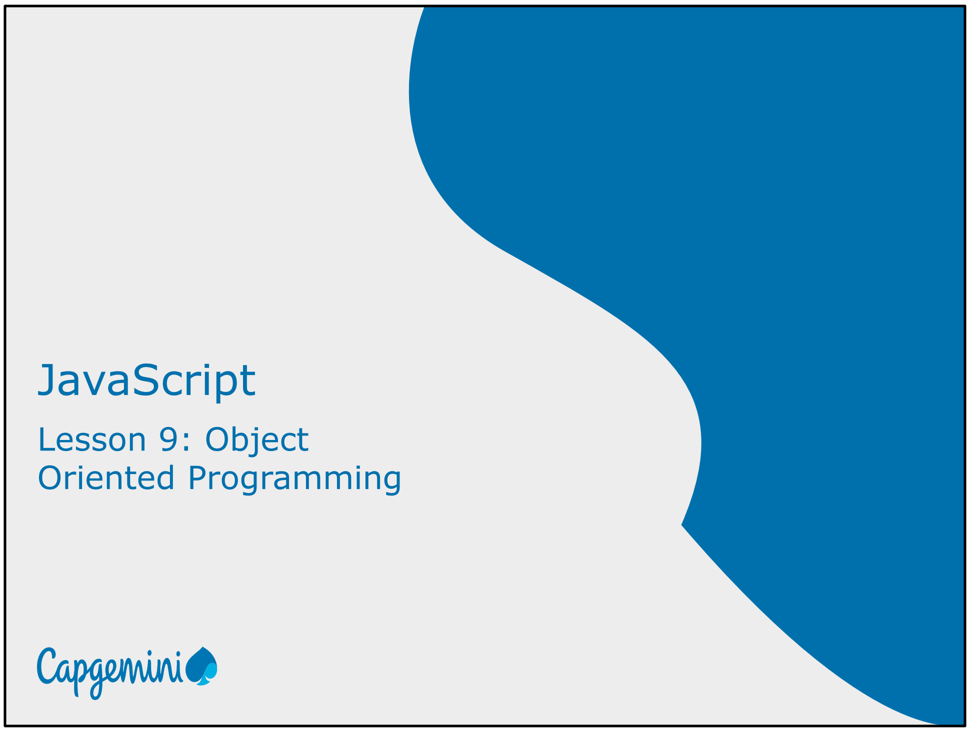
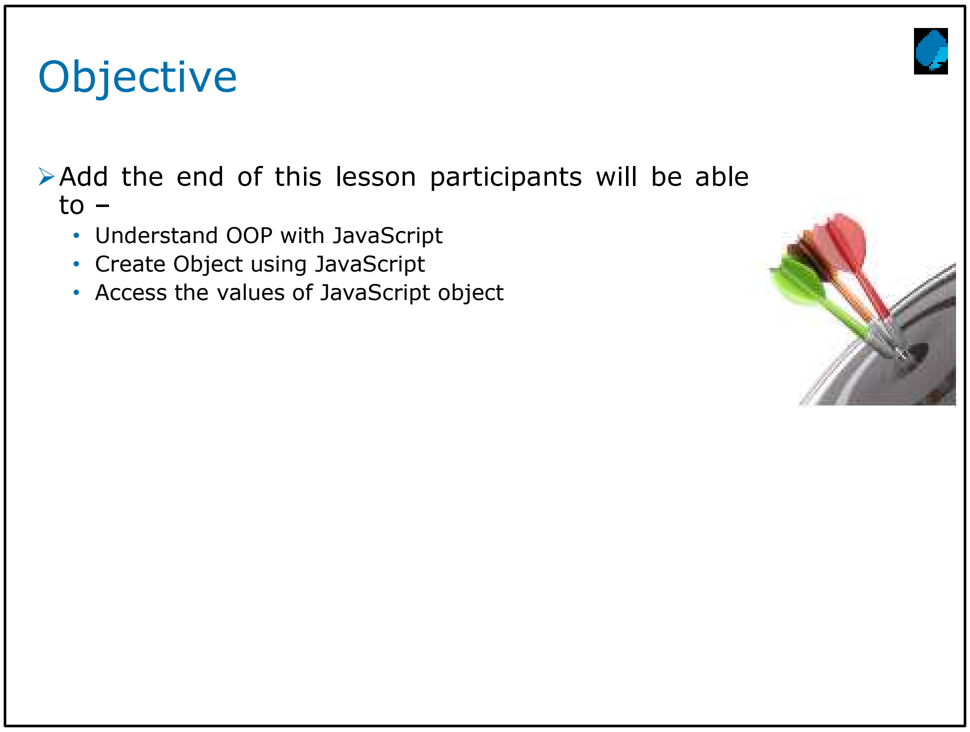


JavaScript

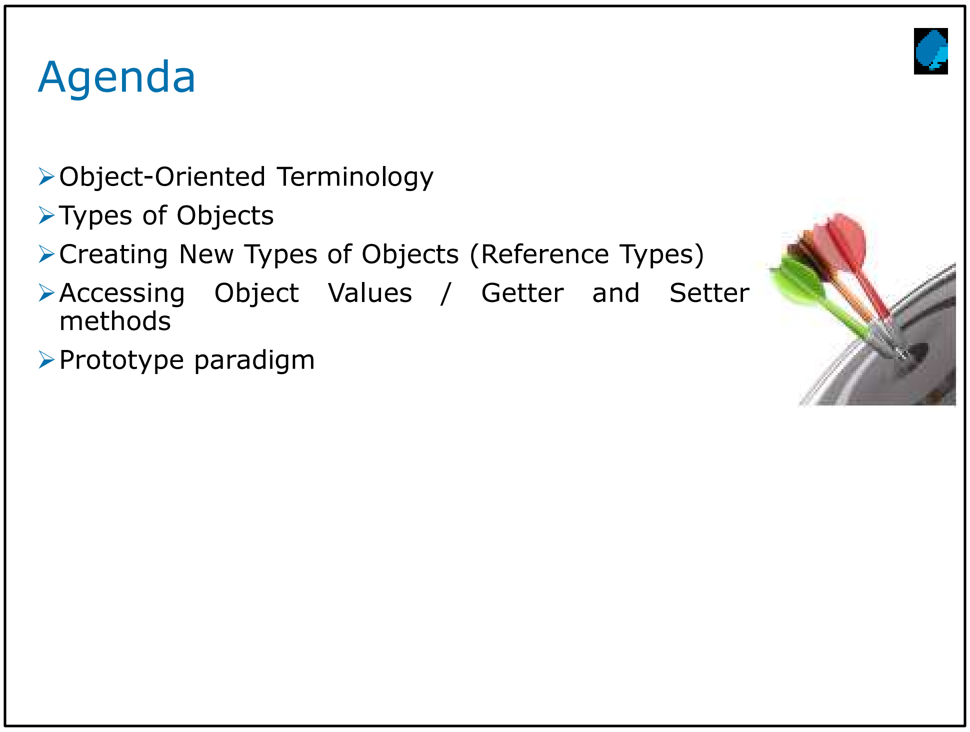
Lesson 9: Object Oriented Programming

Capgemini



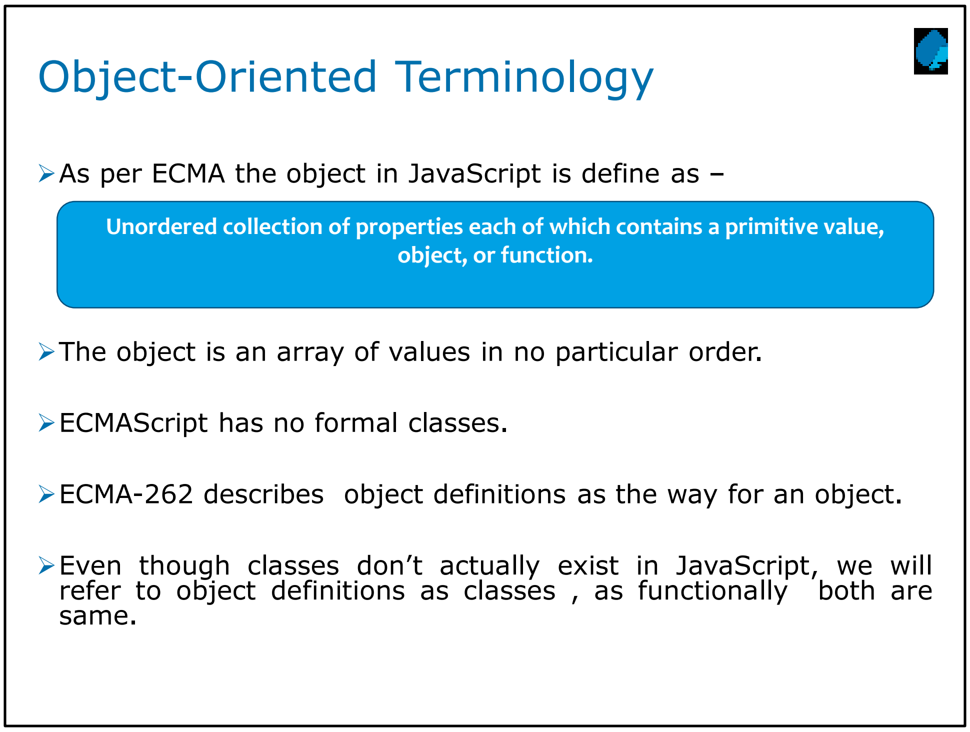
Objective

- Add the end of this lesson participants will be able to –
  - Understand OOP with JavaScript
  - Create Object using JavaScript
  - Access the values of JavaScript object



Agenda

- Object-Oriented Terminology
- Types of Objects
- Creating New Types of Objects (Reference Types)
- Accessing Object Values / Getter and Setter methods
- Prototype paradigm



Object-Oriented Terminology

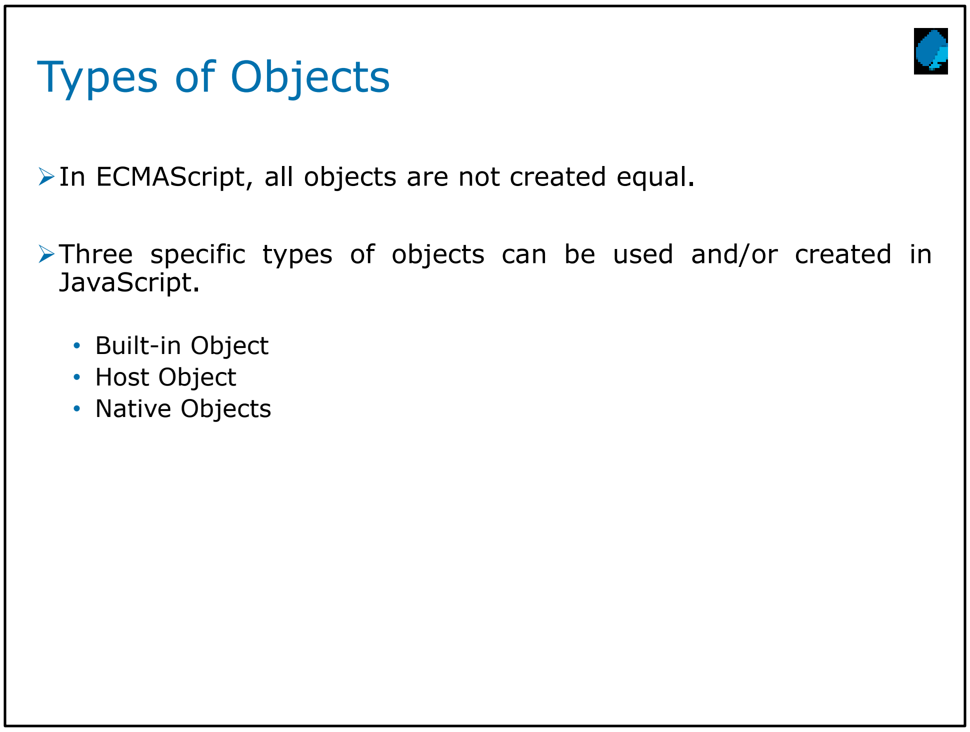
- As per ECMA the object in JavaScript is define as –

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

- The object is an array of values in no particular order.
- 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.



Types of Objects

- In ECMAScript, all objects are not created equal.
- Three specific types of objects can be used and/or created in JavaScript.
  - Built-in Object
  - Host Object
  - Native Objects

Built-in Object

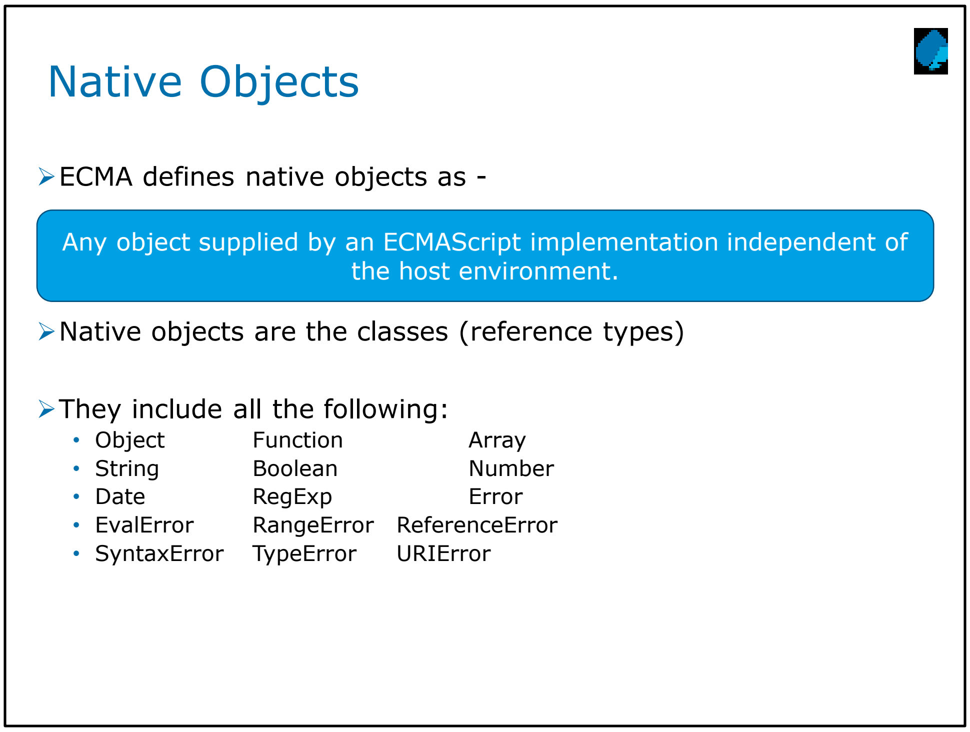
- Developer does not require to explicitly instantiate a built-in object; it is already instantiated.

- 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

Host Object

- Any object that is not native is considered to be a host object, which is defined as an object provided by the host environment of an ECMAScript implementation.
- All BOM and DOM objects are considered to be host objects



Native Objects

- ECMA defines native objects as -

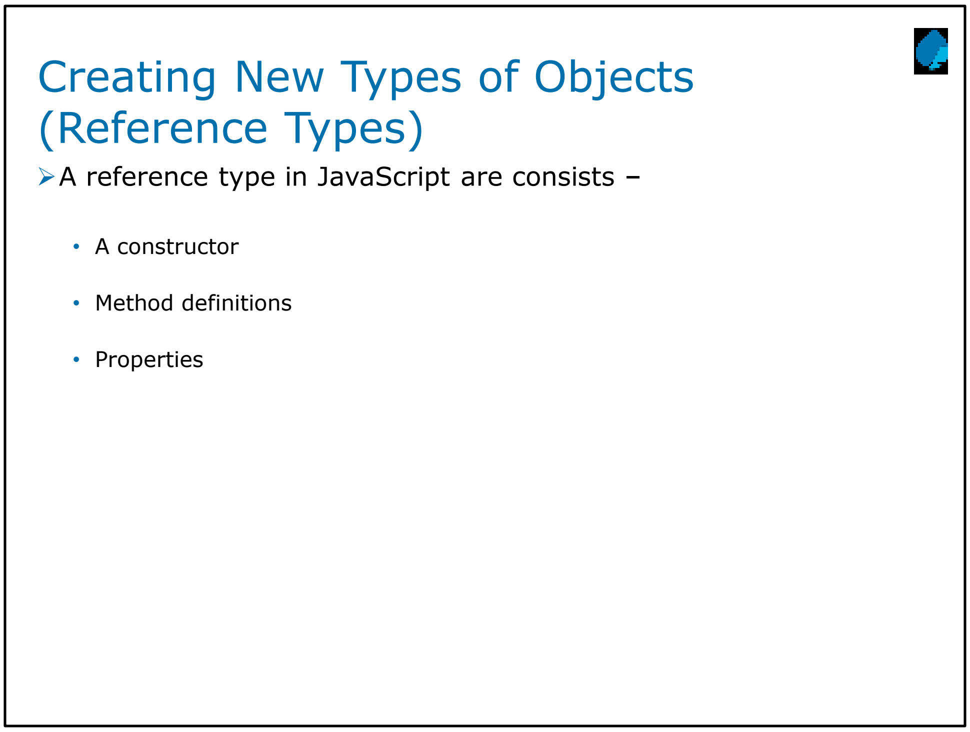
Any object supplied by an ECMAScript implementation independent of the host environment.

- Native objects are the classes (reference types)
- They include all the following:

Object	Function	Array
String	Boolean	Number
Date	RegExp	Error
EvalError	RangeError	ReferenceError
SyntaxError	TypeError	URIError

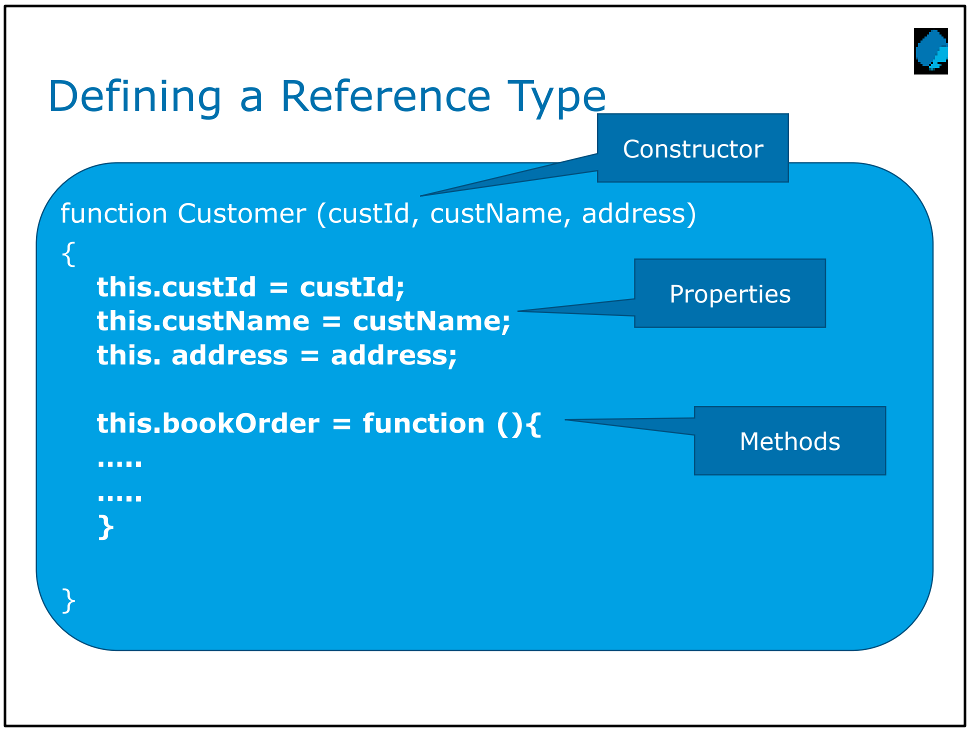
Creating New Types of Objects (Reference Types)

- JavaScript provides a number of built-in objects.
- JavaScript enables developer to create the templates for objects with specification.
- The key to this is JavaScript’s support for the definition of reference types.
- Reference types are essentially templates for an object.
- JavaScript has no formal class construct.
- Reference types and classes are the two terms use interchangeably.



Creating New Types of Objects (Reference Types)

- A reference type in JavaScript are consists –
  - A constructor
  - Method definitions
  - Properties


Constructor
Properties
Methods

Defining a Reference Type

```
function Customer (custId, custName, address)
{
   this.custId = custId;
   this.custName = custName;
   this. address = address;

   this.bookOrder = function (){
   .....
   .....
   }

}
```

Instantiation

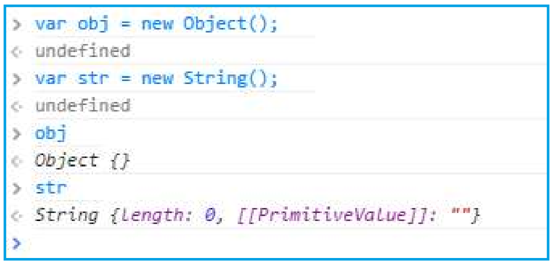
- Objects are created by using the new keyword followed by the name of the class to be instantiate -

```
var obj = new Object();

var str = new String();
```

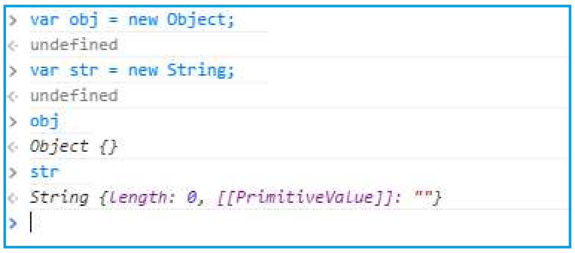
- The parentheses are optional, when the constructor doesn't require arguments

```
var obj = new Object;

var str = new String;
```

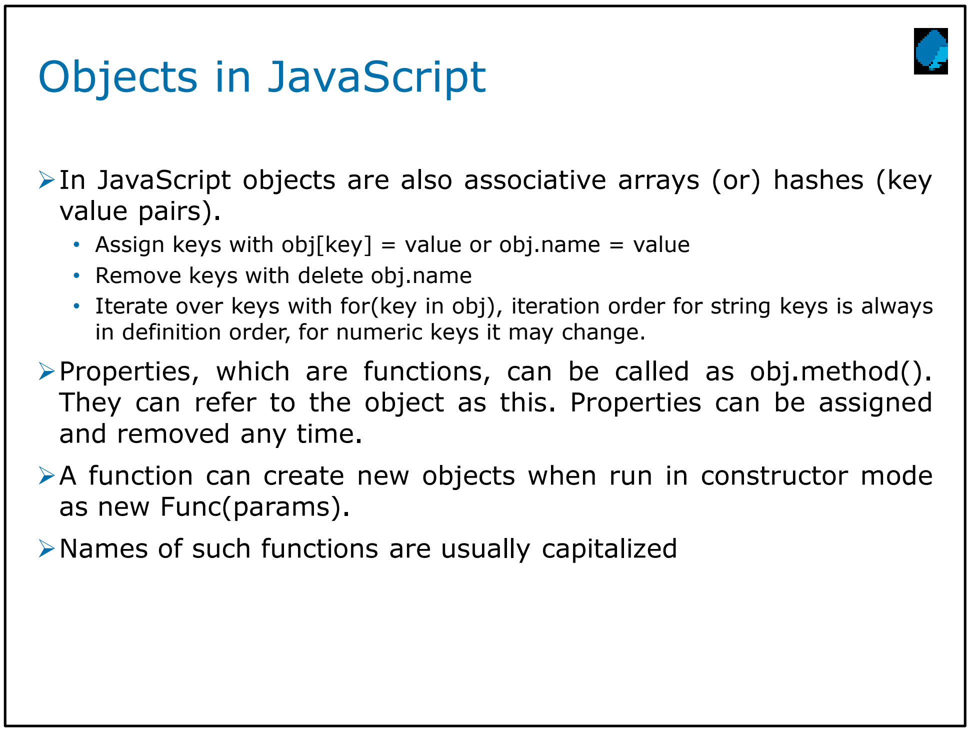
Declaration and instantiation (Contd.)



```
> var obj = new Object();
< undefined
> var str = new String();
< undefined
> obj
< Object {}
> str
< String {length: 0, [[PrimitiveValue]]: ""}
>
```



```
> var obj = new Object;
< undefined
> var str = new String;
< undefined
> obj
< Object {}
> str
< String {length: 0, [[PrimitiveValue]]: ""}
> |
```



Objects in JavaScript

- In JavaScript objects are also associative arrays (or) hashes (key value pairs).
  - Assign keys with obj[key] = value or obj.name = value
  - Remove keys with delete obj.name
  - Iterate over keys with for(key in obj), iteration order for string keys is always in definition order, for numeric keys it may change.
- Properties, which are functions, can be called as obj.method(). They can refer to the object as this. Properties can be assigned and removed any time.
- A function can create new objects when run in constructor mode as new Func(params).
- Names of such functions are usually capitalized

Add instructor notes here.

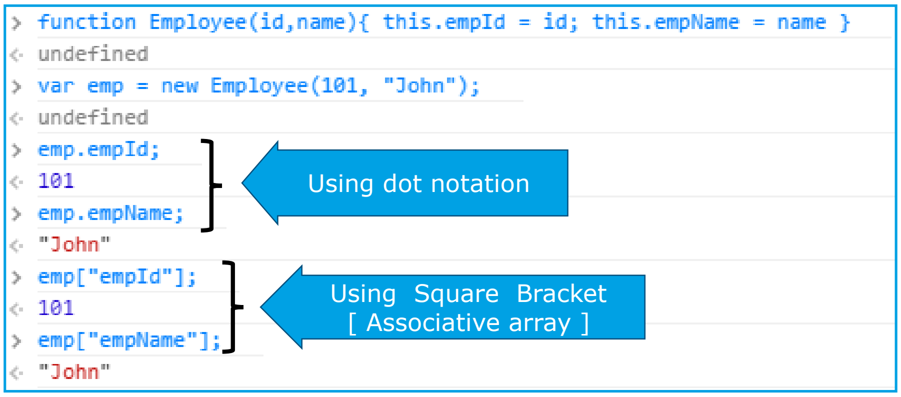
Creating objects - Using Constructors

- 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).
- Example for User defined Object creation :

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

var emp1=new
Employee(1001,”John");
```

- Object created using constructor will be reusable.

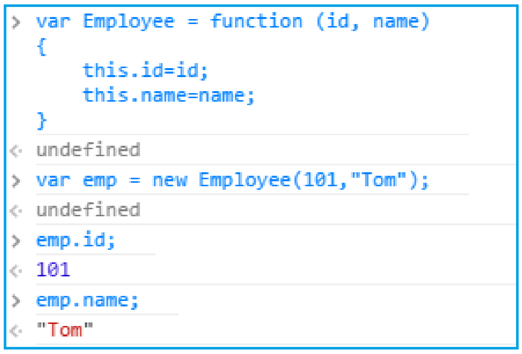
Accessing Object Values



```
> function Employee(id,name){ this.empId = id; this.empName = name }
<- undefined
> var emp = new Employee(101, "John");
<- undefined
> emp.empId;
<- 101
> emp.empName;
<- "John"
> emp["empId"];
<- 101
> emp["empName"];
<- "John"
```

Creating objects - Using Constructors

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

var emp1=new Employee(101, “Tom");
```



```
> var Employee = function (id, name)
  {
      this.id=id;
      this.name=name;
  }
<· undefined
> var emp = new Employee(101,"Tom");
<· undefined
> emp.id;
<· 101
> emp.name;
<· "Tom"
```

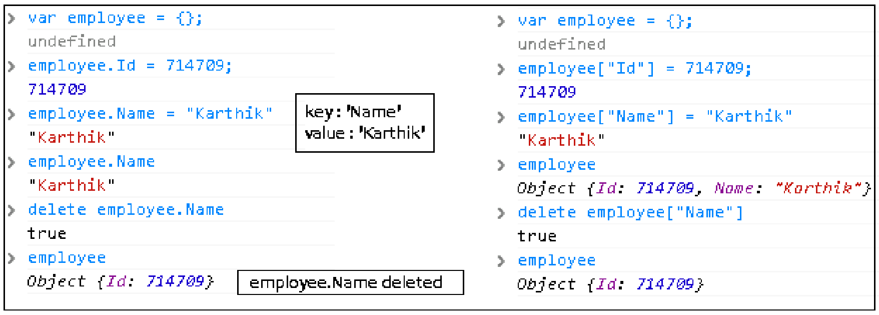
Add instructor notes here.

Creating objects

- An empty object can be creating using
  - obj = new Object(); (or) obj = { };
  - It stores values by key, with that we can assign or delete it using "dot notation" or "Square Brackets" (associative arrays).

using dot notation

using square brackets



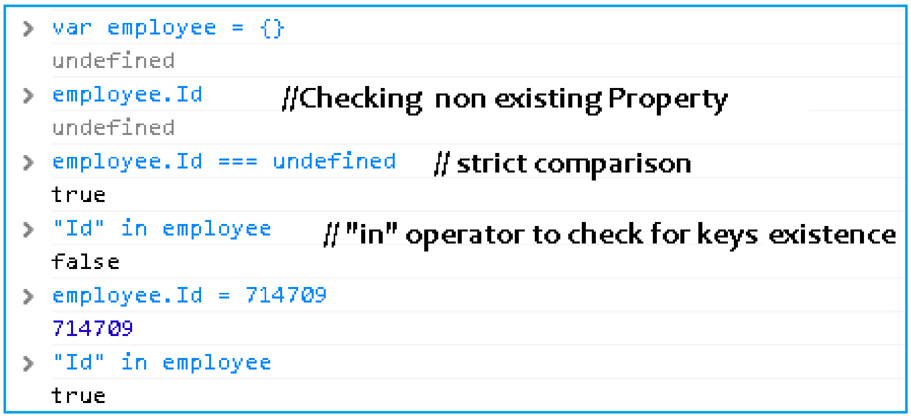
```
> var employee = {};
  undefined
> employee.Id = 714709;
  714709
> employee.Name = "Karthik"
  "Karthik"
> employee.Name
  "Karthik"
> delete employee.Name
  true
> employee
  Object {Id: 714709}
```

```
> var employee = {};
  undefined
> employee["Id"] = 714709;
  714709
> employee["Name"] = "Karthik"
  "Karthik"
> employee
  Object {Id: 714709, Name: "Karthik"}
> delete employee["Name"]
  true
> employee
  Object {Id: 714709}
```

Add instructor notes here.

Checking for non existing property in object

- If the property does not exist in the object , then undefined is returned
- To check whether key existence we can use in operator

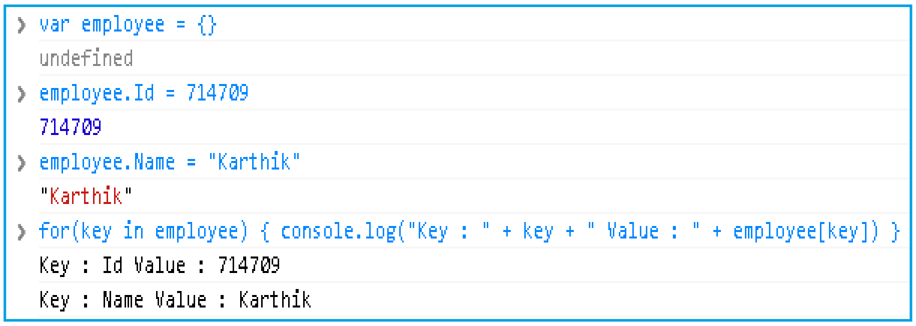


```
> var employee = {}
  undefined
> employee.Id              //Checking non existing Property
  undefined
> employee.Id === undefined     // strict comparison
  true
> "Id" in employee         // "in" operator to check for keys existence
  false
> employee.Id = 714709
  714709
> "Id" in employee
  true
```

Add instructor notes here.

Iterating over object keys

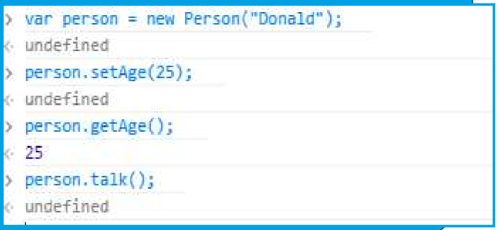
- We can iterate over keys using *for .. In*



```
❯ var employee = {}
  undefined
❯ employee.Id = 714709
  714709
❯ employee.Name = "Karthik"
  "Karthik"
❯ for(key in employee) { console.log("Key : " + key + " Value : " + employee[key]) }
  Key : Id Value : 714709
  Key : Name Value : Karthik
```

Getter/Setter Methods

```
function Person(name) {
        var age;                          //Define a private member
        this.name = name;                 //Define a public variable
        this.talk = function() {          // Define a method
            alert( "My name is " + this.name)
        }
        this.setAge = function(argAge){
            age = argAge;
        }
        this.getAge = function(){
            return age;
        }
}
```

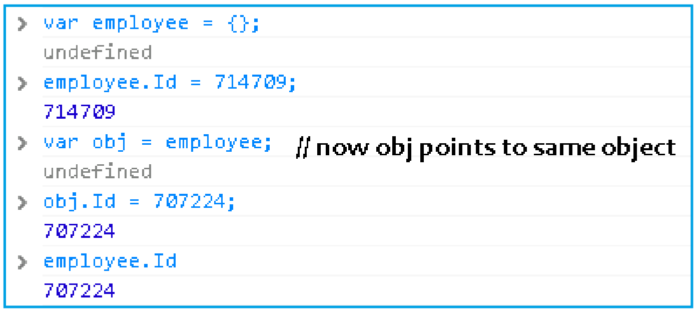


```
> var person = new Person("Donald");
< undefined
> person.setAge(25);
< undefined
> person.getAge();
< 25
> person.talk();
< undefined
```

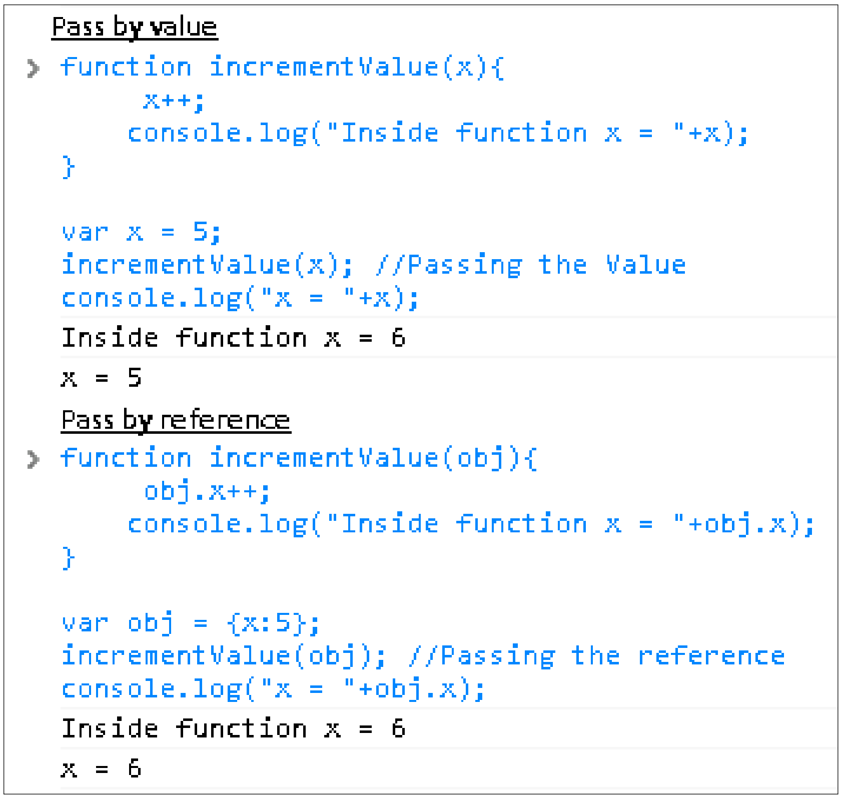
Add instructor notes here.

Object reference

- A variable which is assigned to object actually keeps reference to it.
- It acts like a pointer which points to the real data. Using reference variable we can change the properties of object.
- Variable is actually a reference, not a value when we pass an object to a function.



```
> var employee = {};
  undefined
> employee.Id = 714709;
  714709
> var obj = employee;    // now obj points to same object
  undefined
> obj.Id = 707224;
  707224
> employee.Id
  707224
```



Pass by value

```
> function incrementValue(x){
        x++;
      console.log("Inside function x = "+x);
  }

  var x = 5;
  incrementValue(x); //Passing the Value
  console.log("x = "+x);
  Inside function x = 6
  x = 5
```

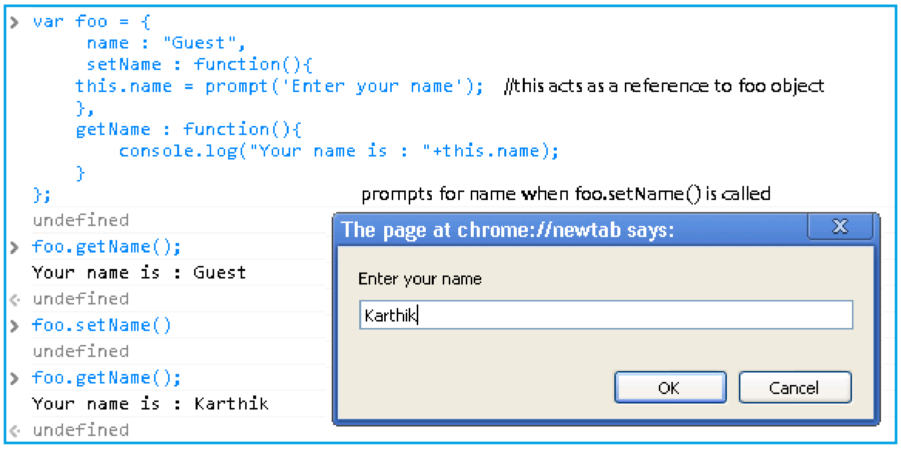
Pass by reference

```
> function incrementValue(obj){
        obj.x++;
      console.log("Inside function x = "+obj.x);
  }

  var obj = {x:5};
  incrementValue(obj); //Passing the reference
  console.log("x = "+obj.x);
  Inside function x = 6
  x = 6
```

Add instructor notes here.

this keyword

- When a function is called from the object, this becomes a reference to this object.



```
> var foo = {
       name : "Guest",
       setName : function(){
      this.name = prompt('Enter your name');   //this acts as a reference to foo object
      },
      getName : function(){
          console.log("Your name is : "+this.name);
      }
  };
  undefined
> foo.getName();
  Your name is : Guest
< undefined
> foo.setName()
  undefined
> foo.getName();
  Your name is : Karthik
< undefined
```

prompts for name when foo.setName() is called

The page at chrome://newtab says:

Enter your name

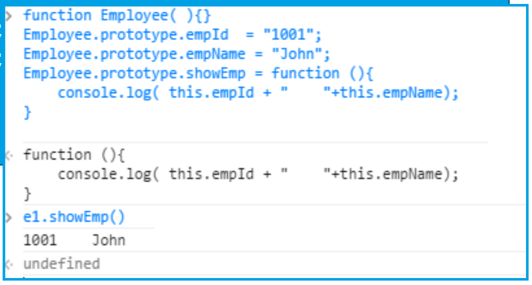
Karthik

Prototype paradigm

- 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.

Prototype paradigm

```
function Employee( ){       }
Employee.prototype.empId  = “1001”;
Employee.prototype.empName = “John”;
Employee.prototype.showEmp = function (){
    console.log( this.empId + “    “+this.empName);
}

var e1 = new Employee();
var e2 = new Employee();
```



```
> function Employee( ){}
  Employee.prototype.empId  = "1001";
  Employee.prototype.empName = "John";
  Employee.prototype.showEmp = function (){
      console.log( this.empId + "    "+this.empName);
  }
< function (){
      console.log( this.empId + "    "+this.empName);
  }
> e1.showEmp()
  1001    John
< undefined
```

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