**Understanding JavaScript**

**Frameworks**

Frameworks like jQuery, Angular JS is written using JavaScript

**Objects in Javascript**

Name – Value pairs

Address : {

STREET1 : “”

CITY\_STATE : {

“CITY” : “BANGALORE”,

“STATE” : “KARNATAKA”

}

}

**Execution Object**

Global execution object – this

* Depending on where it is execution, a global context is created. Ex: in a browser – this will be a window object , in nodejs it will be different

**Hoisting**

* Before execution, all the variables & functions are placed inside the memory space
* Variables will be set to undefined. When executing line by line, depending upon whatever value it has it is used
* Javascript is single threaded & synchronous

**Multiple Execution Contexts**

function a() {

b();

}

function b() {

}

a()

It finds out b(), creates Execution context for function b()

--------------------------------------------------------------------------

Execution context - function a()

While it starts to run a(), it finds function and creates a execution context for a

Global Execution context would be created – this

**Scope Chain**

function a() {

b();

}

function b() {

}

a()

|  |
| --- |
| B() Execution context |
| A() Execution context |
| this ( Global Execution context ) |

function a() {

function b() {

}

}

a()

Here function b is inside a -🡪 In this case B() outer context is A()

|  |
| --- |
| B() Execution context |
| A() Execution context |
| this ( Global Execution context ) |

**LET variables 🡪 Block Scoping**

let a

* Only available for the particular Block scope
* It can be used only after it is declared , can’t be used before that like var

**Asynchronous Callbacks**

* Apart from Execution context, there is something called Event Queue

Ex: Click event, etc

Once all the execution context completes, it will listen to the event queue (ex: click event)

**Primitive types**

* undefined
* NULL - use null if you want to check for non-existence in your code
* boolean
* number
* string
* symbol 🡪 only in ES6

**OBJECTS AND FUNCTIONS**

**OBJECTS**

* sitting in memory and can have the following. This object refers to the members in memory
  + properties ( primitive data type)
  + other objects ( This is also called as properties)
  + functions

**Object creation – not a preferred way**

person = new Object();

person[“fname”] = “sunil”

Accessing fname 🡪 person.fname 🡪 Retrieve sunil

Either. (DOT) operator or [] can be used for referring

**Another way of creating Object**

var person = {}

**OBJECT LITERALS**

**JavaScript and JSON**

JSON is strict, it requires double quotes whereas JavaScript object it is not mandatory.

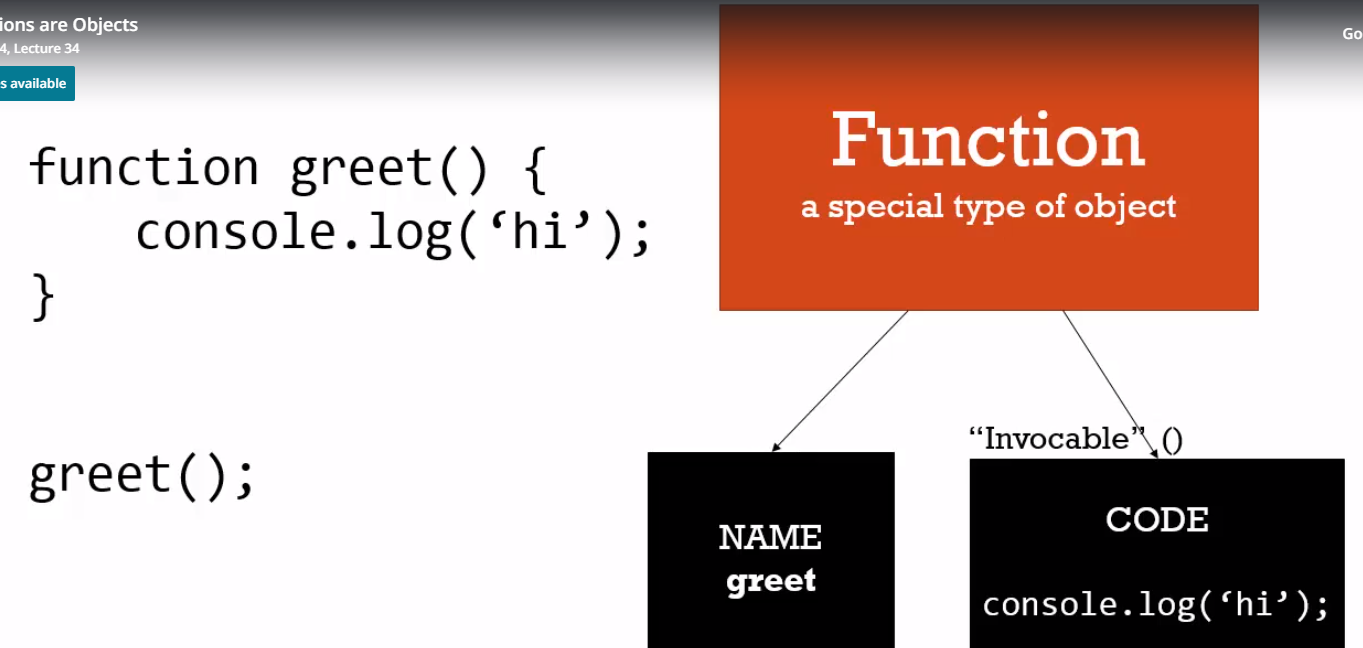
* All JSON are JavaScript object but not the other way so
* Javascript object -🡪 use stringify to convert to JSON
* JSON parse 🡪 use to parse JSON string to Javascript object

**Javascript and Functions**

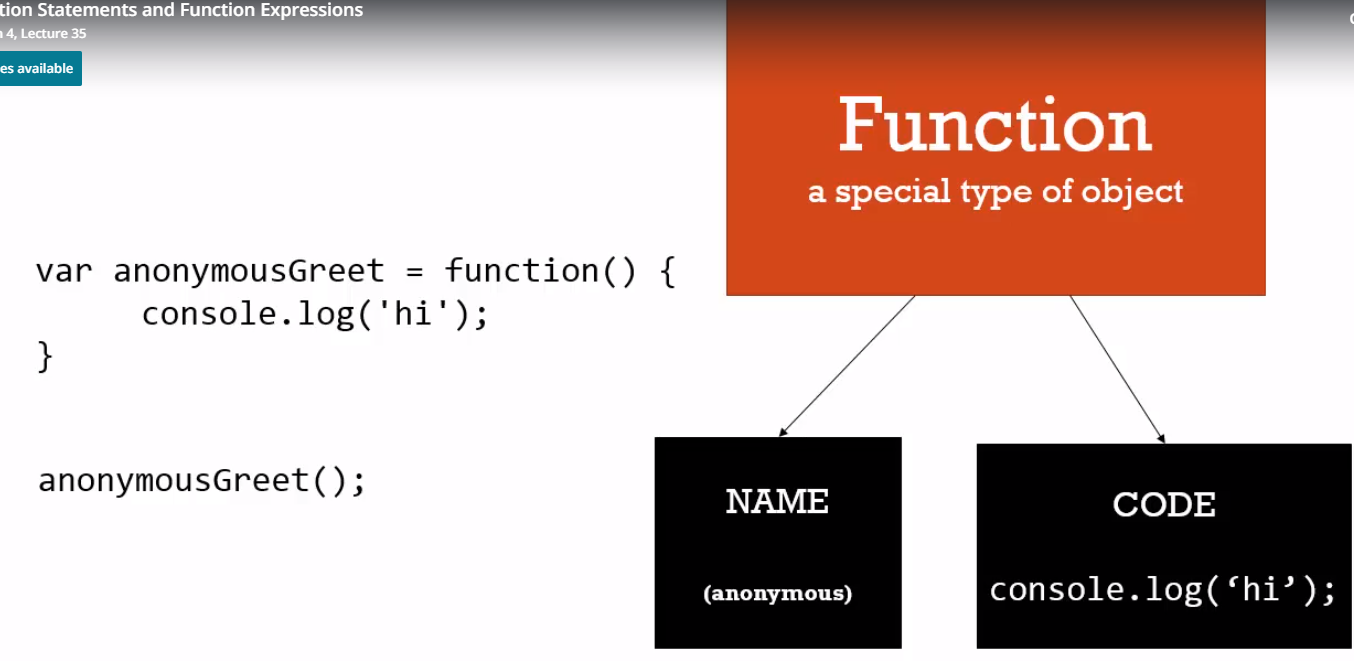
Functions are object in javascript. It is present in memory

**Function object**

* Need not have a name
* Code what you write inside a function is nothing but the properties
* It can have other properties like variables



**Anonymous Functions**



**Passing function as a parameter**



**Object and this keyword**

* this keyword by default points to global object (Ex: Window Object)
* if Function is a method of an object, it points to the current object

**For example:**

function add() {

console.log(this) 🡪 Here this keyword points to the global object

}

var x = function() {

name : “sunil”,

getAddress : function() {

console.log (this) 🡪 This points to current object

}

var y = function() {

Console.log(this) 🡪 this points to the Global object

}

// Suppose there is a inner function

Y()

}

**Arrays**

Arrays can contain functions, objects, mixed types



Spread and arguments

arguments – This is a built in variable used to find the number of arguments which are passed to a function. This is not an array though. However, it can be accessed via [] operator

**Understanding Closures**

* Even if the Function Execution context ends, all the variables/memory associated would remain

**STEP 1**



**STEP 2**

* After function execution is completed, function execution context goes away. Under normal cases

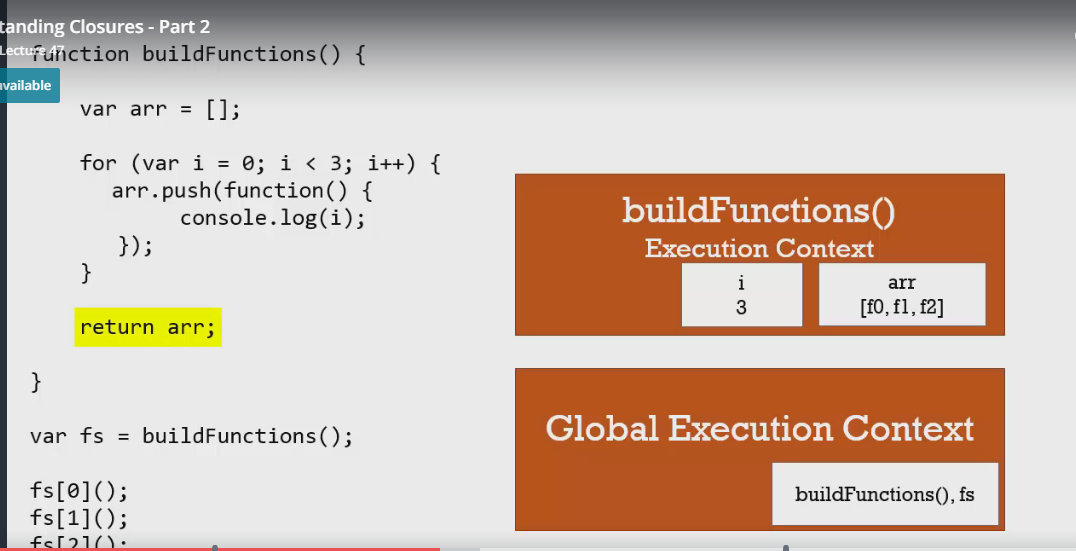


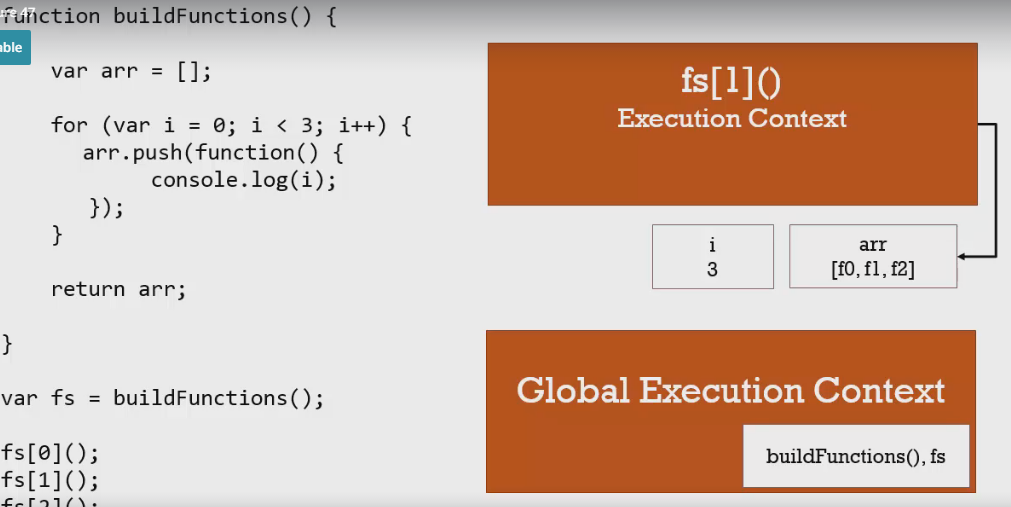
**STEP 3**

When sayHi(‘Tony’) is invoked, new function execution context is created.

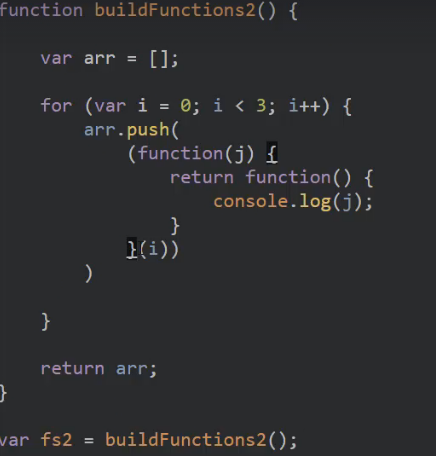


**Closures : Part 2**





**Output** 3 3 3

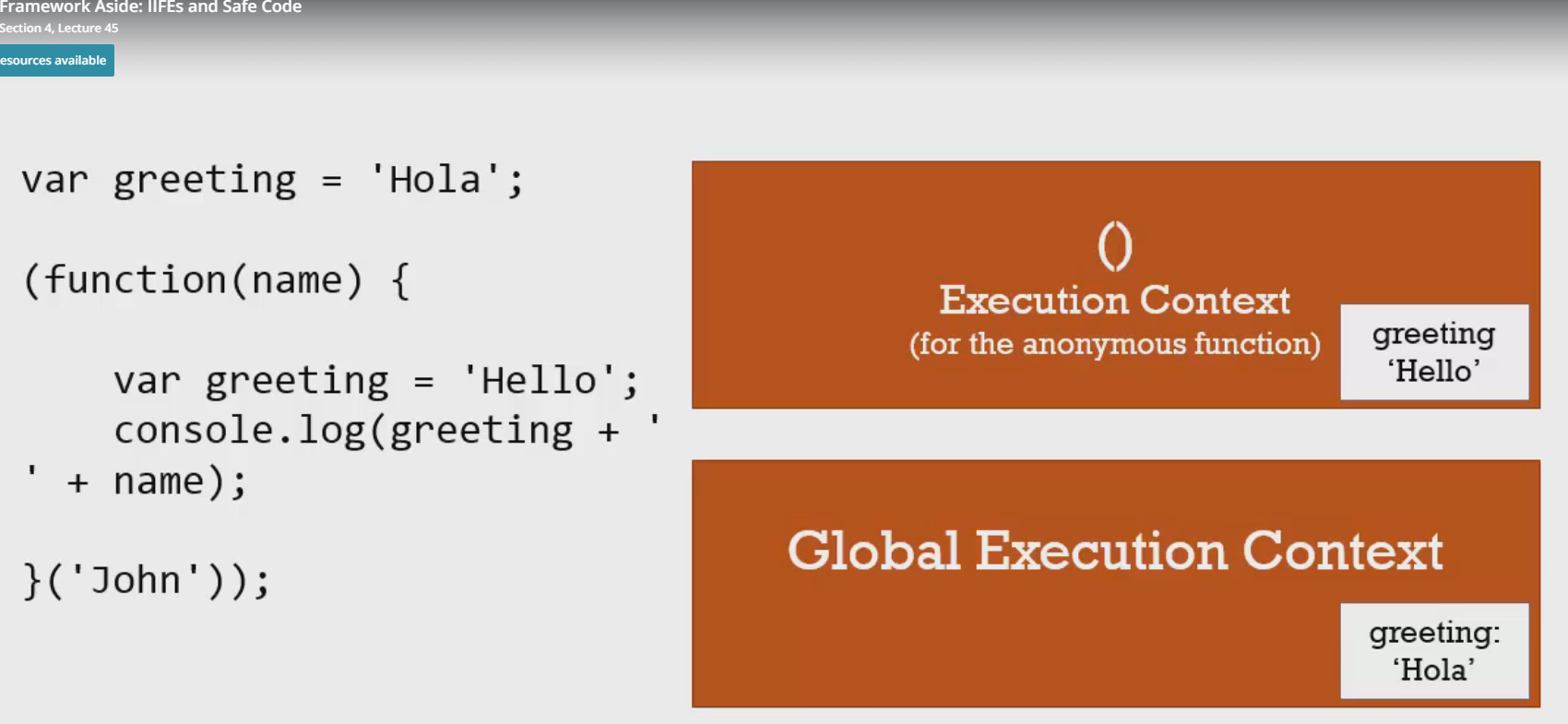




**CallBacks**

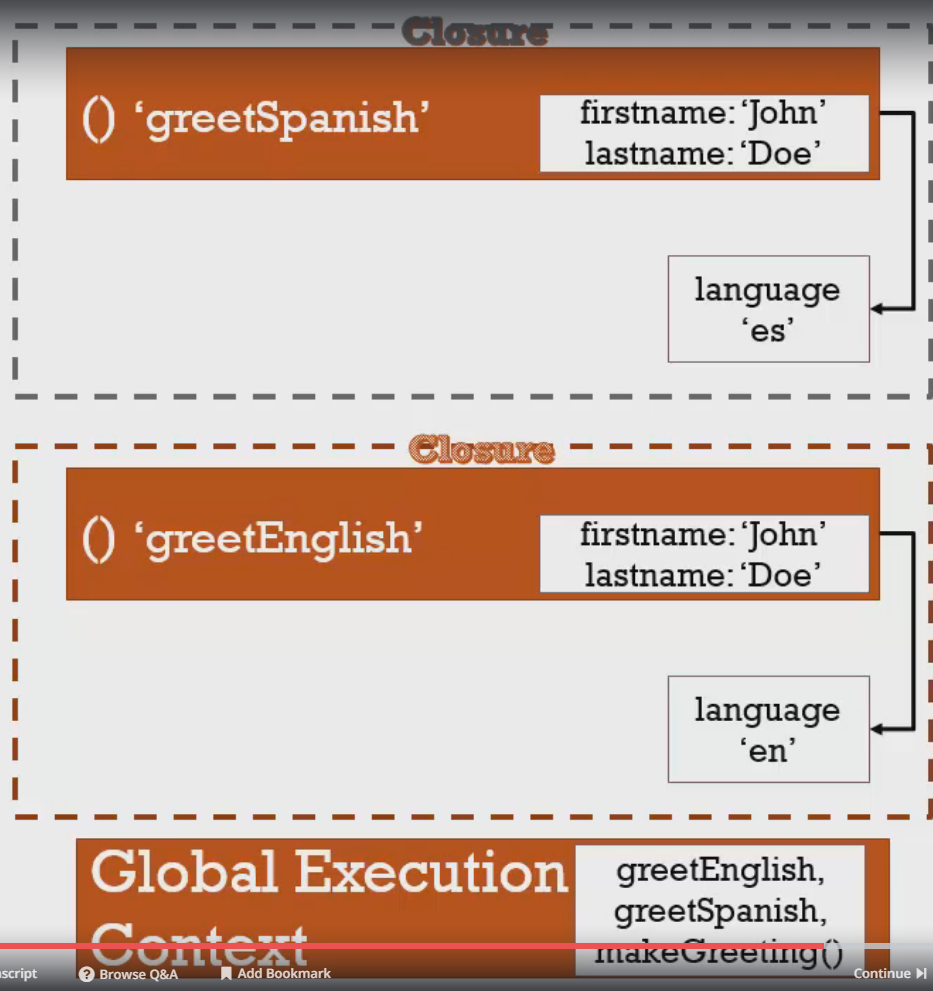


**Immediately invoked functions**



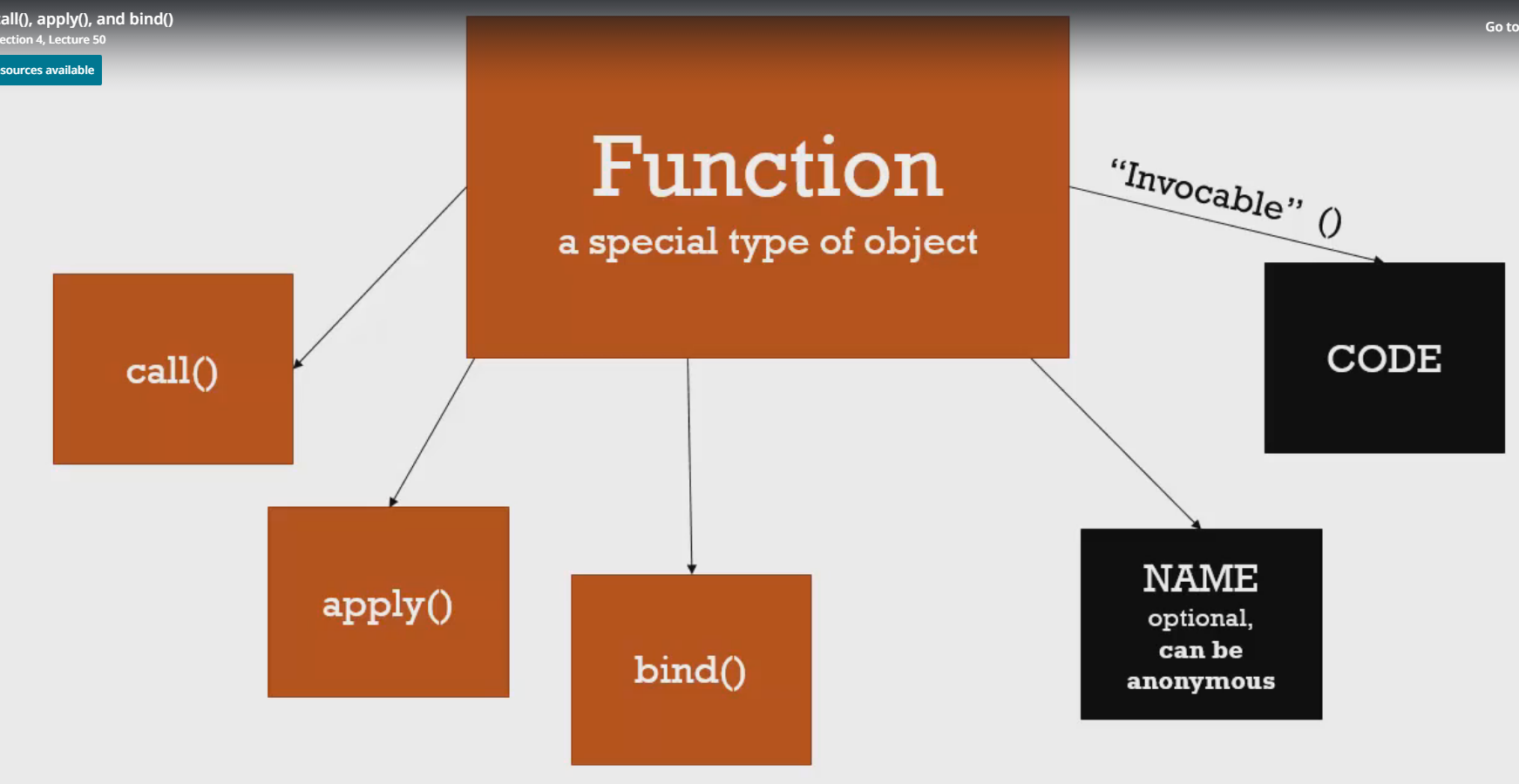
* Important thing to be noted is IIFE execution context is different from Global Execution Context

**Function factories**





**CALL/APPLY/BIND**



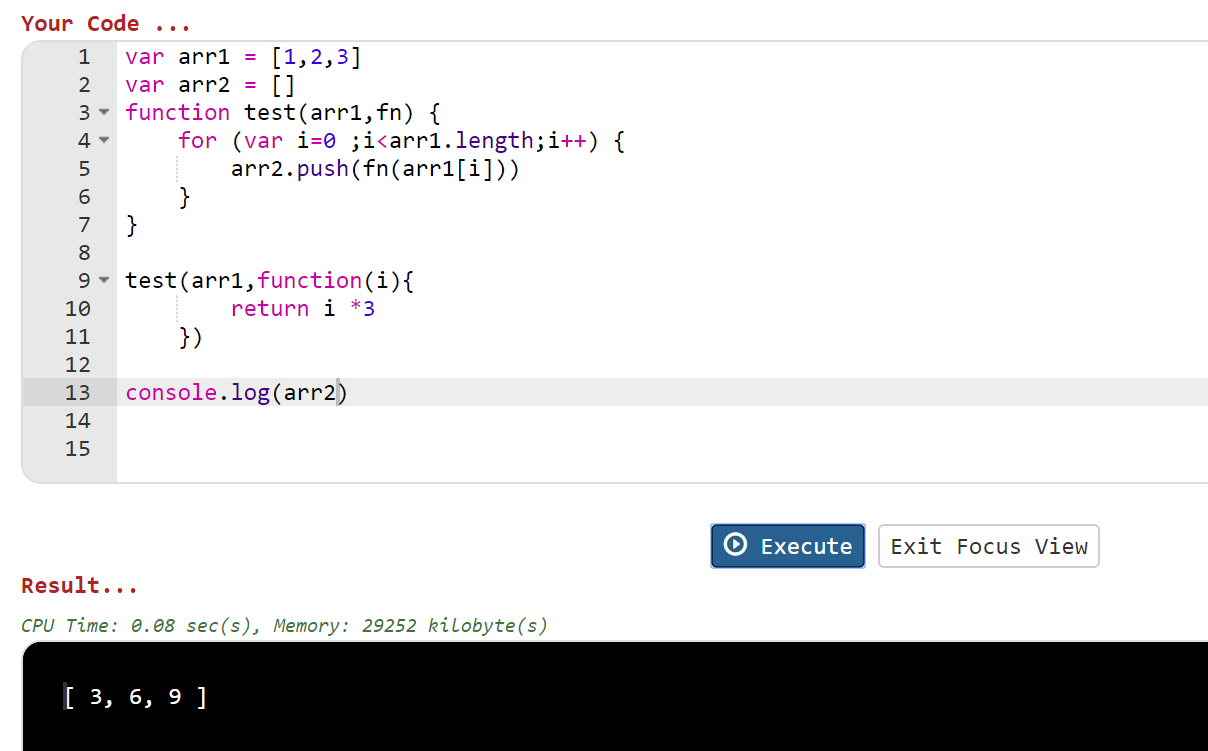
**All the functions have access to these 3 methods**

* call()
* apply()
* bind()

**Functional Programming**

In this example below , each value of array is invoked using function callback . Implementation can vary based on user needs. In this case, the value is multiplied by 3. However , it can be used for different needs.

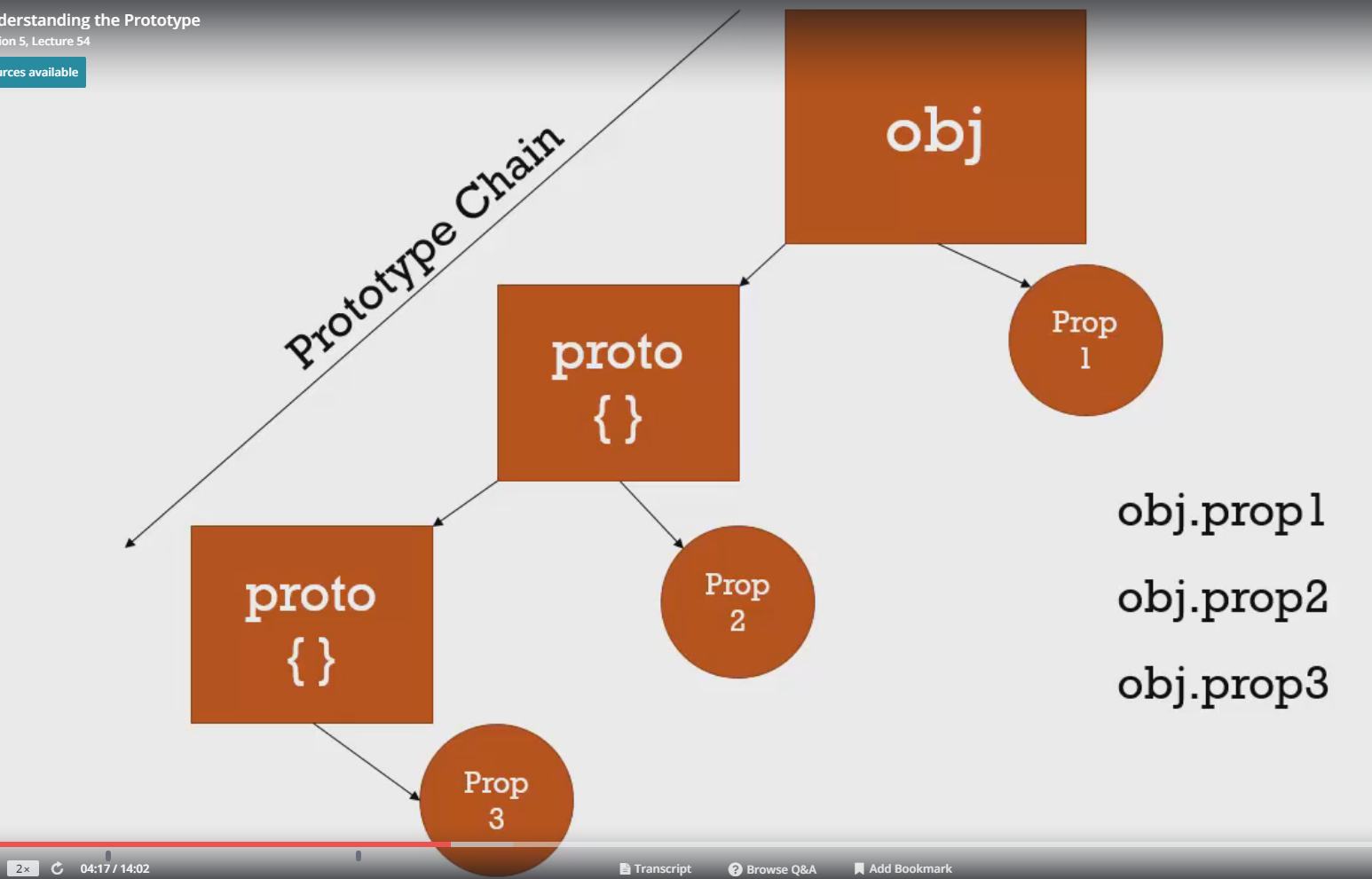
Example : In scala, map – reduce map etc. are used. What needs to happen to data is our control



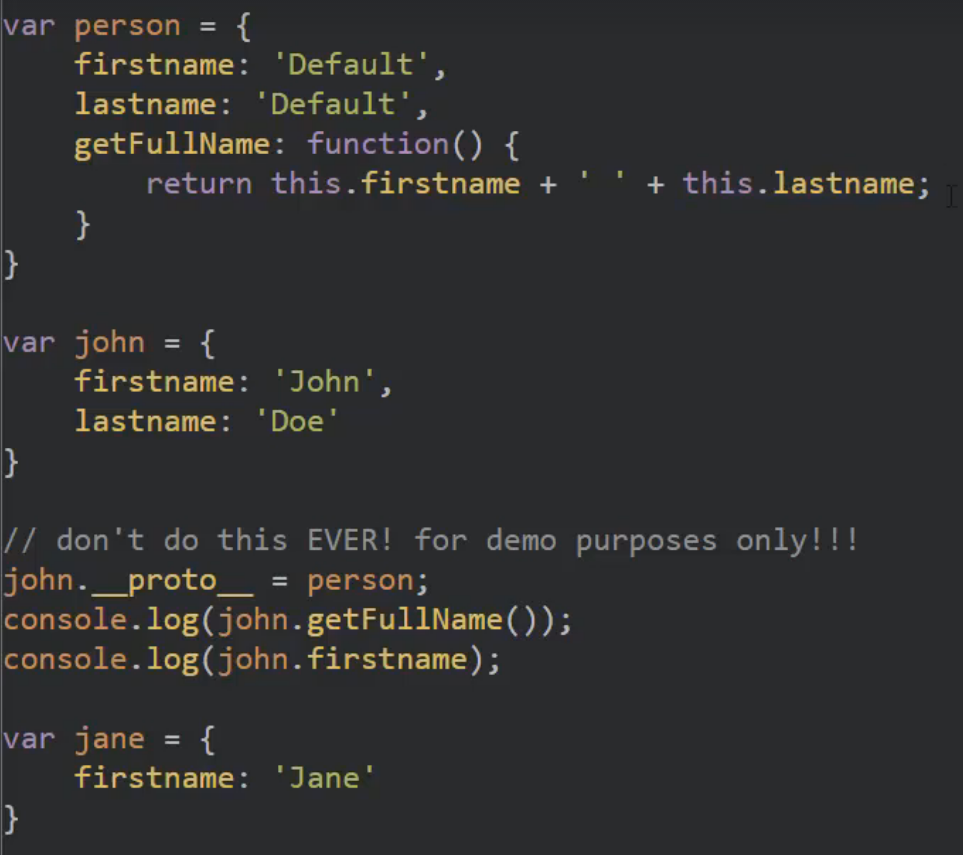
* Underscore.js library has all the features like map, filter, invoke etc.
* Loadash library is similar to Underscore but comparatively fast

Javascript – Object Oriented

Prototype Inheritance

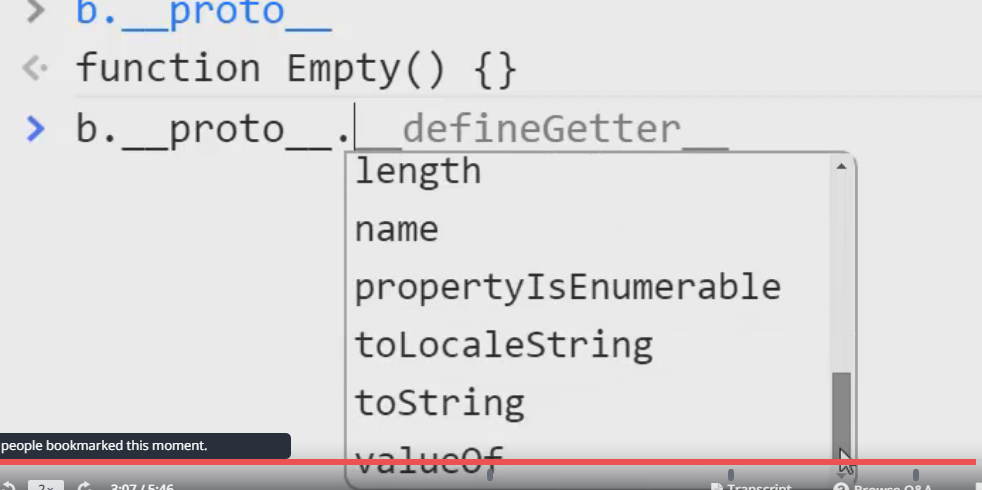


* When objects/properties are referenced, it will first look if it has direct access otherwise – it will get from the proto. It can even drill down to next proto and next proto and so on..



DONOT ever inherit using \_\_proto\_\_. It could cause performance issues

This is the reason we have access to basic methods like call, apply, toString etc



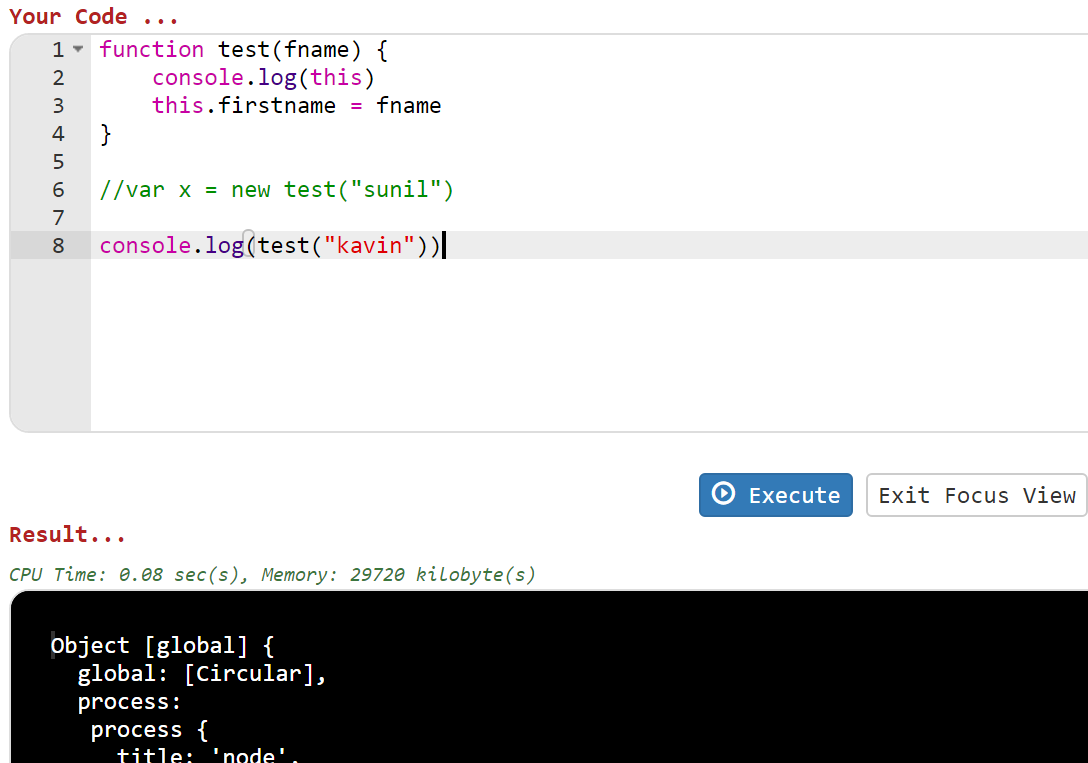
**Object Oriented Concepts**

**Function constructors**

* Functions are nothing but the objects



Without new Keyword. this points to the global object



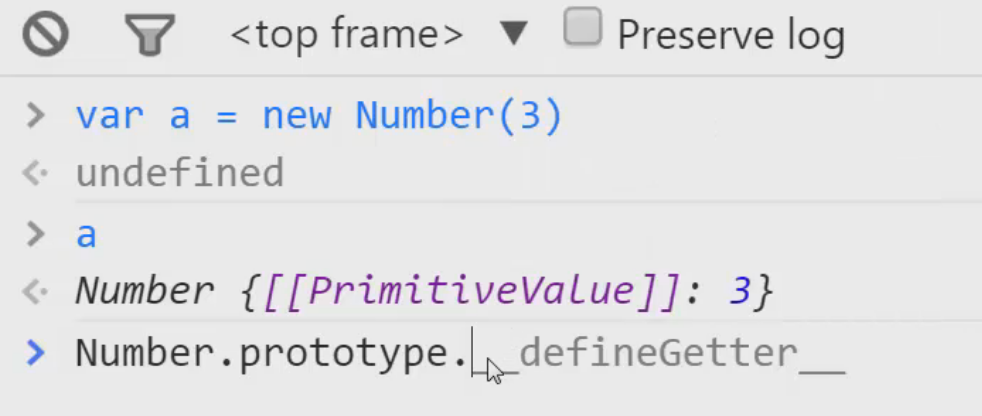
With new Keyword. this creates an empty object



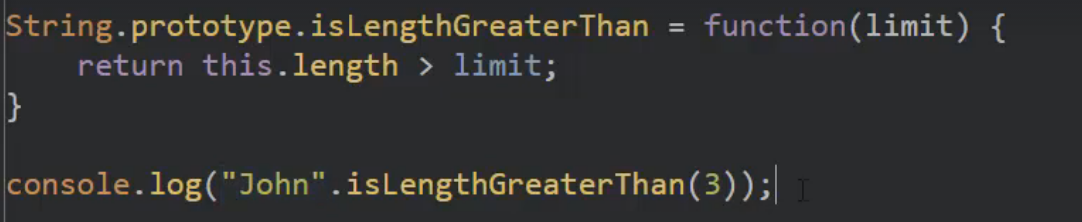
Bottom Line : new keyword is not used normally but just created to have a feel for other language developers that JS isn’t different

**Prototype**

* By default, every function (which is a object) gets a prototype
* One can add methods , properties using prototypes



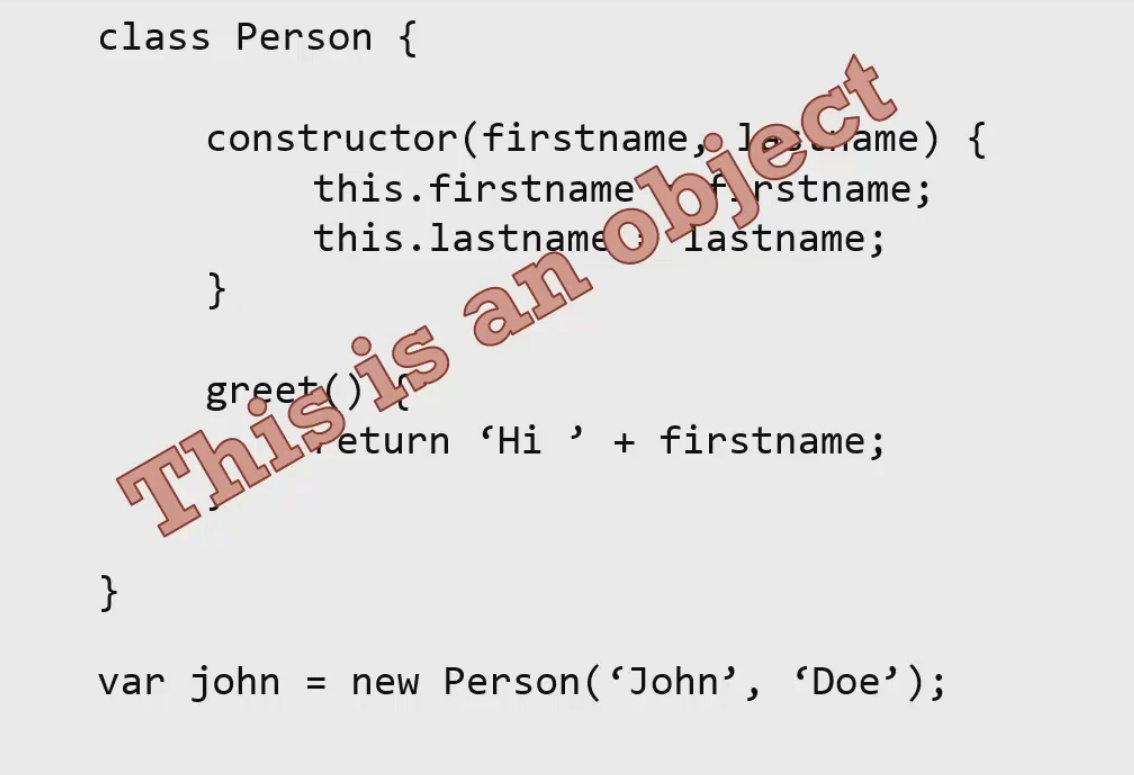
Below new method is added to String prototype and you can notice the method is called as it is



**Class Keyword**

* Class keyword can be used in JS. But in other languages, object gets instantiated only when encountered with new keyword. However, in JS function is an object. By using new , you are creating a new object using function object
* Class keyword is available from ES6 JS

Here Person is an object



Adding properties to Person object using extends keyword instead of using prototype. ( Behind the scenes it is using prototype)



* Here InformalPerson object is set with the properties of Person

**typeof Keyword**

* Finding out variable type

**instanceof Keyword**

* Finding out is the object is an instance of <defined type>

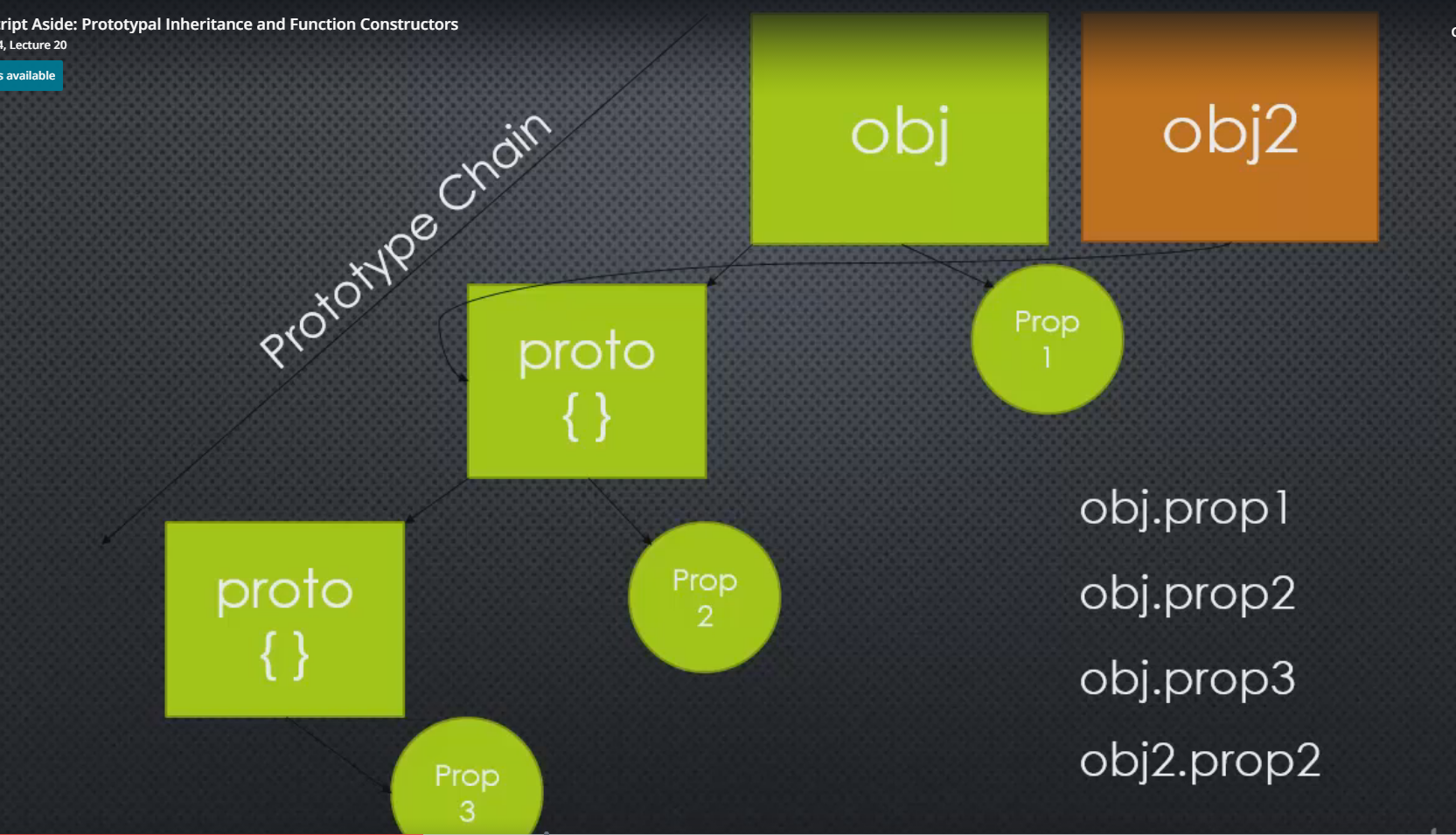
**Strict mode**

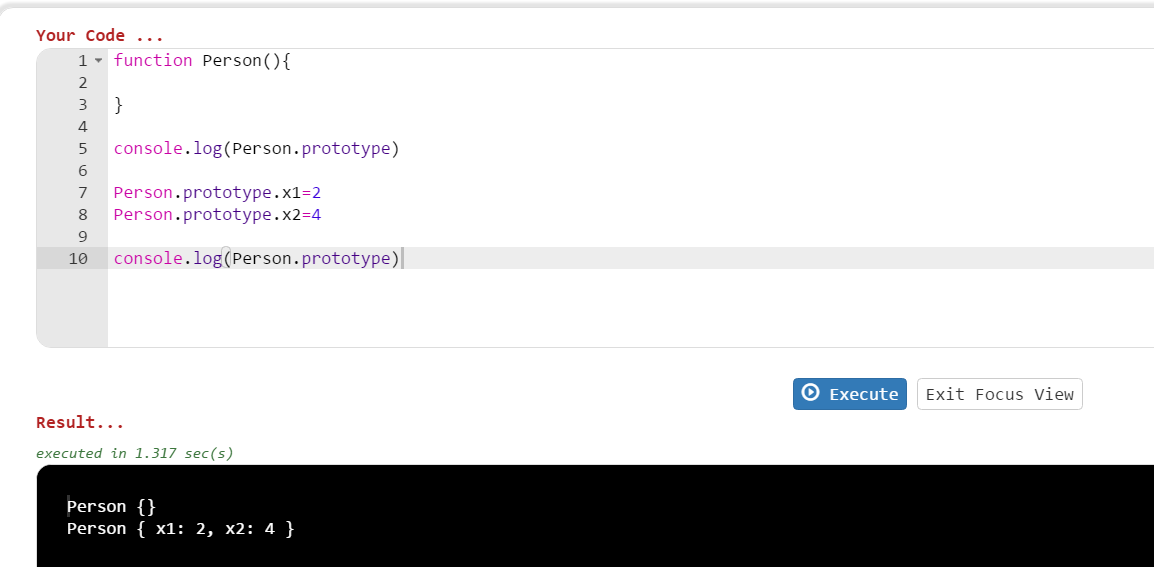
* use strict – can be used globally or inside the function



**Prototype Inheritance – Revisited**

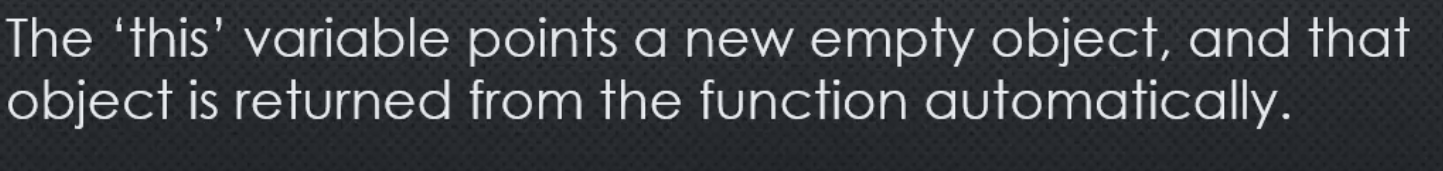
* Inheritance – One Class implements properties/functions of other class
* Every Object has a property called – prototype
* Using this prototype, one can access other objects – This is called Prototype inheritance





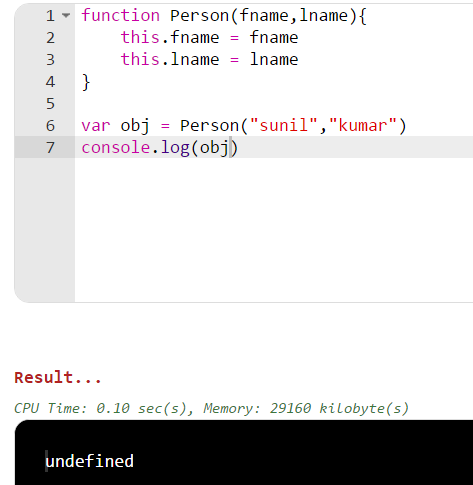
Here using Person object – we are adding more properties to it. This is also available to other methods. This becomes inheritance

**Function constructors**



* Notice below function has a this keyword. Whenever new keyword is used to create an object, the reference or this is returned to the caller.

**Without new Keyword – this is not returned**



**With new Keyword – this is returned**

