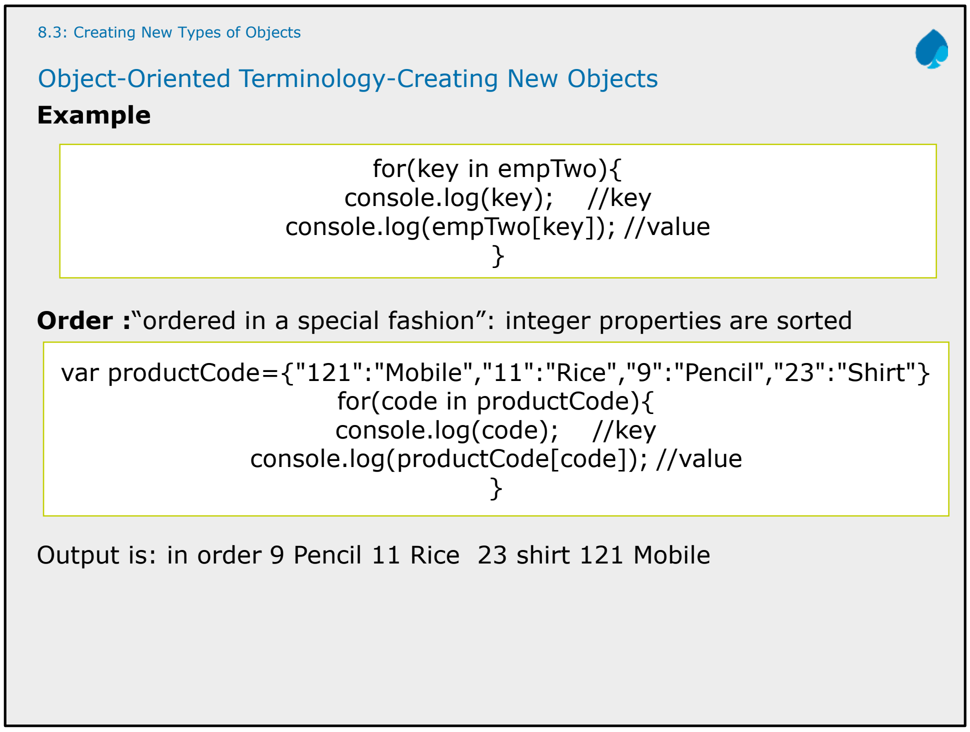
```
var empOne={eid:1001,ename:'ABCD'};
alert("eid" in empOne);//true
alert("edep" in empOne);//false
```

“for…in” loop:

Syntax

```
for(key in object) {
// executes the body for each key among object properties
}
```

Instructor Notes:



8.3: Creating New Types of Objects

Object-Oriented Terminology-Creating New Objects

Example

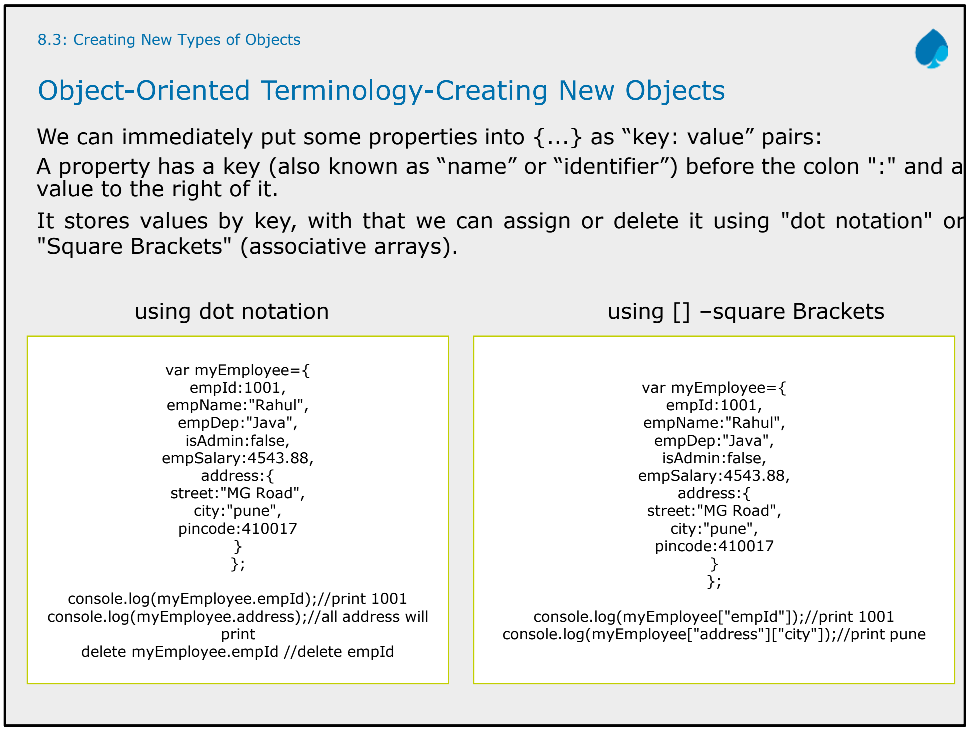
```
for(key in empTwo){
console.log(key);    //key
console.log(empTwo[key]); //value
}
```

Order :“ordered in a special fashion”: integer properties are sorted

```
var productCode={"121":"Mobile","11":"Rice","9":"Pencil","23":"Shirt"}
for(code in productCode){
console.log(code);    //key
console.log(productCode[code]); //value
}
```

Output is: in order 9 Pencil 11 Rice 23 shirt 121 Mobile

Instructor Notes:



8.3: Creating New Types of Objects

Object-Oriented Terminology-Creating New Objects

We can immediately put some properties into {...} as "key: value" pairs:

A property has a key (also known as "name" or "identifier") before the colon ":" and a value to the right of it.

It stores values by key, with that we can assign or delete it using "dot notation" or "Square Brackets" (associative arrays).

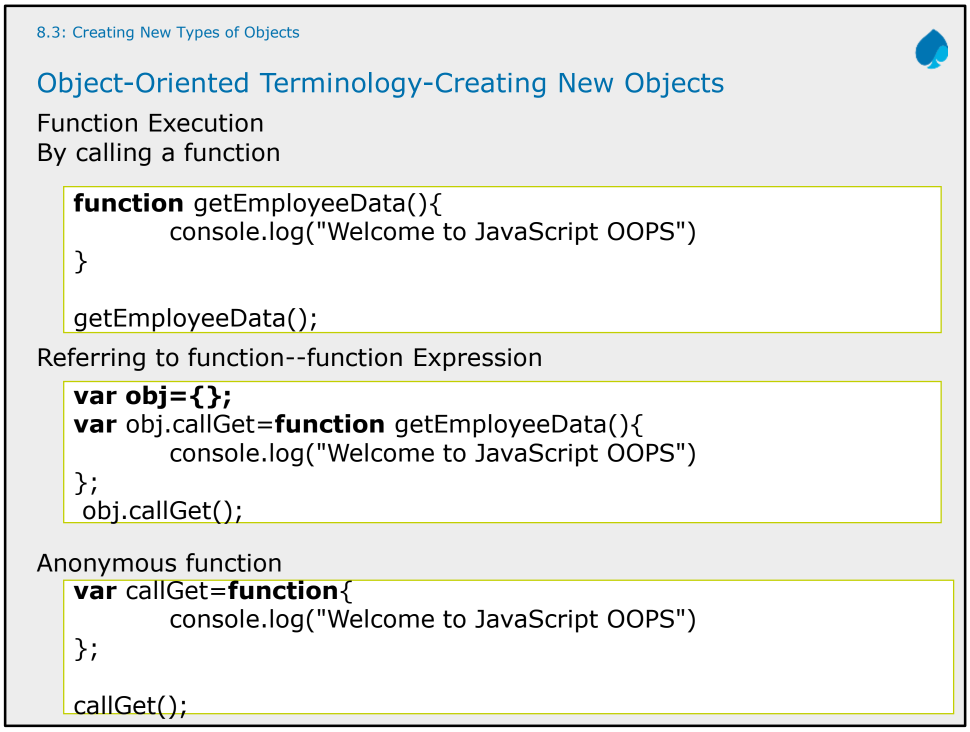
using dot notation

```
var myEmployee={
empId:1001,
empName:"Rahul",
empDep:"Java",
isAdmin:false,
empSalary:4543.88,
address:{
street:"MG Road",
city:"pune",
pincode:410017
}
};

console.log(myEmployee.empId);//print 1001
console.log(myEmployee.address);//all address will
print
delete myEmployee.empId //delete empId
```

using [] –square Brackets

```
var myEmployee={
empId:1001,
empName:"Rahul",
empDep:"Java",
isAdmin:false,
empSalary:4543.88,
address:{
street:"MG Road",
city:"pune",
pincode:410017
}
};

console.log(myEmployee["empId"]);//print 1001
console.log(myEmployee["address"]["city"]);//print pune
```

Instructor Notes:



8.3: Creating New Types of Objects

Object-Oriented Terminology-Creating New Objects

Function Execution
By calling a function

```
function getEmployeeData(){
          console.log("Welcome to JavaScript OOPS")
}

getEmployeeData();
```

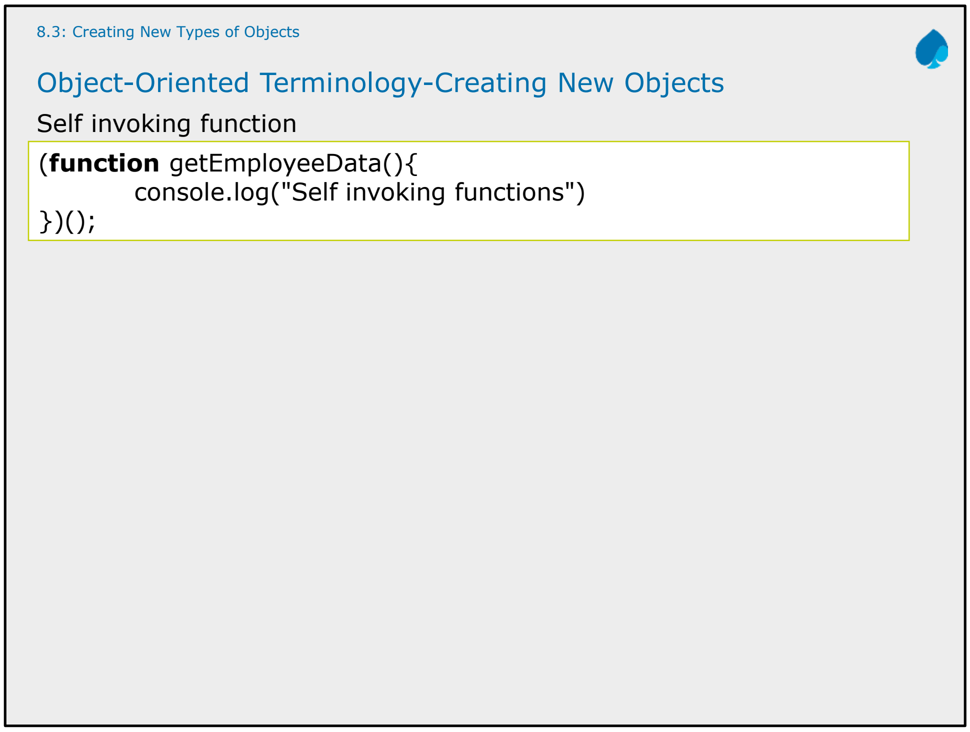
Referring to function--function Expression

```
var obj={};
var obj.callGet=function getEmployeeData(){
          console.log("Welcome to JavaScript OOPS")
};
 obj.callGet();
```

Anonymous function

```
var callGet=function{
          console.log("Welcome to JavaScript OOPS")
};

callGet();
```

Instructor Notes:



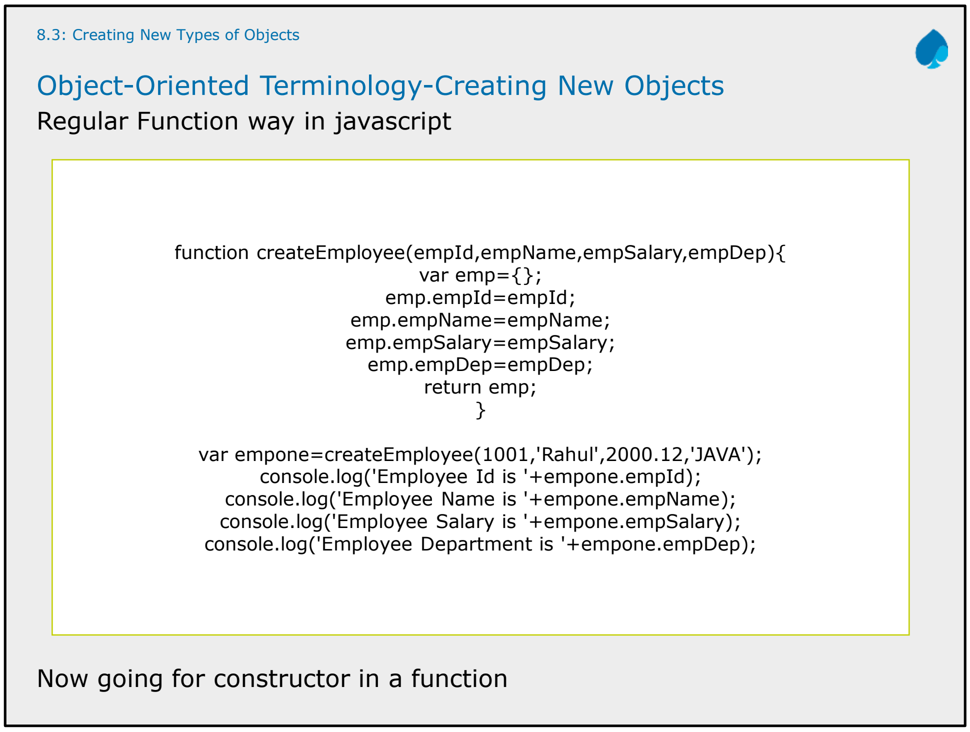
8.3: Creating New Types of Objects

Object-Oriented Terminology-Creating New Objects

Self invoking function

```
(function getEmployeeData(){
        console.log("Self invoking functions")
})();
```

Instructor Notes:



8.3: Creating New Types of Objects

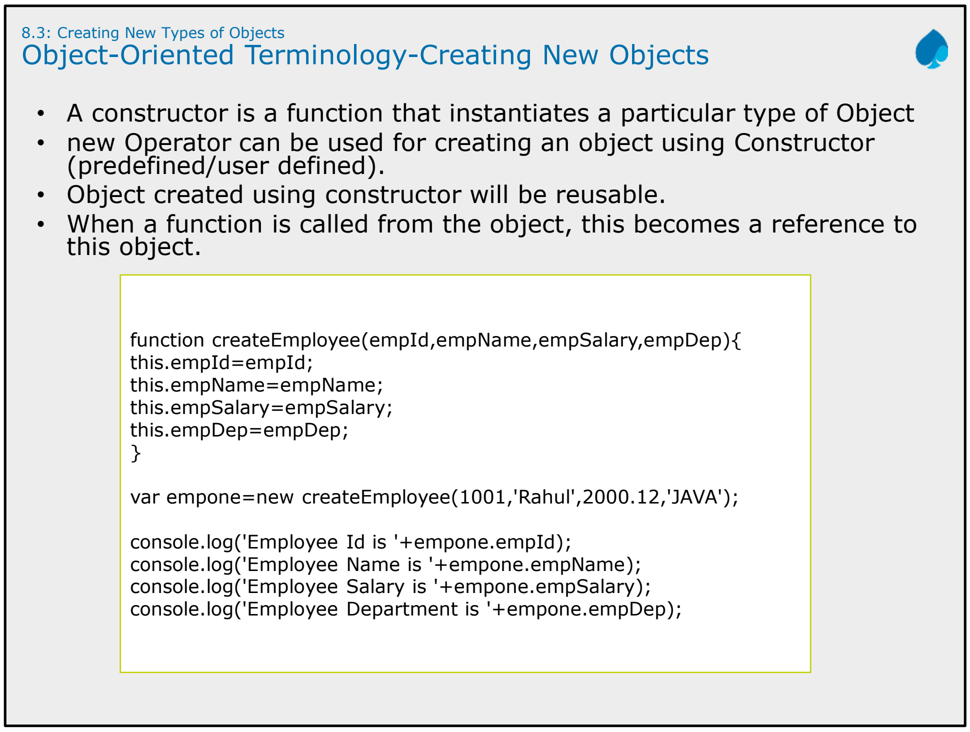
Object-Oriented Terminology-Creating New Objects

Regular Function way in javascript

```
function createEmployee(empId,empName,empSalary,empDep){
var emp={};
emp.empId=empId;
emp.empName=empName;
emp.empSalary=empSalary;
emp.empDep=empDep;
return emp;
}

var empone=createEmployee(1001,'Rahul',2000.12,'JAVA');
console.log('Employee Id is '+empone.empId);
console.log('Employee Name is '+empone.empName);
console.log('Employee Salary is '+empone.empSalary);
console.log('Employee Department is '+empone.empDep);
```

Now going for constructor in a function

Instructor Notes:



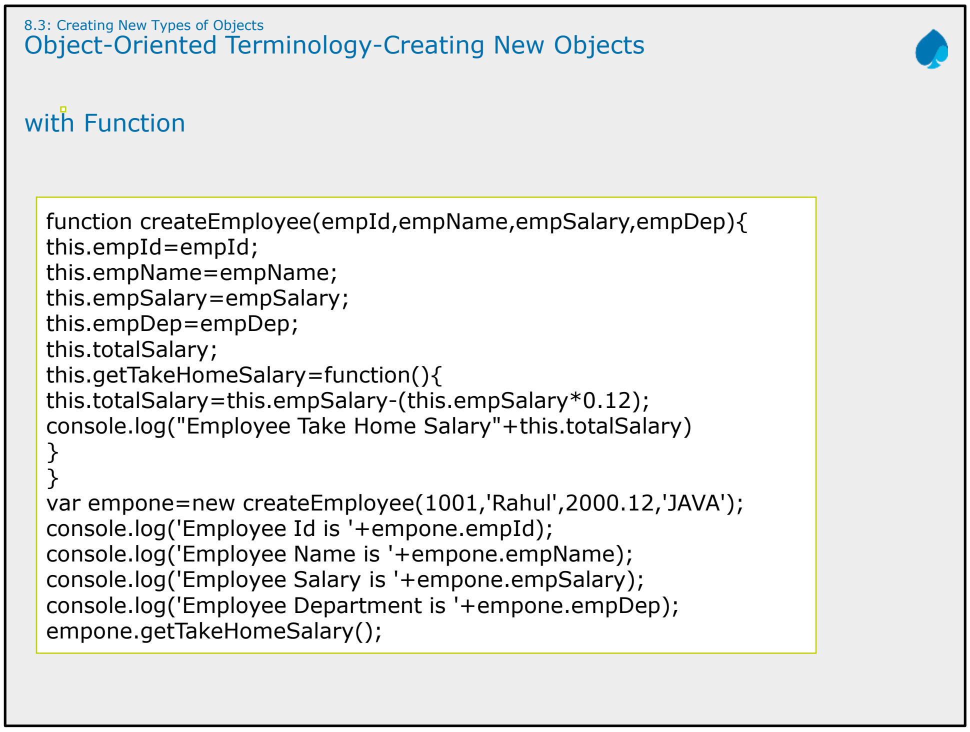
8.3: Creating New Types of Objects

Object-Oriented Terminology-Creating New Objects

- A constructor is a function that instantiates a particular type of Object
- new Operator can be used for creating an object using Constructor (predefined/user defined).
- Object created using constructor will be reusable.
- When a function is called from the object, this becomes a reference to this object.

```
function createEmployee(empId,empName,empSalary,empDep){
this.empId=empId;
this.empName=empName;
this.empSalary=empSalary;
this.empDep=empDep;
}

var empone=new createEmployee(1001,'Rahul',2000.12,'JAVA');

console.log('Employee Id is '+empone.empId);
console.log('Employee Name is '+empone.empName);
console.log('Employee Salary is '+empone.empSalary);
console.log('Employee Department is '+empone.empDep);
```

Instructor Notes:



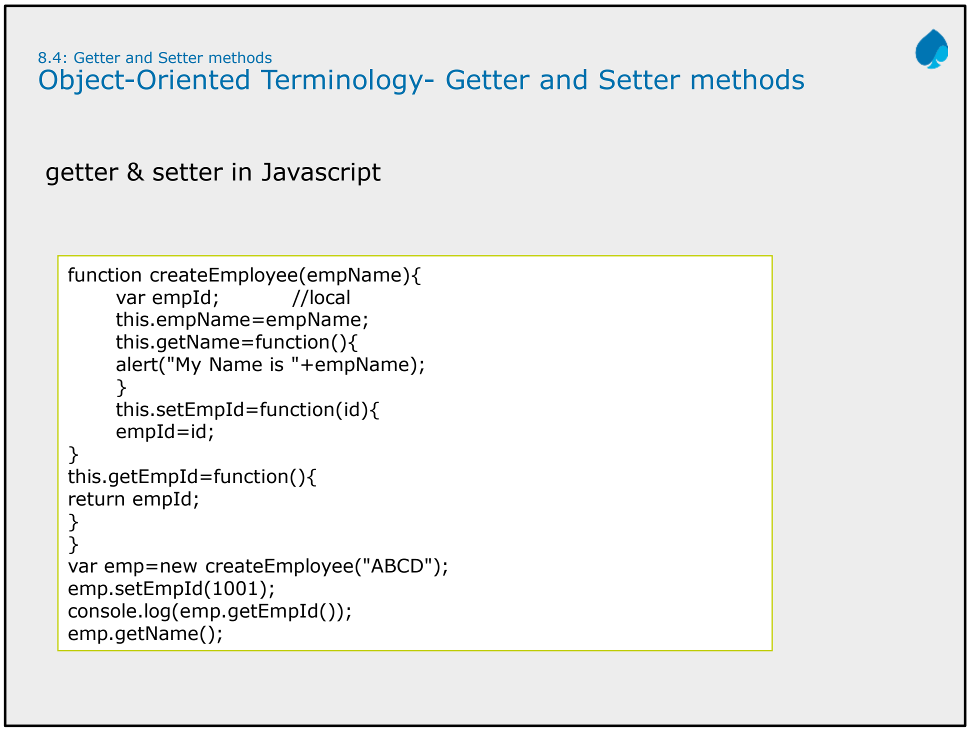
8.3: Creating New Types of Objects

Object-Oriented Terminology-Creating New Objects

with Function

```
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');
console.log('Employee Id is '+empone.empId);
console.log('Employee Name is '+empone.empName);
console.log('Employee Salary is '+empone.empSalary);
console.log('Employee Department is '+empone.empDep);
empone.getTakeHomeSalary();
```

Instructor Notes:



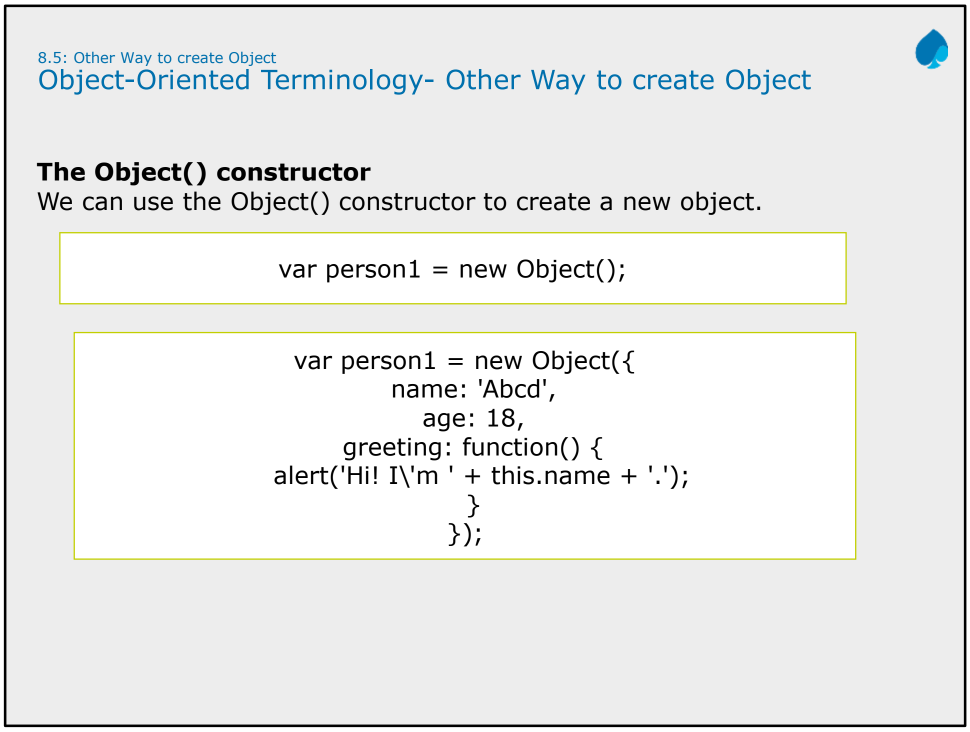
8.4: Getter and Setter methods

Object-Oriented Terminology- Getter and Setter methods

getter & setter in Javascript

```
function createEmployee(empName){
        var empId;            //local
        this.empName=empName;
        this.getName=function(){
        alert("My Name is "+empName);
        }
        this.setEmpId=function(id){
        empId=id;
}
this.getEmpId=function(){
return empId;
}
}
var emp=new createEmployee("ABCD");
emp.setEmpId(1001);
console.log(emp.getEmpId());
emp.getName();
```

Instructor Notes:



8.5: Other Way to create Object

Object-Oriented Terminology- Other Way to create Object

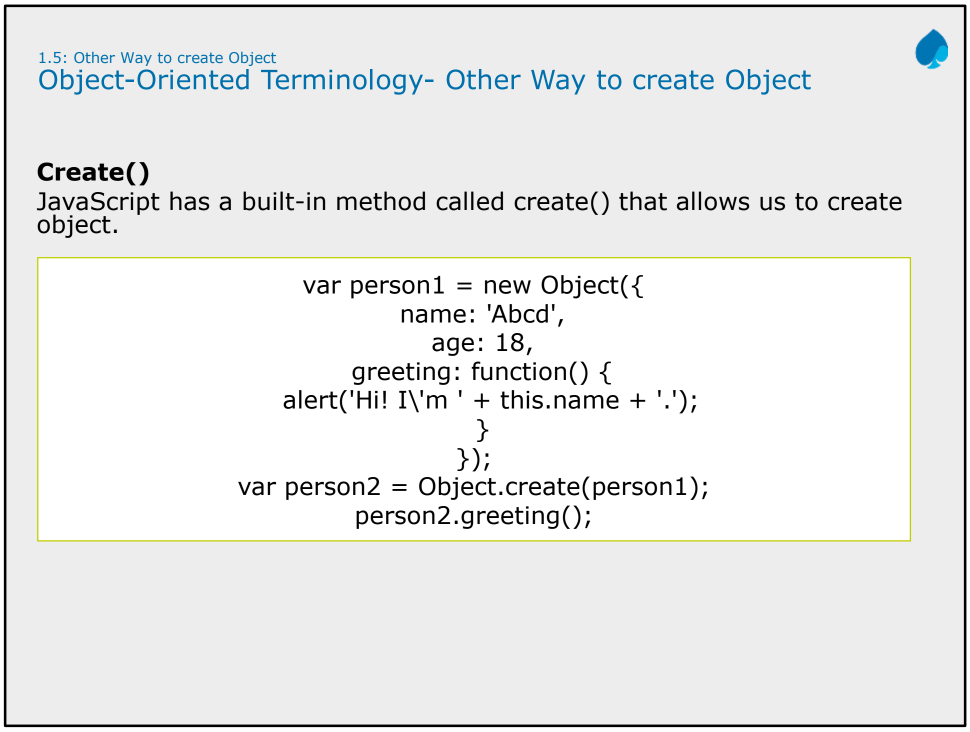
The Object() constructor

We can use the Object() constructor to create a new object.

```
var person1 = new Object();
```

```
var person1 = new Object({
name: 'Abcd',
age: 18,
greeting: function() {
alert('Hi! I\'m ' + this.name + '.');
}
});
```

Instructor Notes:



1.5: Other Way to create Object

Object-Oriented Terminology- Other Way to create Object

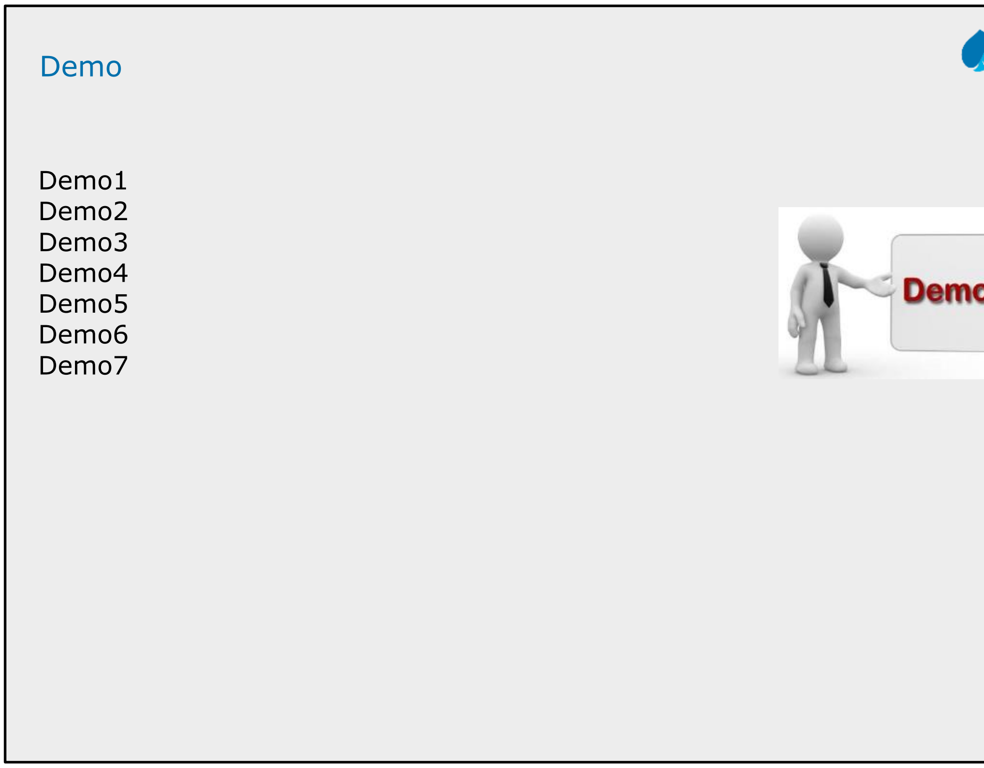
Create()

JavaScript has a built-in method called create() that allows us to create object.

```
var person1 = new Object({
name: 'Abcd',
age: 18,
greeting: function() {
alert('Hi! I\'m ' + this.name + '.');
}
});
var person2 = Object.create(person1);
person2.greeting();
```

Instructor Notes:

Add instructor notes here.



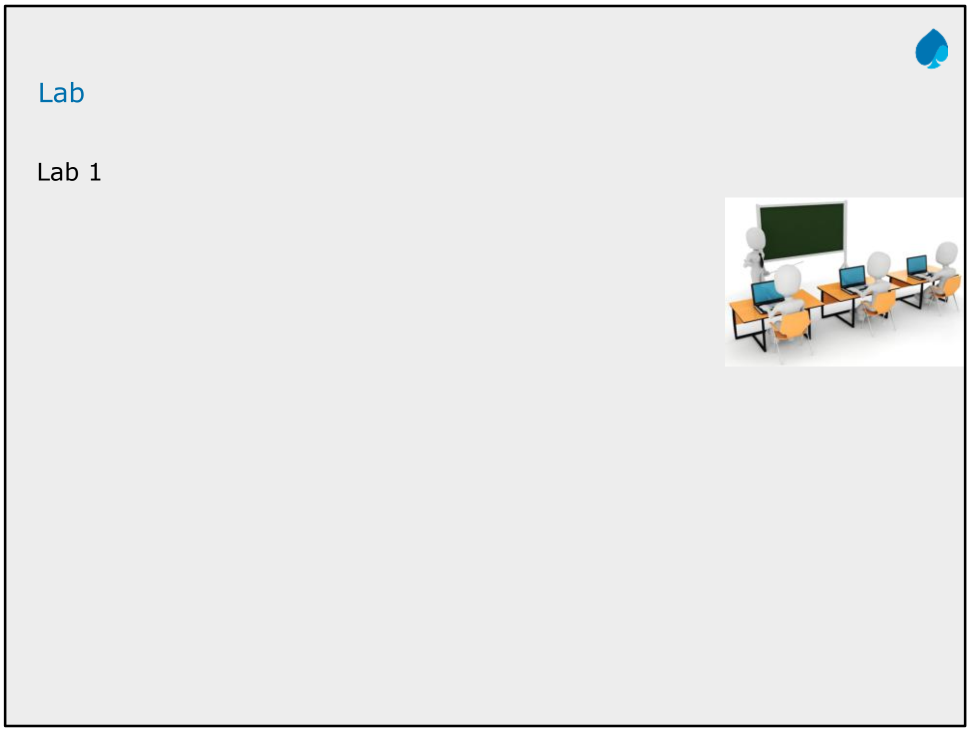
Demo

Demo1
Demo2
Demo3
Demo4
Demo5
Demo6
Demo7

Add the notes here.

Instructor Notes:

Add instructor notes here.

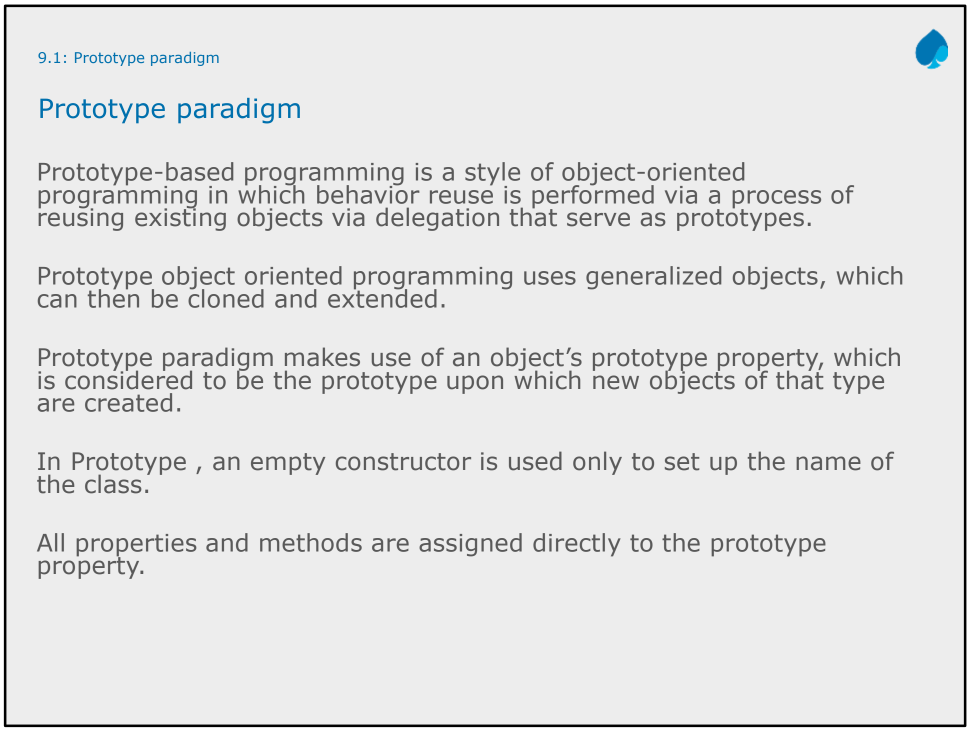


Lab

Lab 1

Add the notes here.

Instructor Notes:



9.1: 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.

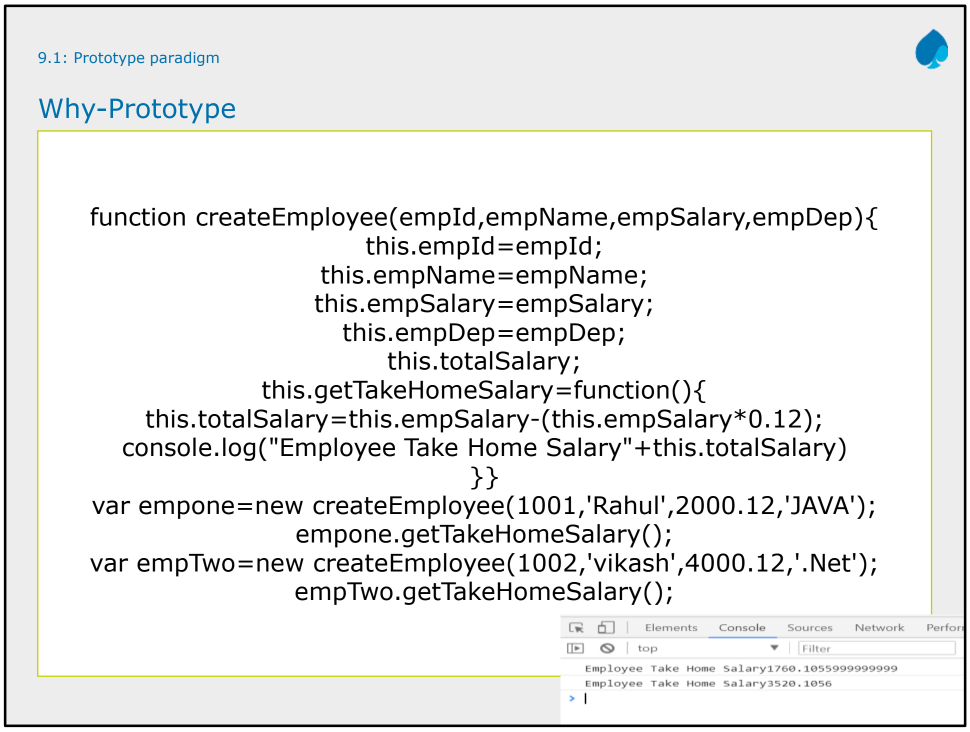
Prototype object oriented programming uses generalized objects, which can then be cloned and extended.

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:



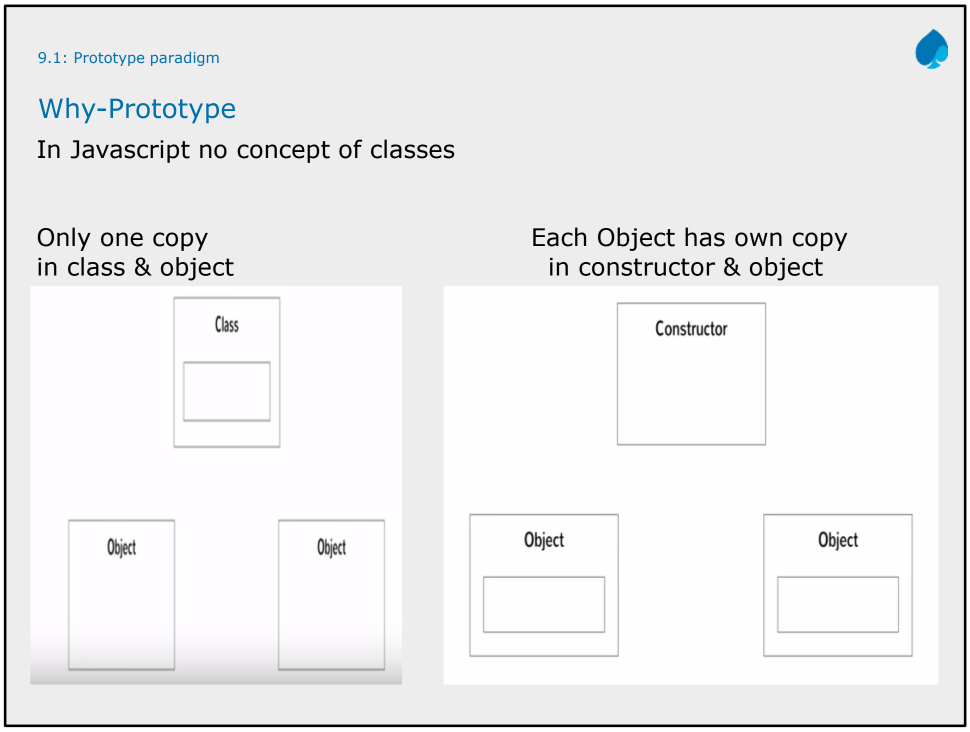
9.1: Prototype paradigm

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();
```

```
Employee Take Home Salary1760.1055999999999
Employee Take Home Salary3520.1056
>
```

Instructor Notes:



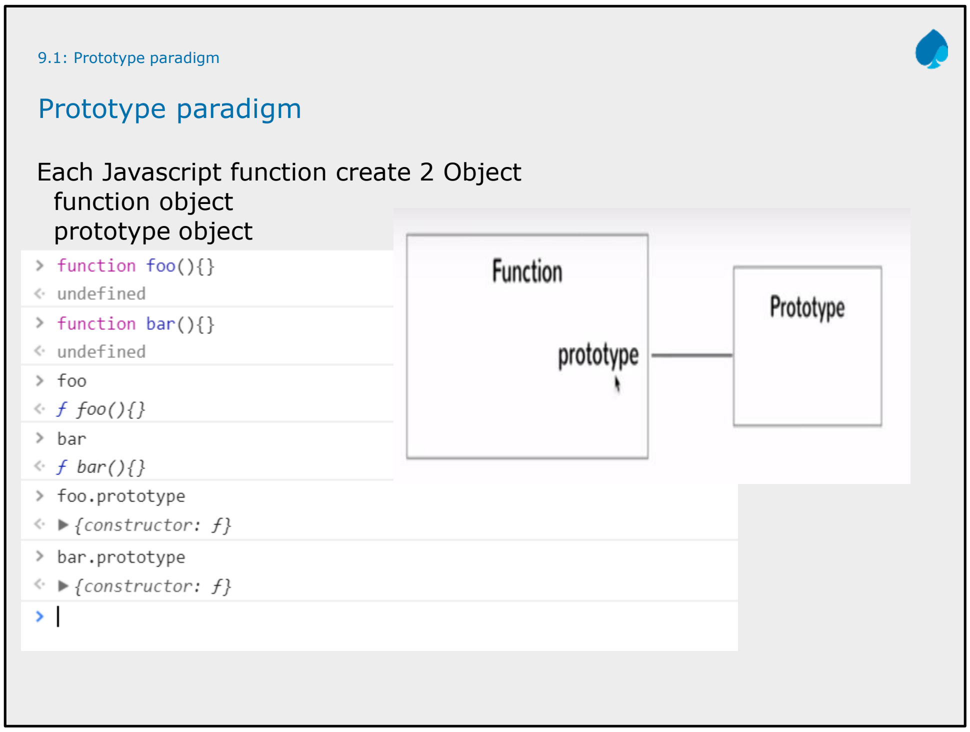
Why-Prototype

In Javascript no concept of classes

Only one copy
in class & object

Each Object has own copy
in constructor & object

Instructor Notes:



9.1: Prototype paradigm

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}
> |
```

Function
prototype
Prototype

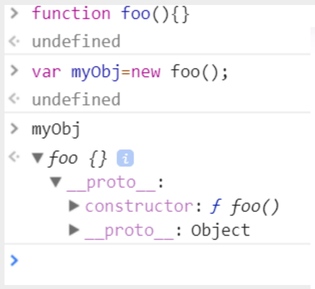
When you attempt to access a property or method of an object, JavaScript will first search on the object itself, and if it is not found, it will search the object's [[Prototype]]. If after consulting both the object and its [[Prototype]] still no match is found, JavaScript will check the prototype of 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:

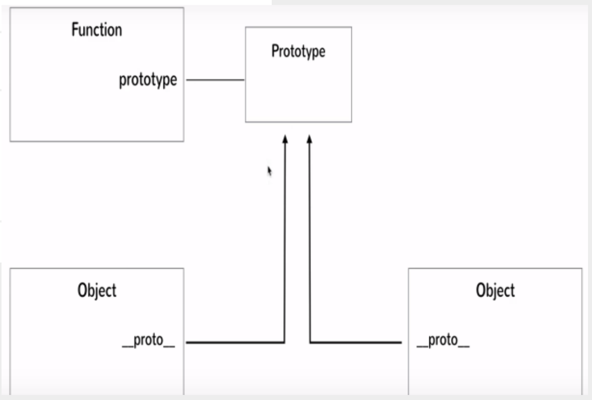
9.1: Prototype paradigm

Prototype paradigm

Now the any Objects will refer to _proto not function

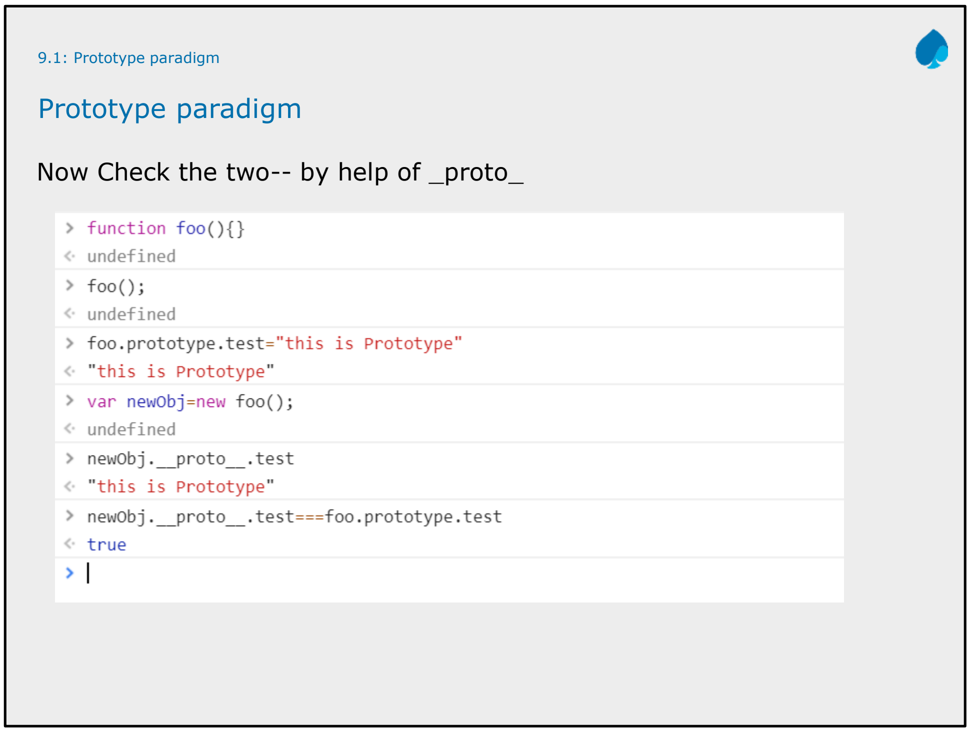


```
> function foo(){}
< undefined
> var myObj=new foo();
< undefined
> myObj
< ▼foo {}
    ▼__proto__:
      ▶constructor: ƒ foo()
      ▶__proto__: Object
>
```



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:



9.1: Prototype paradigm

Prototype paradigm

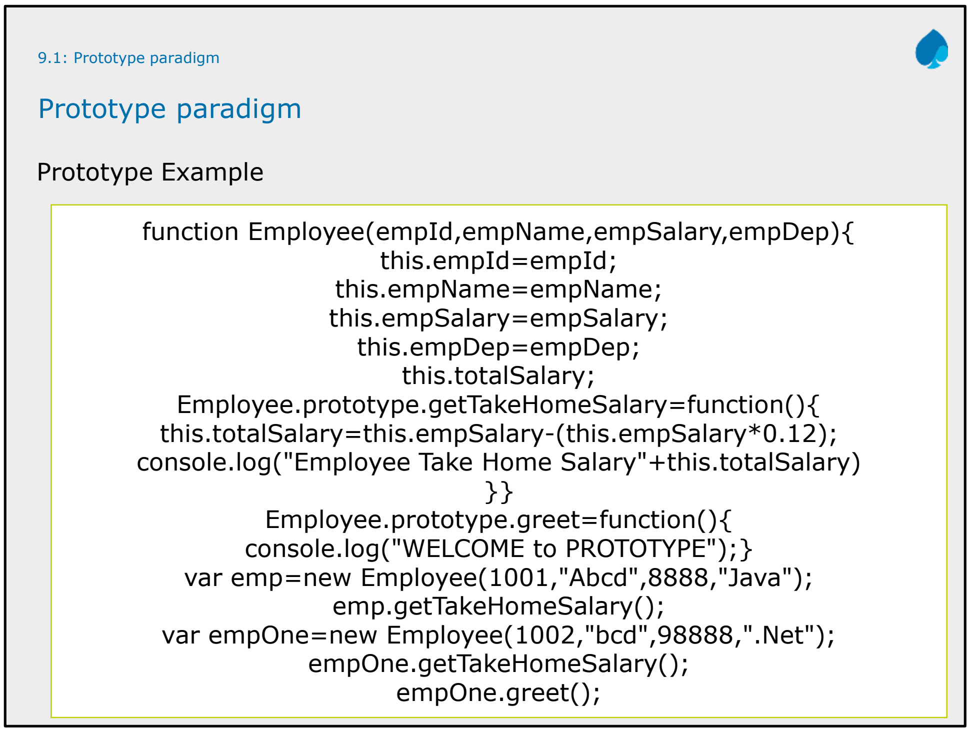
Now Check the two-- by help of _proto_

```
> 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:



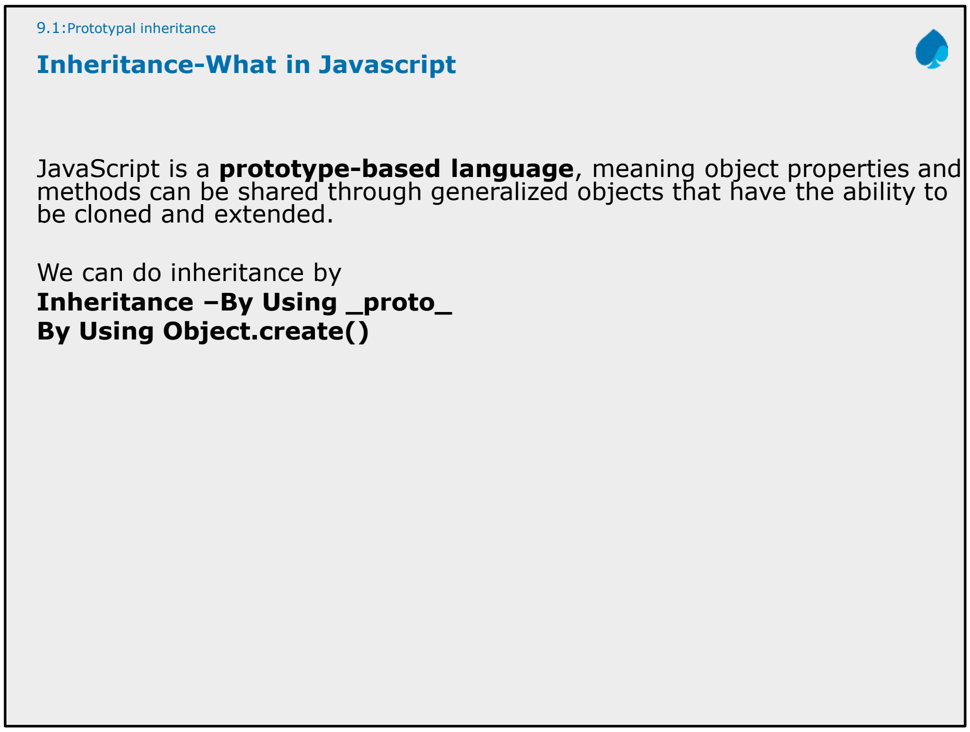
9.1: Prototype paradigm

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:



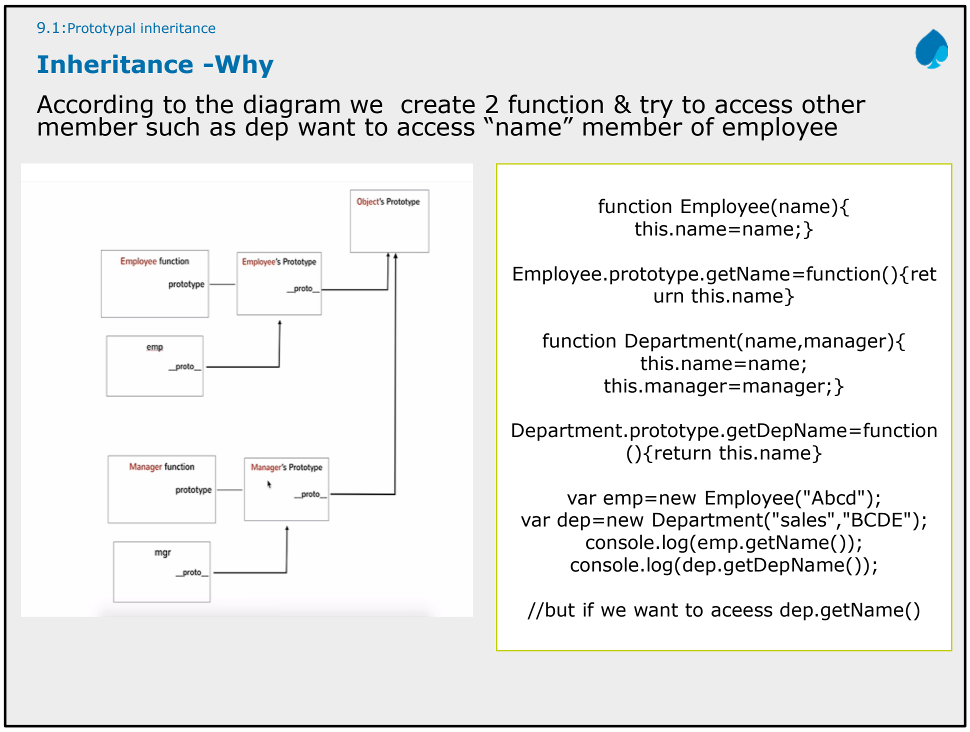
9.1:Prototypal inheritance

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()

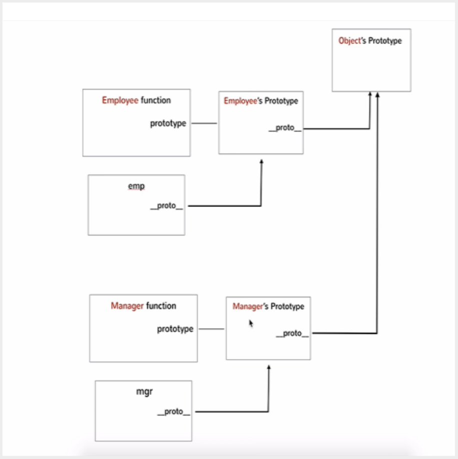
Instructor Notes:



9.1: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



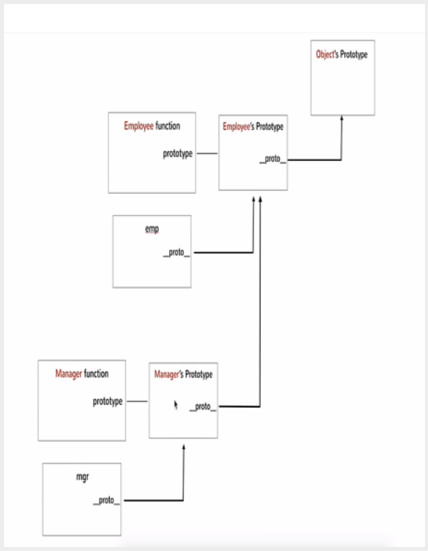
```
function Employee(name){
this.name=name;}

Employee.prototype.getName=function(){ret
urn 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:

9.2:Prototypal inheritance

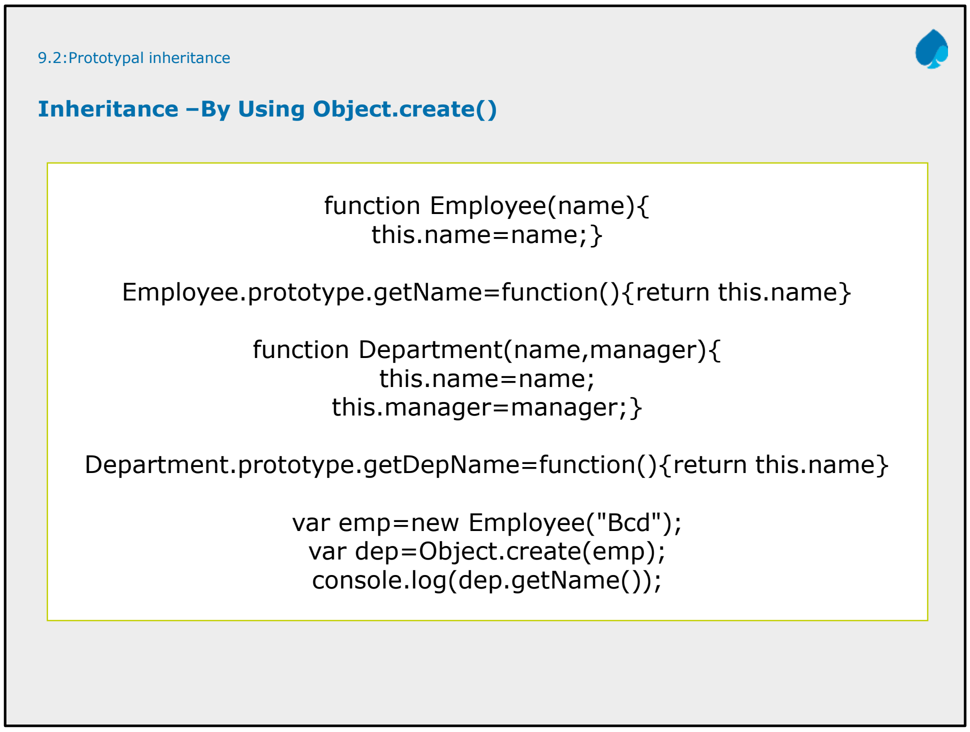
Inheritance –By Using _proto_



```
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(){retur
n 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:

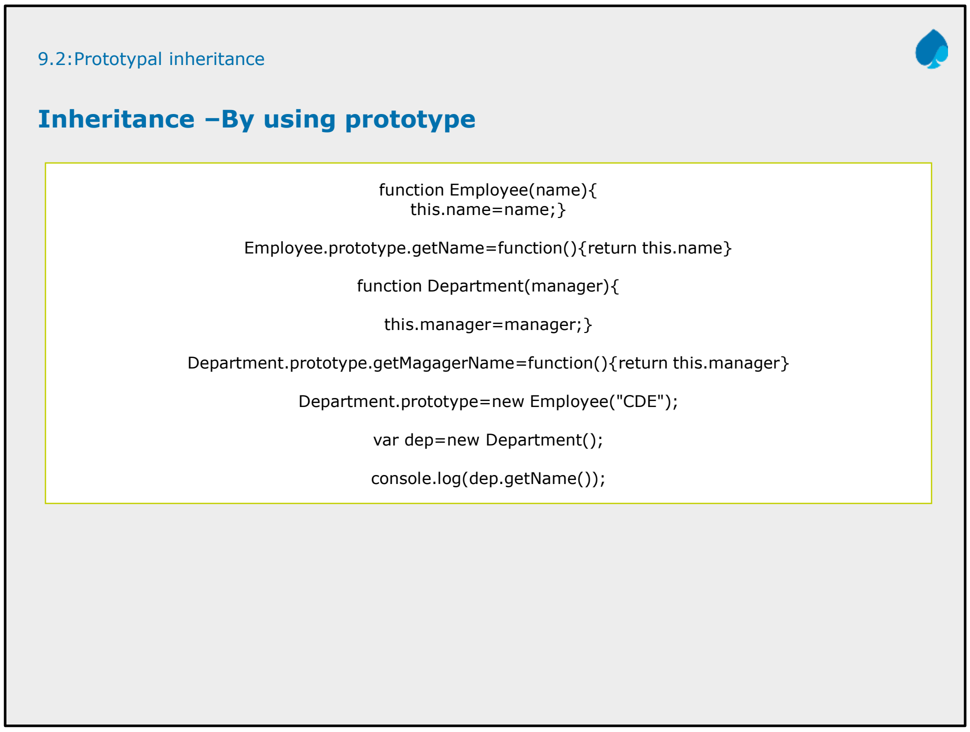


9.2:Prototypal inheritance

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:



9.2: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.

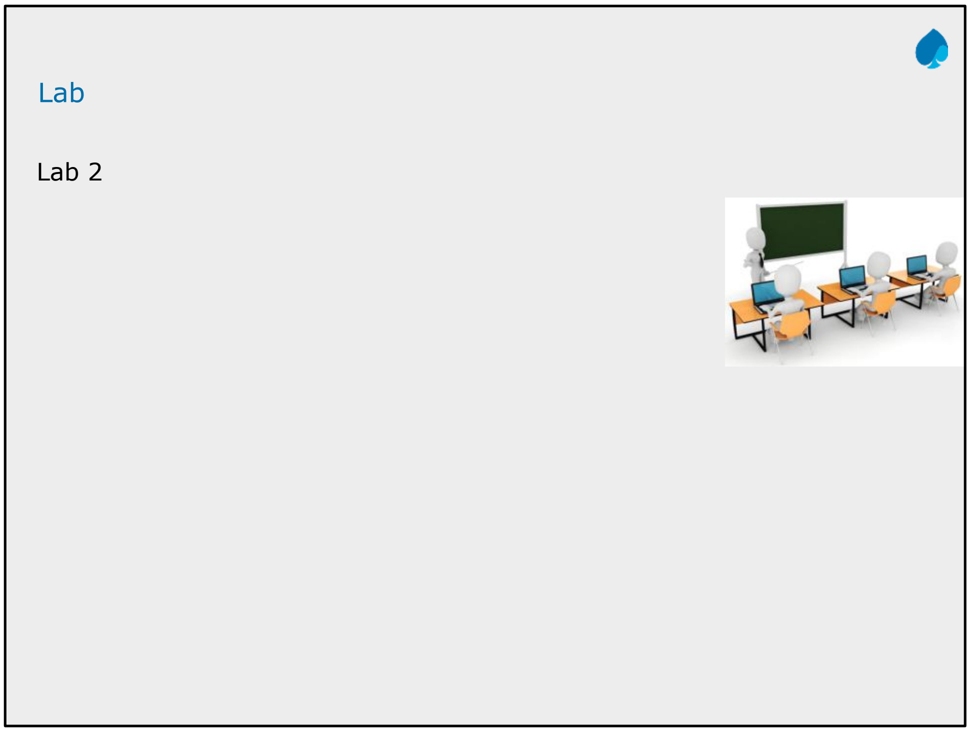
Demo

Demo1
Demo2
Demo3
Demo4

Add the notes here.

Instructor Notes:

Add instructor notes here.

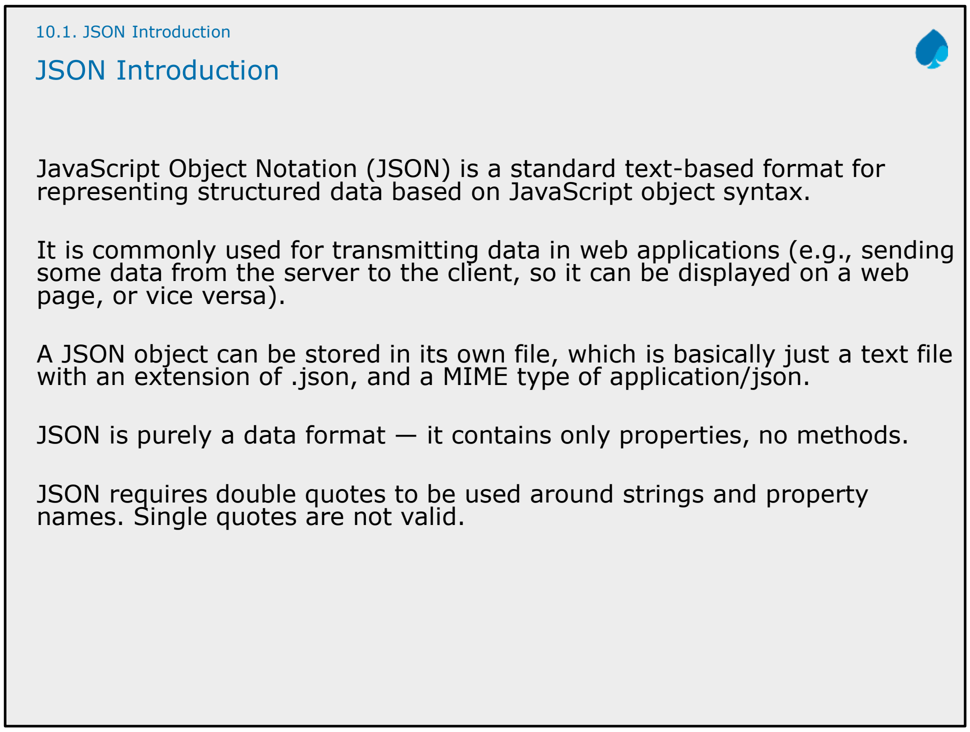


Lab

Lab 2

Add the notes here.

Instructor Notes:



10.1. JSON Introduction

JSON Introduction

JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.

It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

A JSON object can be stored in its own file, which is basically just a text file with an extension of .json, and a MIME type of application/json.

JSON is purely a data format — it contains only properties, no methods.

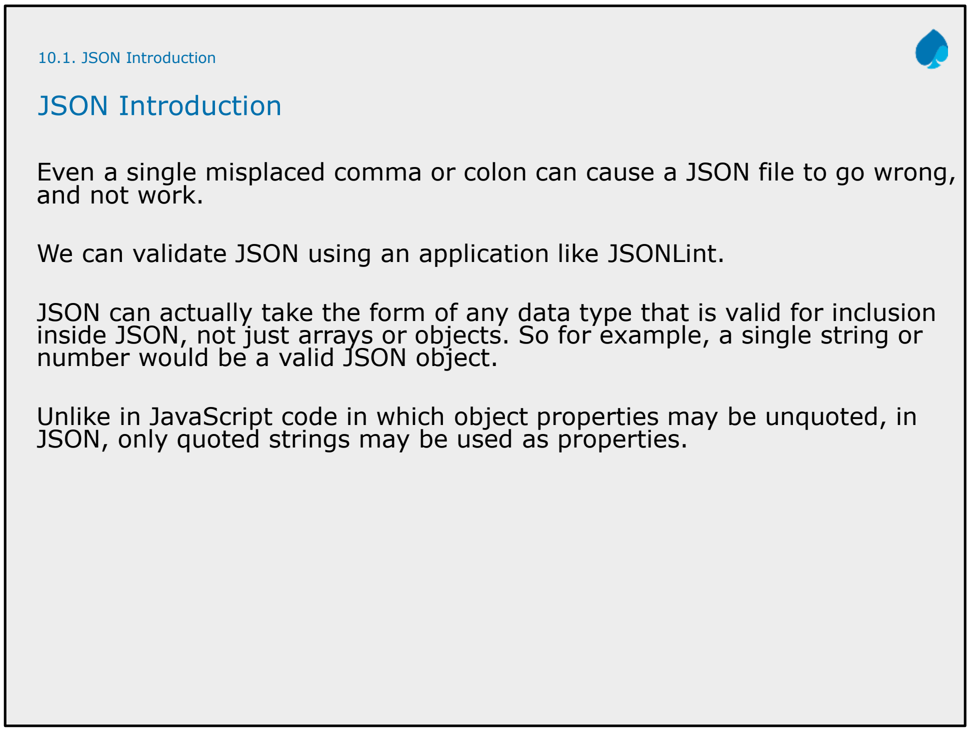
JSON requires double quotes to be used around strings and property names. Single quotes are not valid.

Ecma International is an industry association founded in 1961 and dedicated to the standardization of Information and Communication Technology (ICT) and Consumer Electronics

ECMAScript is the scripting language standardized by Ecma International in the ECMA-262 specification and ISO/IEC 16262. The language is widely used for client-side scripting on the web, in the form of several well-known implementations such as JavaScript, JScript and ActionScript.

Terminology
Namespace A container which lets developers bundle all functionality under a unique, application-specific name.Class Defines the object's characteristics. A class is a template definition of an object's properties and methods.Object An instance of a class.Property An object characteristic, such as color.Method An object capability, such as walk. It is a subroutine or function associated with a class.Constructor A method called at the moment an object is instantiated. It usually has the same name as the class containing it.Inheritance A class can inherit characteristics from another class.Encapsulation A method of bundling the data and methods that use the data.Abstraction The conjunction of an object's complex inheritance, methods, and properties must adequately reflect a reality model.Polymorphism Poly means "*many*" and morphism means "*forms*". Different classes might define the same method or property.

Instructor Notes:



10.1. JSON Introduction

JSON Introduction

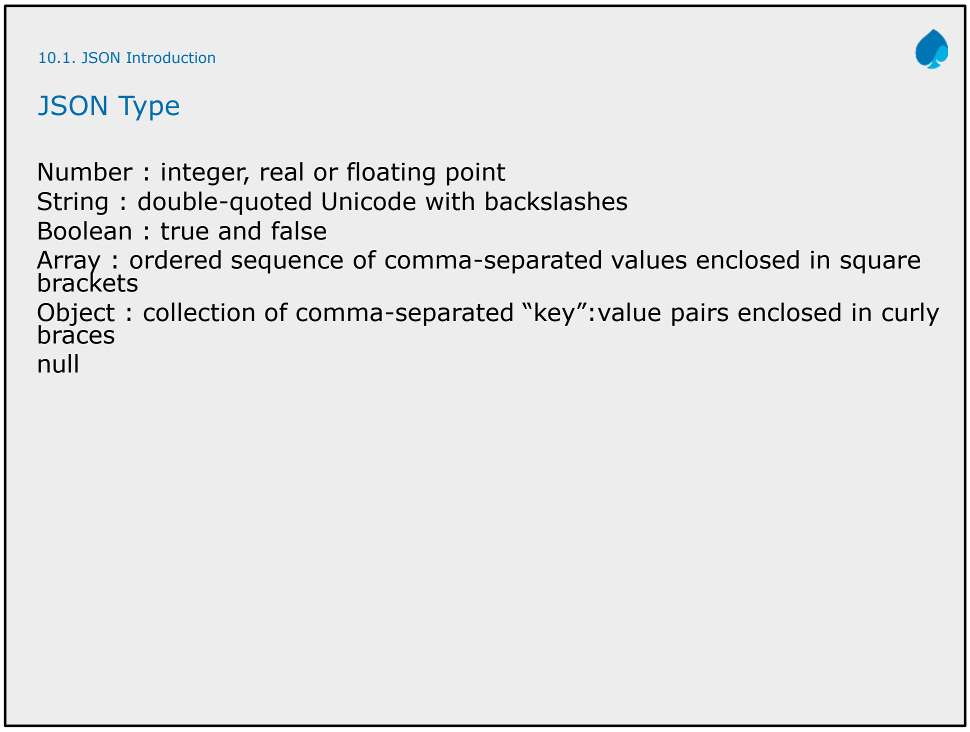
Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work.

We can validate JSON using an application like JSONLint.

JSON can actually take the form of any data type that is valid for inclusion inside JSON, not just arrays or objects. So for example, a single string or number would be a valid JSON object.

Unlike in JavaScript code in which object properties may be unquoted, in JSON, only quoted strings may be used as properties.

Instructor Notes:

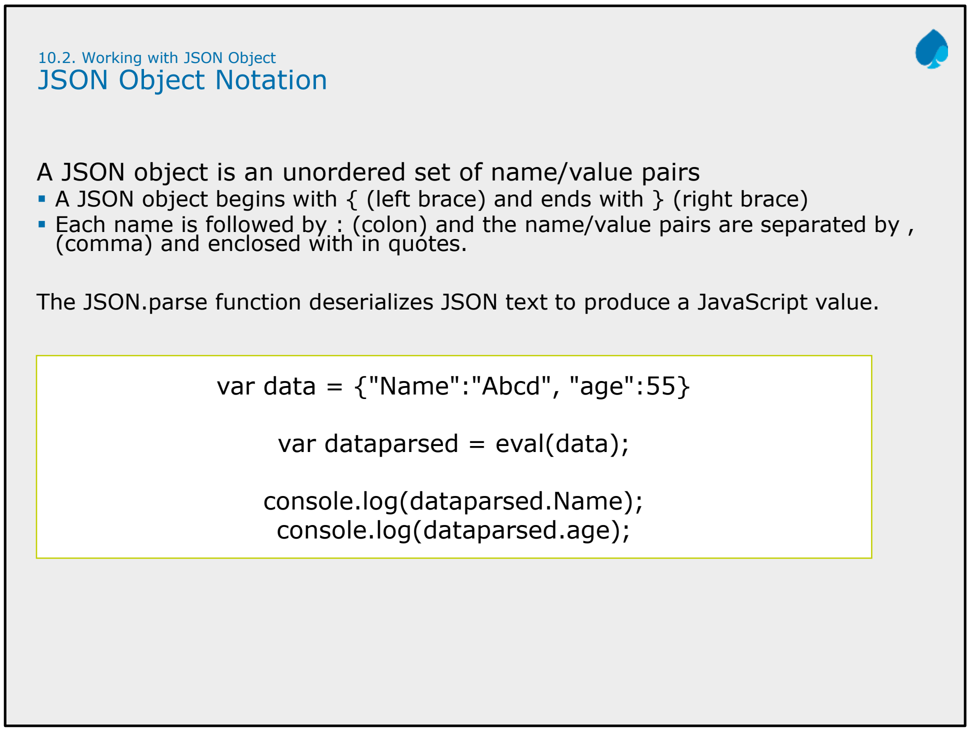


10.1. JSON Introduction

JSON Type

Number : integer, real or floating point
String : double-quoted Unicode with backslashes
Boolean : true and false
Array : ordered sequence of comma-separated values enclosed in square brackets
Object : collection of comma-separated “key”:value pairs enclosed in curly braces
null

Instructor Notes:



10.2. Working with JSON Object

JSON Object Notation

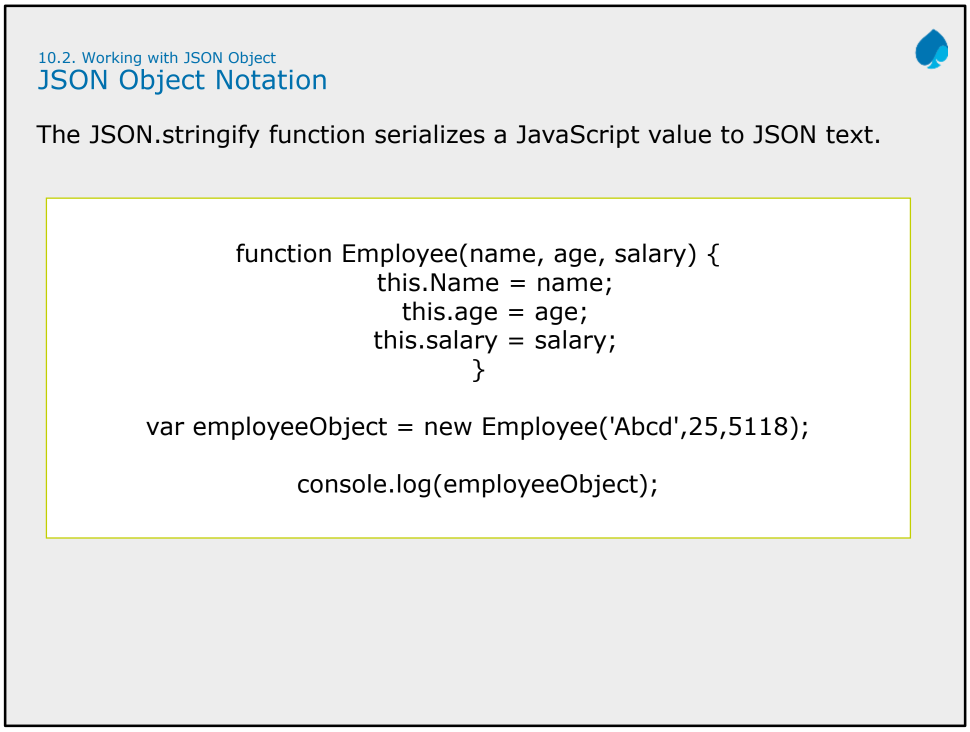
A JSON object is an unordered set of name/value pairs

- A JSON object begins with { (left brace) and ends with } (right brace)
- Each name is followed by : (colon) and the name/value pairs are separated by , (comma) and enclosed with in quotes.

The JSON.parse function deserializes JSON text to produce a JavaScript value.

```
var data = {"Name":"Abcd", "age":55}

var dataparsed = eval(data);

console.log(dataparsed.Name);
console.log(dataparsed.age);
```

Instructor Notes:



10.2. Working with JSON Object

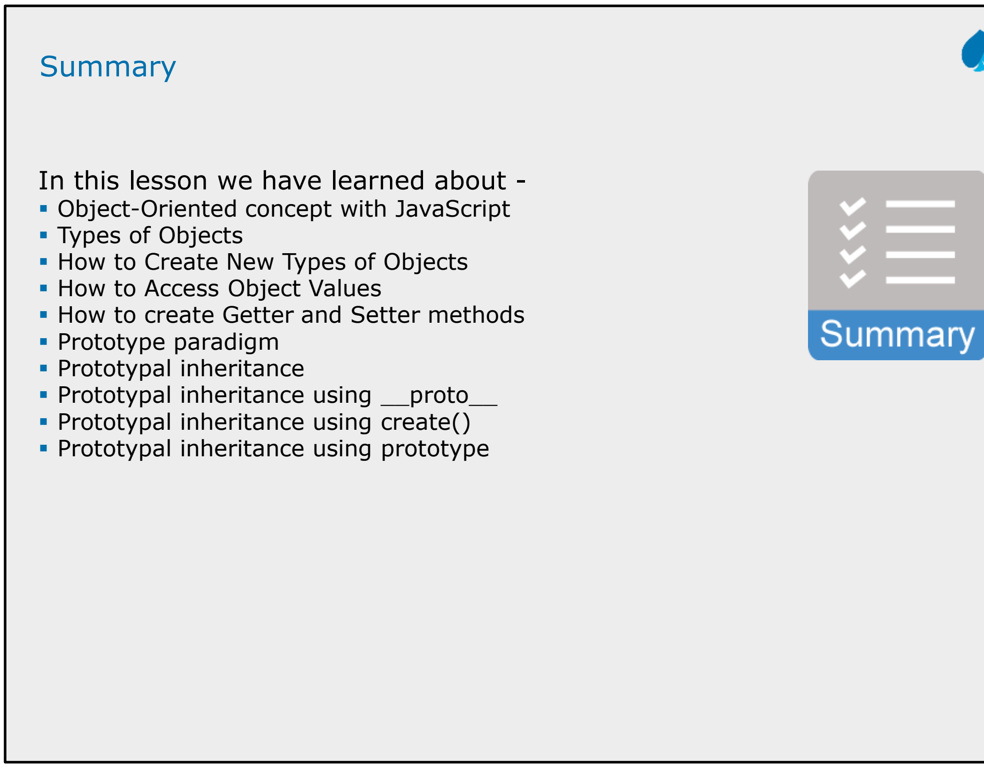
JSON Object Notation

The JSON.stringify function serializes a JavaScript value to JSON text.

```
function Employee(name, age, salary) {
this.Name = name;
this.age = age;
this.salary = salary;
}

var employeeObject = new Employee('Abcd',25,5118);

console.log(employeeObject);
```

Instructor Notes:

Add instructor notes here.



Summary

In this lesson we have learned about -
- Object-Oriented concept with JavaScript
- Types of Objects
- How to Create New Types of Objects
- How to Access Object Values
- How to create Getter and Setter methods
- Prototype paradigm
- Prototypal inheritance
- Prototypal inheritance using __proto__
- Prototypal inheritance using create()
- Prototypal inheritance using prototype

Summary