Introduction To JavaScript

Authored and Presented by : Sushant B.
Extn : 7221, Email : sushantba@cybage.com

Agenda

- Introduction
- Functions
- Scopes
- Objects
- Prototype and Inheritance
- Closures

Introduction

WWW

- World Wide Web or simply ‘The Web’
- It is an open source information space
- Web resources are identified by URL’s
- Resources are interconnected by hypertext links
- Invented by Tim Berners-Lee in 1989
- A huge collection of web pages accessed via internet.

Web Pages

- May contain text, images, videos etc.
- Three essential technologies to develop webpages
  - HTML
  - CSS
  - JavaScript.

What is JavaScript

"JavaScript is a high level, dynamic, untyped and interpreted programming language."
– By Wikipedia

- Developed by *Brendan Eich* in *Netscape Navigator 2.0* in Sep 1995
- Standardized in the ECMAScript language specification
- Supported by all major browsers
- Runs by browser's JavaScript engine.

JavaScript Facts

JavaScript is

- Dynamically typed
- Object based
- Prototype based inheritance

JavaScript Syntax

- JavaScript syntax mostly similar to Java
- JavaScript is case sensitive.

ECMAScript Specification

- JavaScript is standardized at Ecma International
- ECMAScript is a standardized version of JavaScript

Browser engines implemented ECMAScript

- SpiderMonkey in Firefox
- v8 in Chrome
- Chakra in IE9.

JavaScript Editor

- Simple text editor such as Notepad++ is enough
- Brackets, sublime etc.
- Online editors – jsbin, jsfiddle etc.
- Browser's developer tool.

Example

- Example-1

Data Types

- In JavaScript values are typed, not variables
- Built-in types
  - String
  - Number
  - Boolean
  - Null and undefined
  - Object
- Arrays and Functions – Specialized version of object type.

Exercise

- Example-2

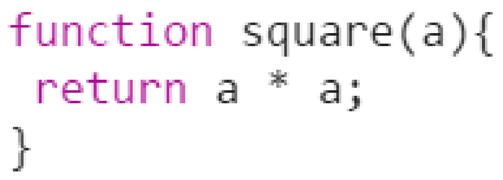
Functions

Functions

- Functions are subtype of object
- A function is a value
  - Can be assigned to a variable
  - Can be passed as parameter to other function
  - Can be returned from other function
- Function expressions
  - Anonymous functions
  - Named functions.

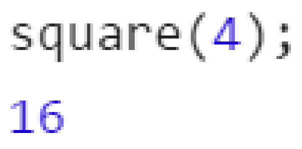
Function Declaration

- Declaring a function



```
function square(a){
 return a * a;
}
```

- Calling a function



```
square(4);
16
```

Exercise

Example-3

Function Expression

- Named function

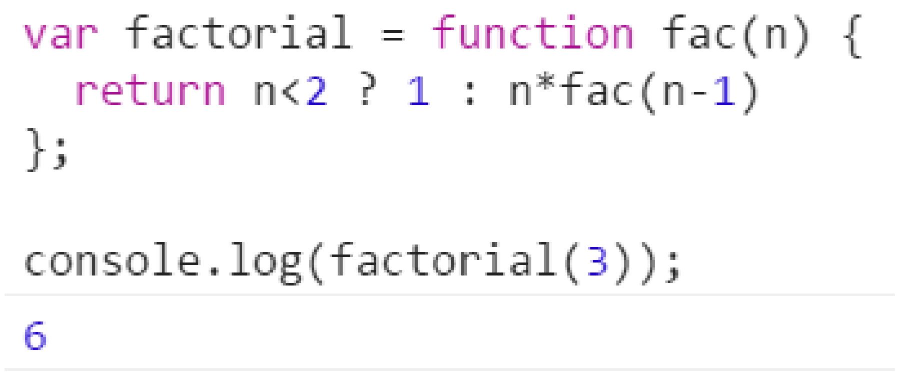
```
var x = function square(a){
 return a * a
};

x(3);
```

- Anonymous function

```
var x = function(a){
 return a * a
};

x(6);
```

Function Expression

- A function calls itself – named function



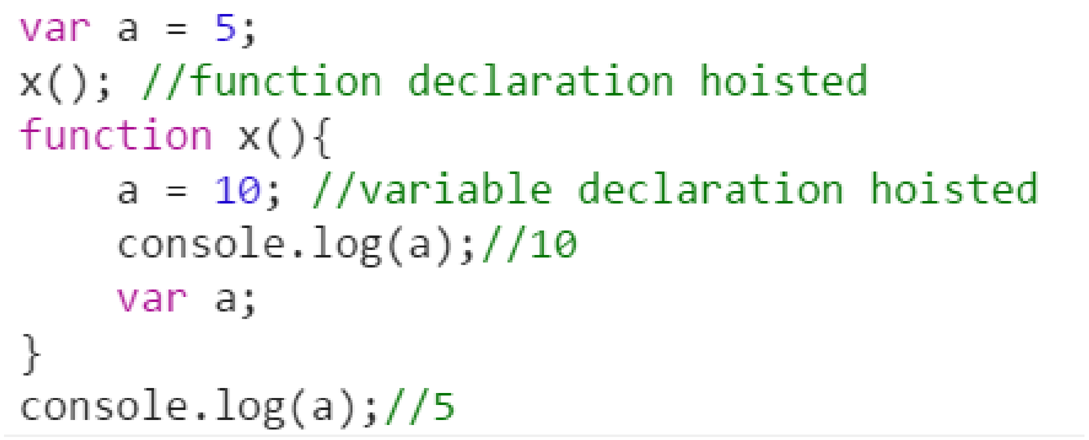
```
var factorial = function fac(n) {
  return n<2 ? 1 : n*fac(n-1)
};

console.log(factorial(3));
6
```

Exercise

Example-4

Hoisting

- Variable and function hoisting



```
var a = 5;
x(); //function declaration hoisted
function x(){
    a = 10; //variable declaration hoisted
    console.log(a);//10
    var a;
}
console.log(a);//5
```

Cybage® Delivering Value Scientifically

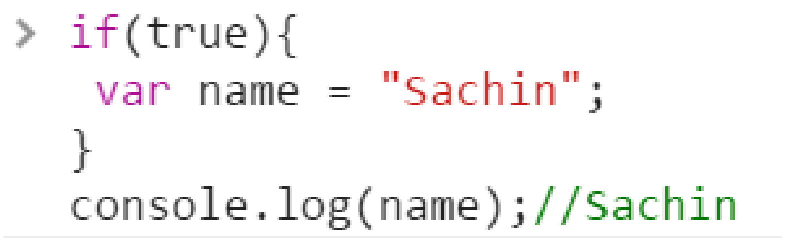
Exercise

[Example-5]

Scope – No Block-Level Scope

- JavaScript scopes are only
- Function level scope
- Global level scope

- Variables declared inside blocks are accessible outside



```
> if(true){
   var name = "Sachin";
  }
  console.log(name);//Sachin
```

Scope – Global Vs. Local

- When Global?
- Variables declared outside all functions are global

- When Local?
- Variables declared inside or in the parameter are local to function.

Exercise

- Example-6

Scope – Nested Scope

- Inner functions are private to outer functions
- An inner function can access a variable of outer function

Exercise

- Example-7

Objects

Objects

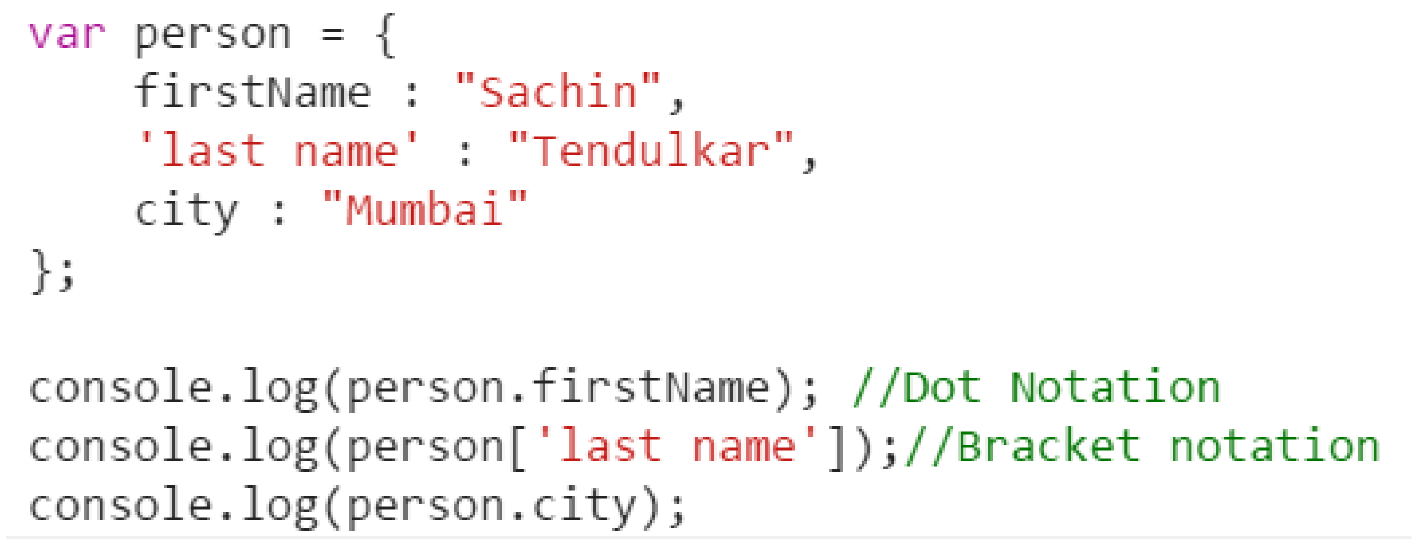
- An object represents entity
- Can contain properties
- An object can be created using
  - Object initializer syntax (also known as literal notation)
  - Constructor function
  - Using Object.create() method

Properties

- An object contains properties
- Properties are variables attached to objects
- Properties can be accessed using
  - Dot notation
  - Bracket notation

Object Initializers

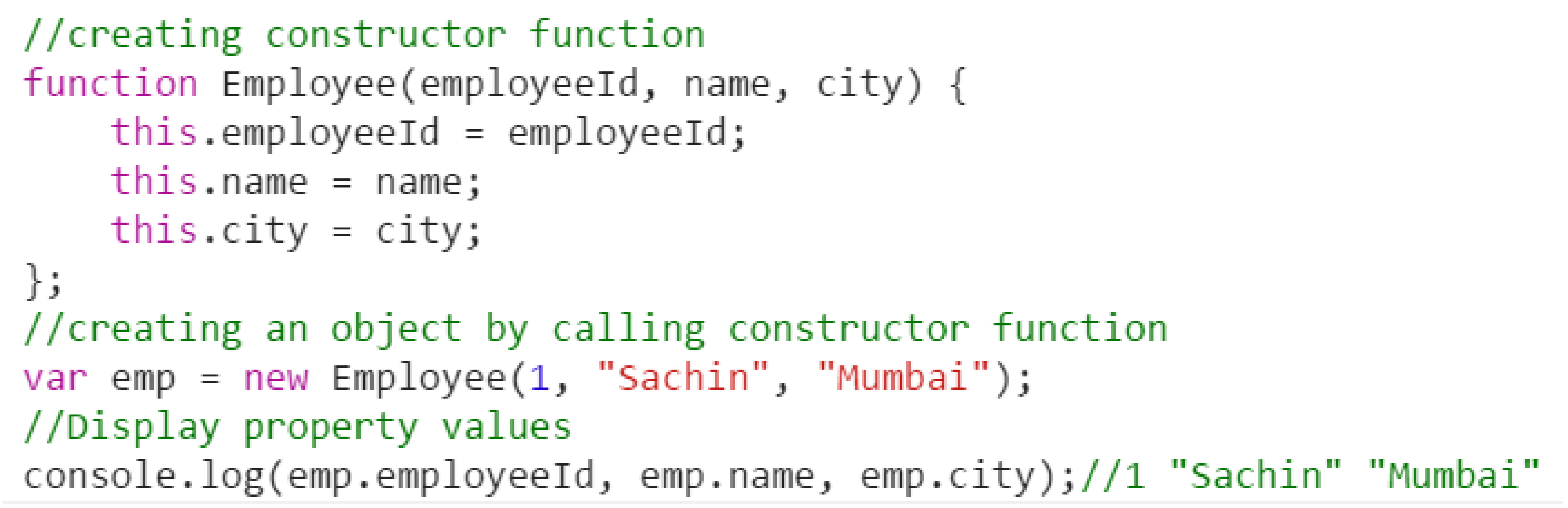
- Creating object with Object Initializers Syntax



```
var person = {
    firstName : "Sachin",
    'last name' : "Tendulkar",
    city : "Mumbai"
};

console.log(person.firstName); //Dot Notation
console.log(person['last name']);//Bracket notation
console.log(person.city);
```

Constructor Function

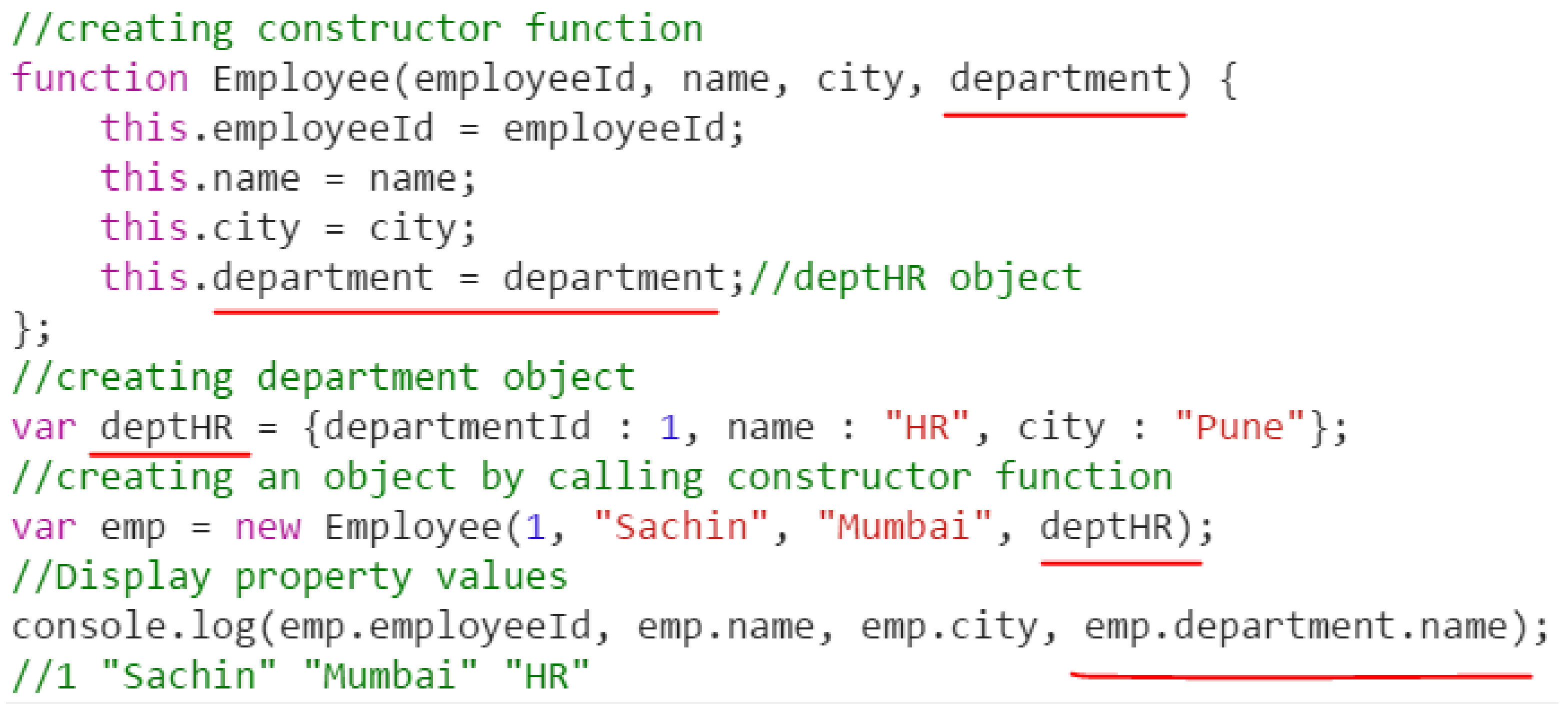
- Creating object with constructor function
- Constructor function convention – initial letter is capital.



```
//creating constructor function
function Employee(employeeId, name, city) {
    this.employeeId = employeeId;
    this.name = name;
    this.city = city;
};
//creating an object by calling constructor function
var emp = new Employee(1, "Sachin", "Mumbai");
//Display property values
console.log(emp.employeeId, emp.name, emp.city);//1 "Sachin" "Mumbai"
```

A property can be object

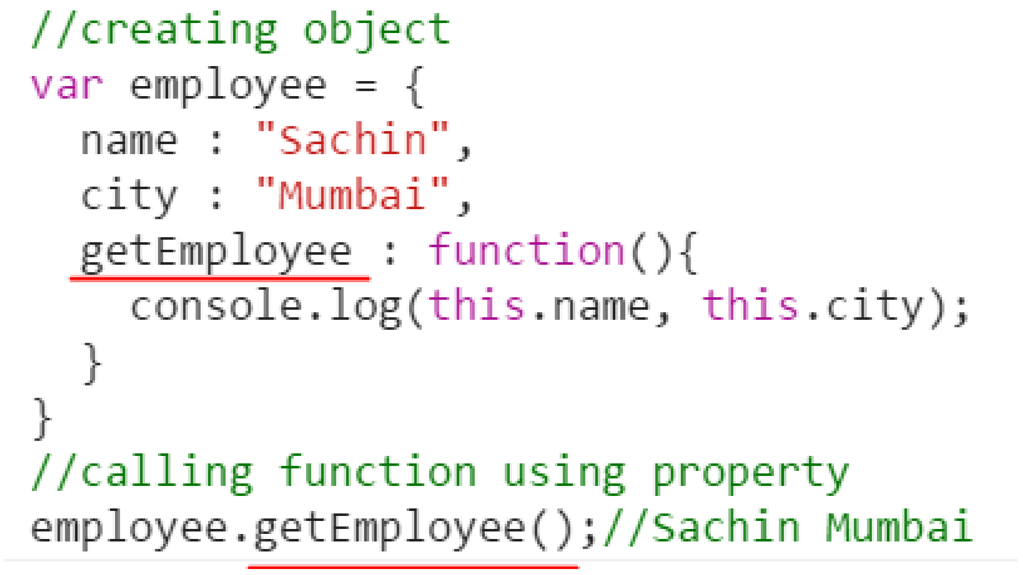
- A property of an object can itself be an object



```
//creating constructor function
function Employee(employeeId, name, city, department) {
    this.employeeId = employeeId;
    this.name = name;
    this.city = city;
    this.department = department;//deptHR object
};
//creating department object
var deptHR = {departmentId : 1, name : "HR", city : "Pune"};
//creating an object by calling constructor function
var emp = new Employee(1, "Sachin", "Mumbai", deptHR);
//Display property values
console.log(emp.employeeId, emp.name, emp.city, emp.department.name);
//1 "Sachin" "Mumbai" "HR"
```

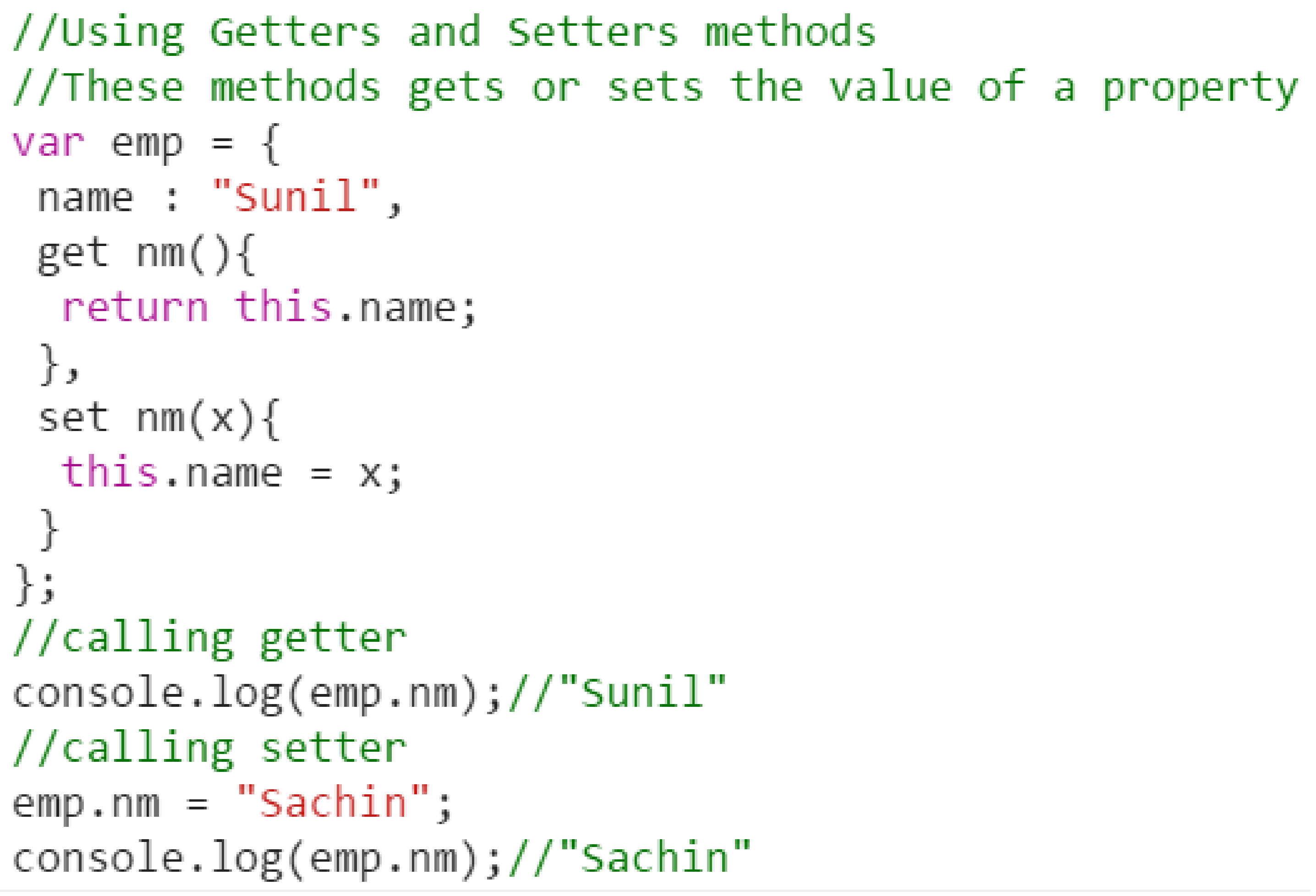
A property can be function (Method)

- A property of an object can be a function
- This type of property is called method.



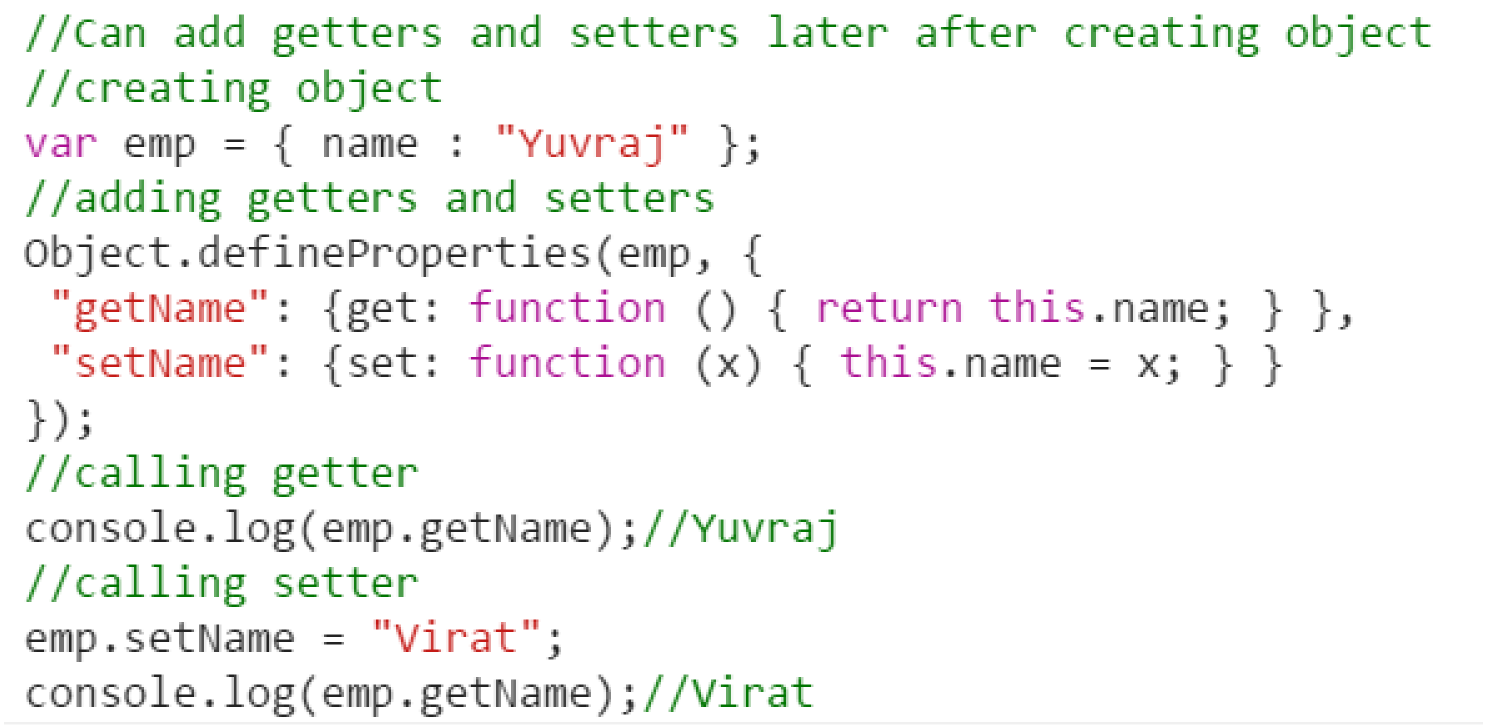
```
//creating object
var employee = {
  name : "Sachin",
  city : "Mumbai",
  getEmployee : function(){
    console.log(this.name, this.city);
  }
}
//calling function using property
employee.getEmployee();//Sachin Mumbai
```

Getters and Setters



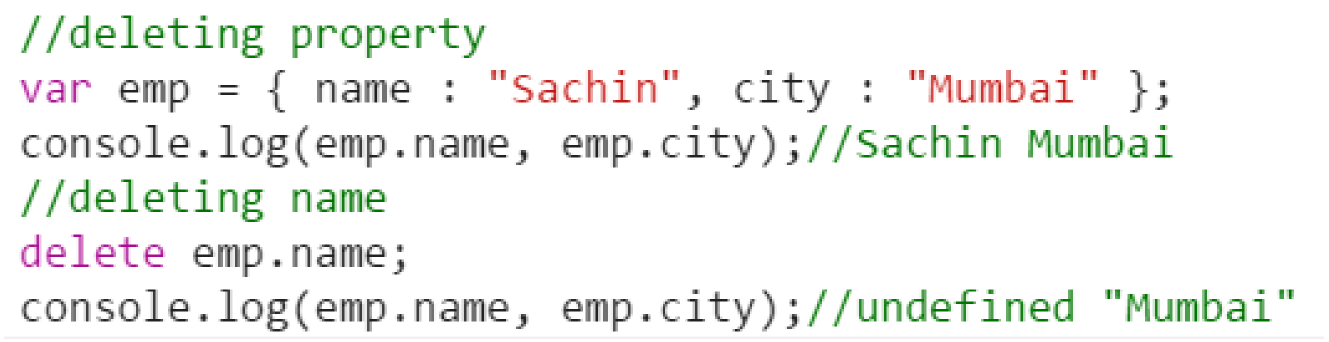
```
//Using Getters and Setters methods
//These methods gets or sets the value of a property
var emp = {
 name : "Sunil",
 get nm(){
  return this.name;
 },
 set nm(x){
  this.name = x;
 }
};
//calling getter
console.log(emp.nm);//"Sunil"
//calling setter
emp.nm = "Sachin";
console.log(emp.nm);//"Sachin"
```

Getters and Setters – Adding Later



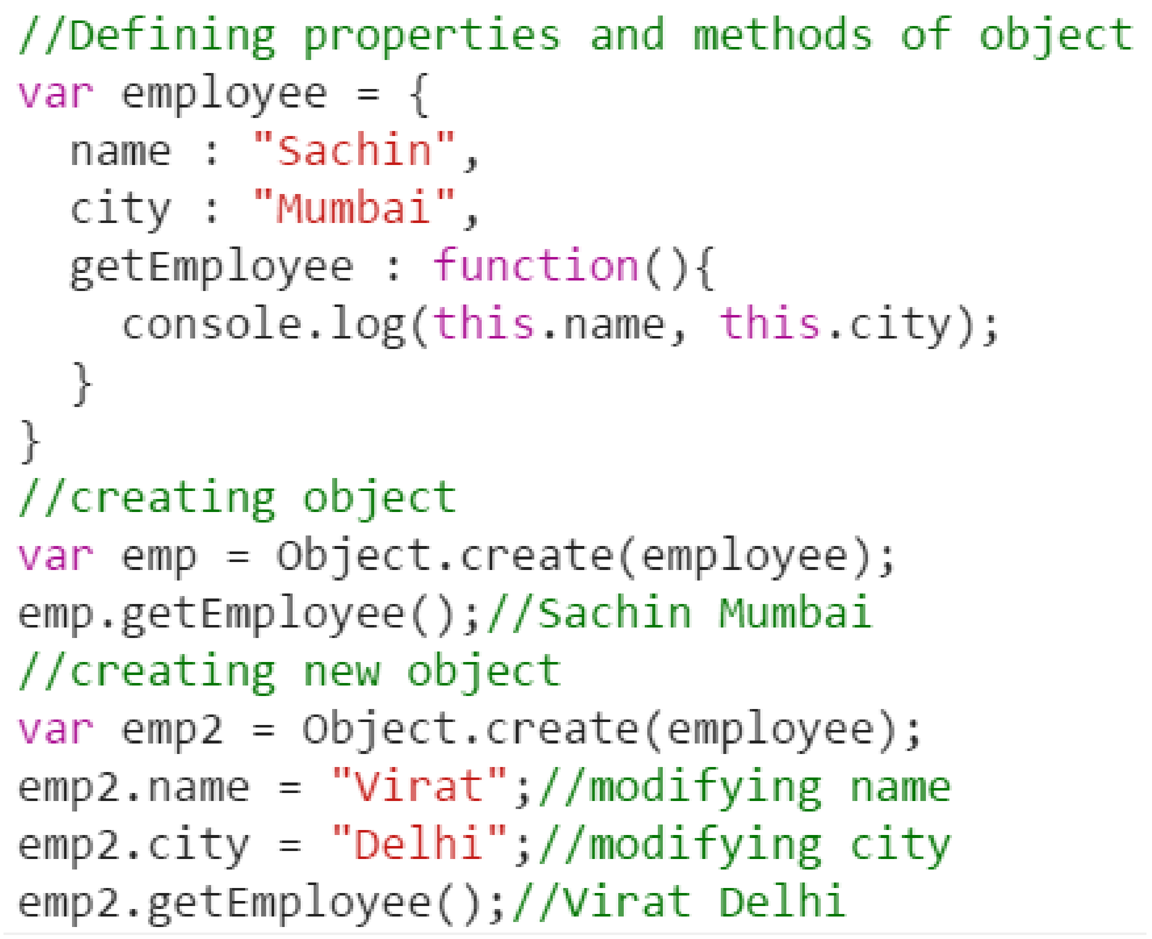
```
//Can add getters and setters later after creating object
//creating object
var emp = { name : "Yuvraj" };
//adding getters and setters
Object.defineProperties(emp, {
 "getName": {get: function () { return this.name; } },
 "setName": {set: function (x) { this.name = x; } }
});
//calling getter
console.log(emp.getName);//Yuvraj
//calling setter
emp.setName = "Virat";
console.log(emp.getName);//Virat
```

Removing Properties



```
//deleting property
var emp = { name : "Sachin", city : "Mumbai" };
console.log(emp.name, emp.city);//Sachin Mumbai
//deleting name
delete emp.name;
console.log(emp.name, emp.city);//undefined "Mumbai"
```

Object.create() Method



```
//Defining properties and methods of object
var employee = {
  name : "Sachin",
  city : "Mumbai",
  getEmployee : function(){
    console.log(this.name, this.city);
  }
}
//creating object
var emp = Object.create(employee);
emp.getEmployee();//Sachin Mumbai
//creating new object
var emp2 = Object.create(employee);
emp2.name = "Virat";//modifying name
emp2.city = "Delhi";//modifying city
emp2.getEmployee();//Virat Delhi
```

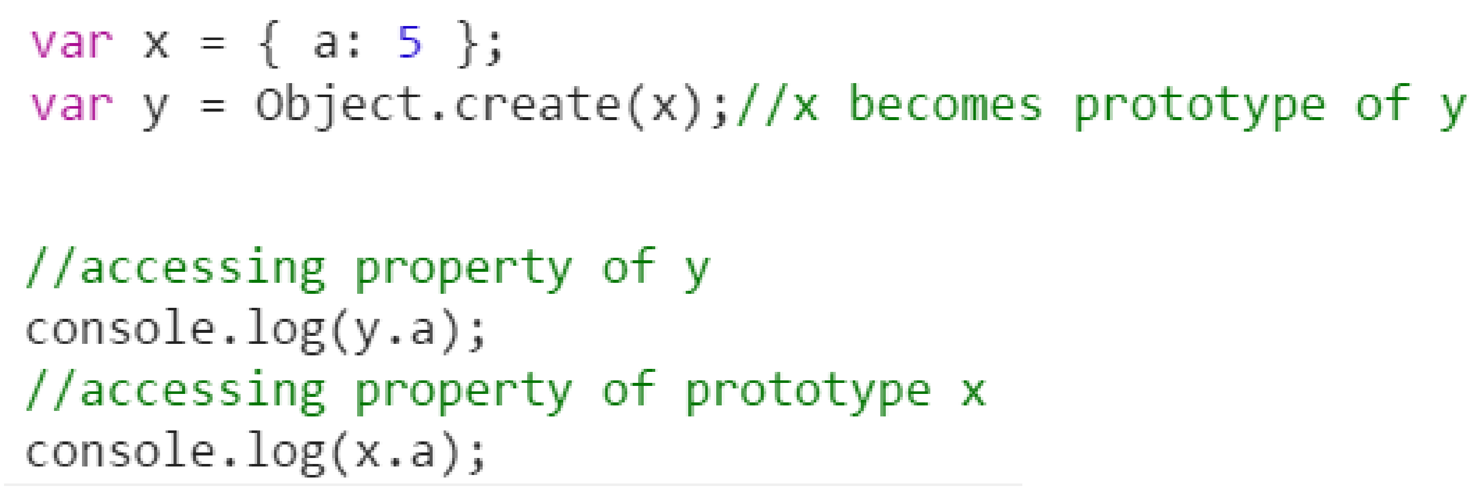
Inheritance

Prototype

- The prototype is an object
- Every Javascript object inherits from Object.prototype
- Object.prototype is on the top of the prototype chain.

Creating Prototype

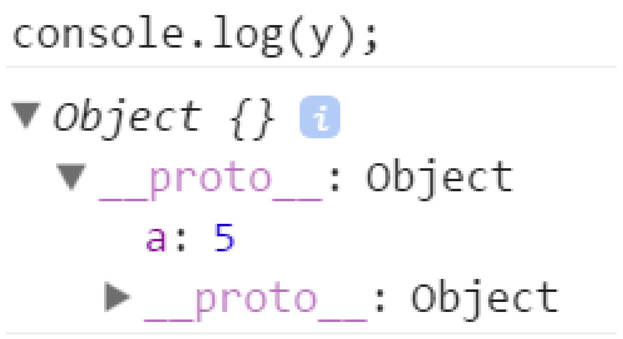
- Creating prototype using Object.Create()



```
var x = { a: 5 };
var y = Object.create(x);//x becomes prototype of y

//accessing property of y
console.log(y.a);
//accessing property of prototype x
console.log(x.a);
```

Looking at prototype

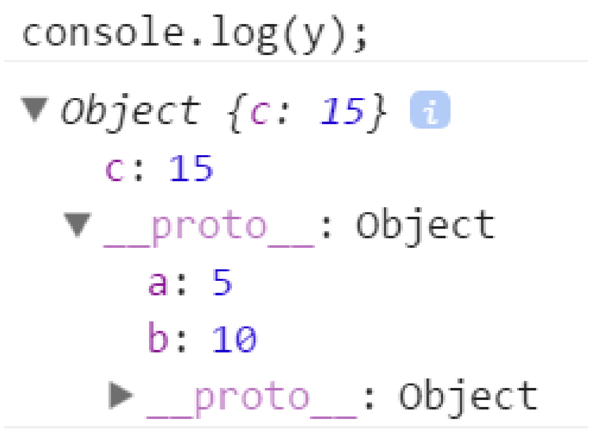
- Prototype object is shown up if expanded



```
console.log(y);
▼Object {} i
  ▼__proto__: Object
     a: 5
   ▶__proto__: Object
```

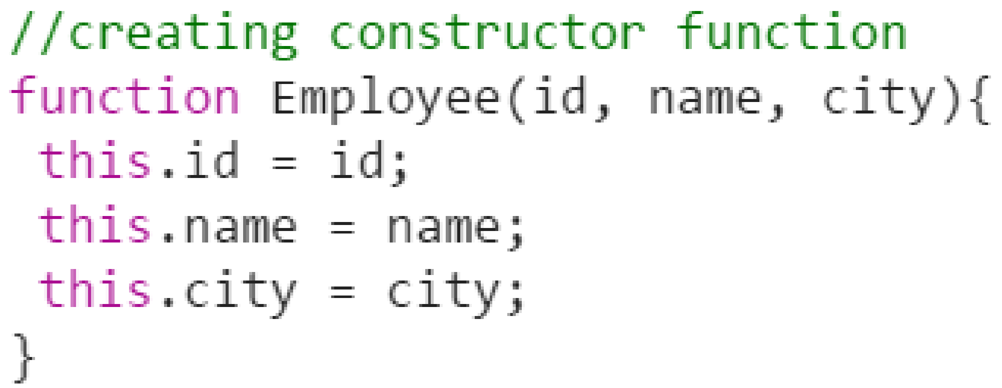
Adding New Property

```
//adding new property to x
x.b = 10;//if new property is added to prototype,
//its available to inherited objects
console.log(x.b, y.b);//10 10
//adding new property to y
y.c = 15;//its added only to y, not available to x
console.log(x.c, y.c);//undefined 15
```

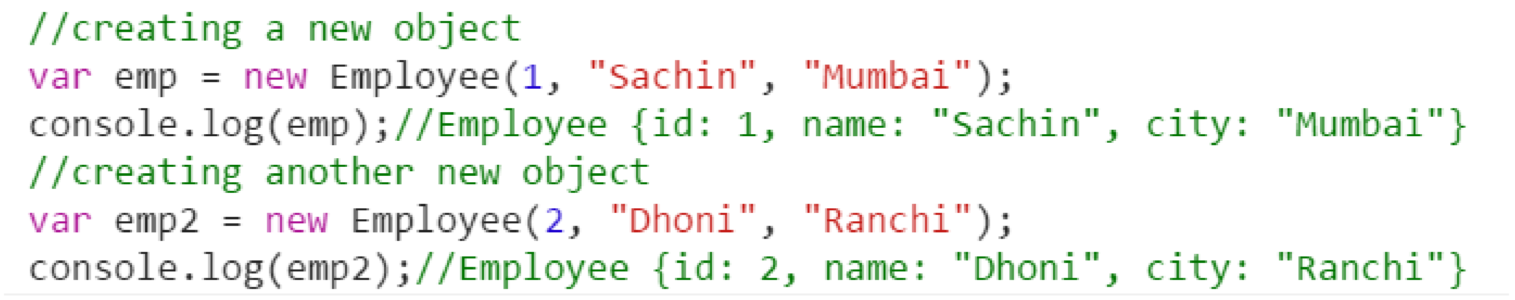


```
console.log(y);

▼Object {c: 15}
   c: 15
  ▼__proto__: Object
     a: 5
     b: 10
   ▶__proto__: Object
```

Using Constructor Function



```
//creating constructor function
function Employee(id, name, city){
 this.id = id;
 this.name = name;
 this.city = city;
}
```



```
//creating a new object
var emp = new Employee(1, "Sachin", "Mumbai");
console.log(emp);//Employee {id: 1, name: "Sachin", city: "Mumbai"}
//creating another new object
var emp2 = new Employee(2, "Dhoni", "Ranchi");
console.log(emp2);//Employee {id: 2, name: "Dhoni", city: "Ranchi"}
```

Adding Property and Method

```
//adding new property to specific object(emp)
emp.email = "sachin@cybage.com";//its added only to emp, not to emp2
console.log(emp);//Employee {id: 1, name: "Sachin", city: "Mumbai",
//email: "sachin@cybage.com"}
//adding new method to specific object(emp2)
emp2.getEmpInfo = function(){
 return "name : " + this.name + ", city : " + this.city;
}//added only to emp2 and not to any other object
emp2.getEmpInfo();//"name : Dhoni, city : Ranchi"
```

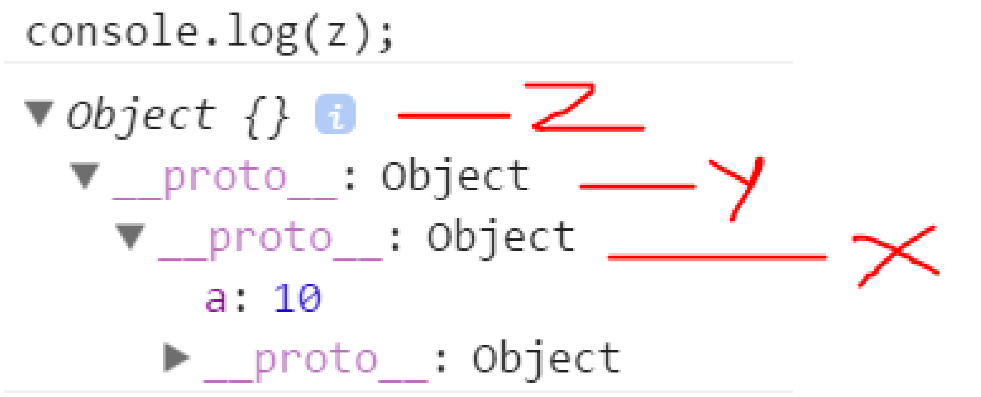
Cybage® Delivering Value Scientifically

Using prototype Property

```
//using prototype property to add new property
Employee.prototype.state = "Delhi";
//now state is available to every object
//JavaScript first looks in the current object.
//if the property is found, the value is retrieved.
//if it is not found in the current object,
//it keeps looking up to the next level of prototype chain
//until the property/method is found or prototype becomes null
emp.state;//"Delhi"
emp2.state;//"Delhi"
```

Inheritance Chain

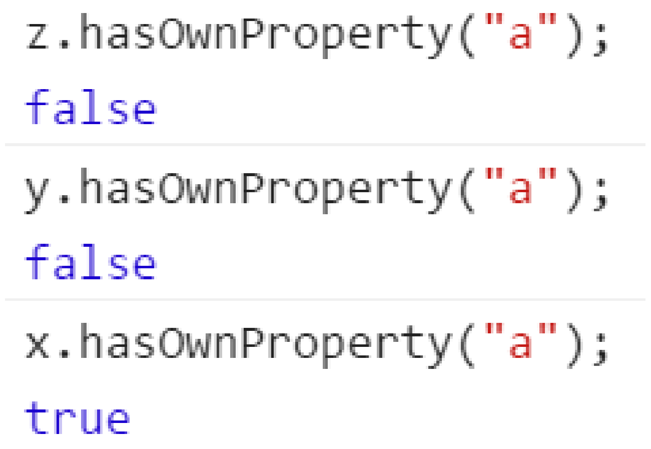
```
var x = {a: 10};
// x ---> Object.prototype ---> null

var y = Object.create(x);
// y ---> x ---> Object.prototype ---> null
console.log(y.a); // 10 (inherited)

var z = Object.create(y);
// z ---> y ---> x ---> Object.prototype ---> null
console.log(z.a); // 10 (inherited)
```



```
console.log(z);

▼Object {}
  ▼__proto__: Object
    ▼__proto__: Object
        a: 10
      ▶__proto__: Object
```

The hasOwnProperty() method

- Returns boolean
- Tells whether object has the property
- Returns false if the property is of prototype



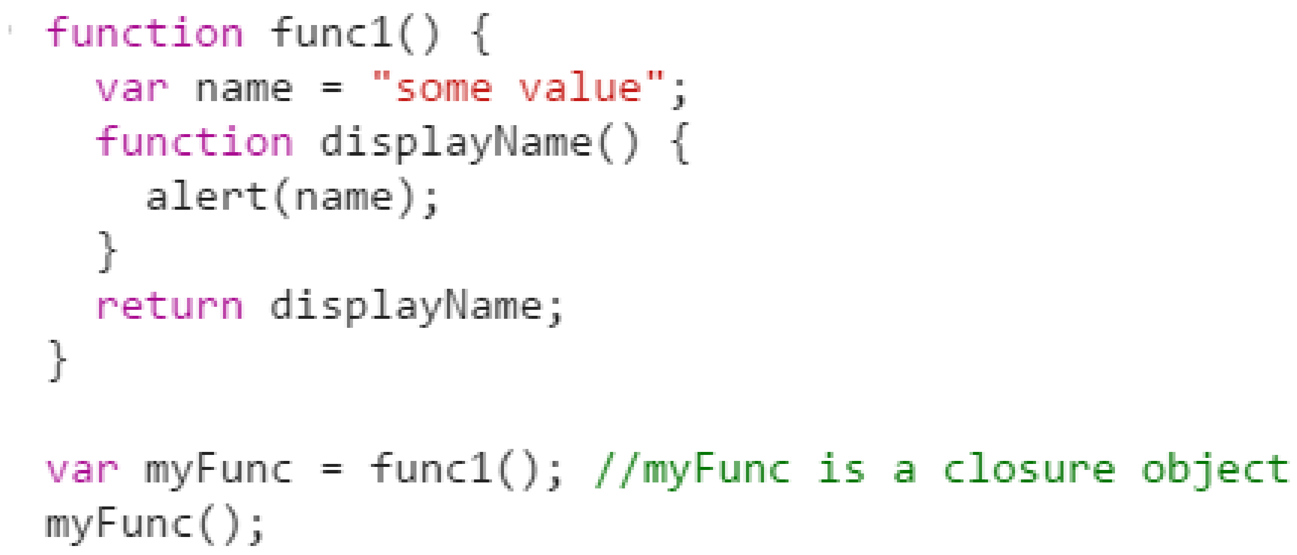
```
z.hasOwnProperty("a");
false
y.hasOwnProperty("a");
false
x.hasOwnProperty("a");
true
```

Closure

- Inner function forms a closure

```
function func1() {
  var name = "some value"; // name is a local variable created by func1
  function displayName() { // displayName() is the inner function, a closure
    alert(name); // use variable declared in the parent function
  }
  displayName();
}
func1();
```

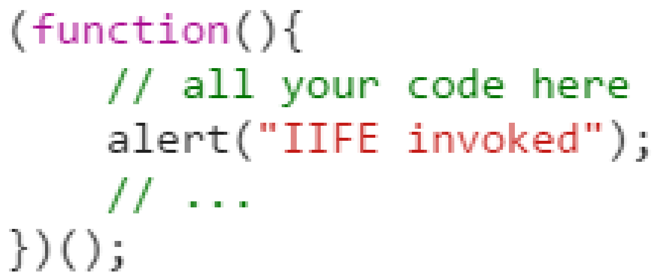
Closure...



```
function func1() {
  var name = "some value";
  function displayName() {
    alert(name);
  }
  return displayName;
}

var myFunc = func1(); //myFunc is a closure object
myFunc();
```

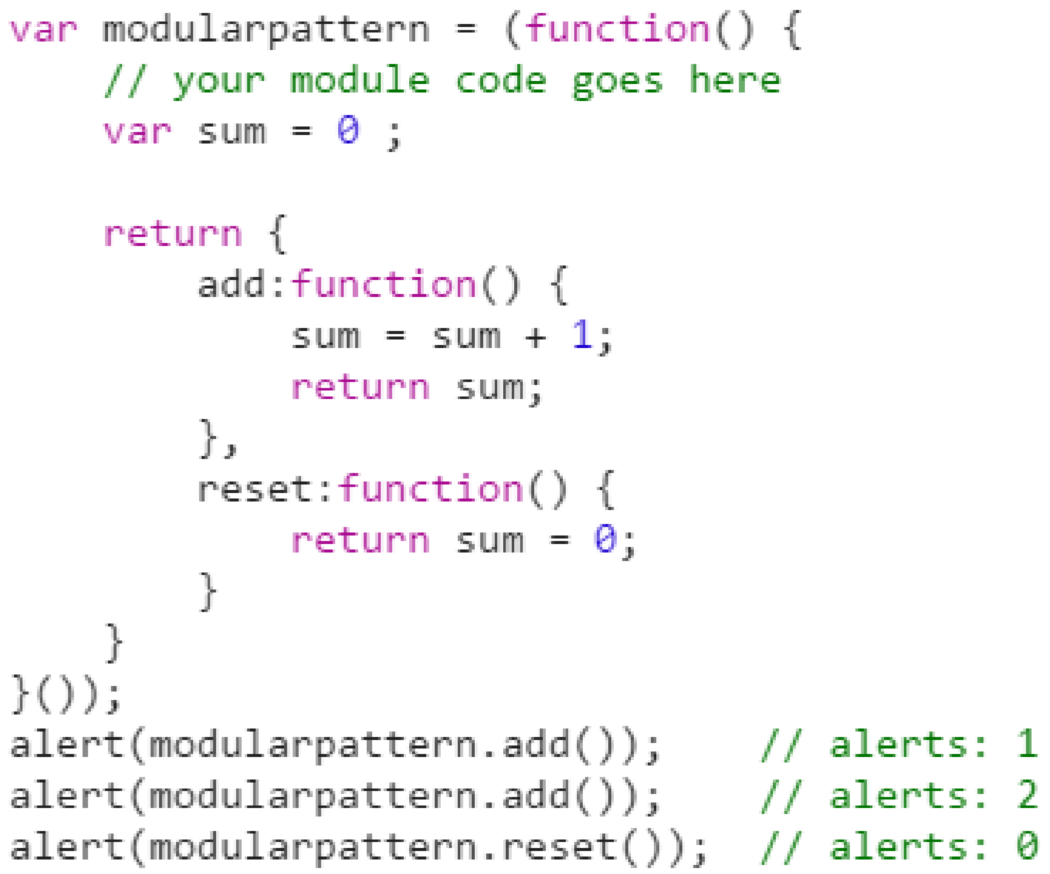
IIFE Pattern

- Immediately invoked function expression
- It executes immediately after its created.



```
(function(){
    // all your code here
    alert("IIFE invoked");
    // ...
})();
```

Modular Pattern



```
var modularpattern = (function() {
    // your module code goes here
    var sum = 0 ;

    return {
        add:function() {
            sum = sum + 1;
            return sum;
        },
        reset:function() {
            return sum = 0;
        }
    }
}());
alert(modularpattern.add());    // alerts: 1
alert(modularpattern.add());    // alerts: 2
alert(modularpattern.reset());  // alerts: 0
```

Summary

- Functions
- Scopes
- Objects
- Inheritance

Bibliography, Important Links

- https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Books:

Delivering Value Scientifically

Any Questions?

Thank you!