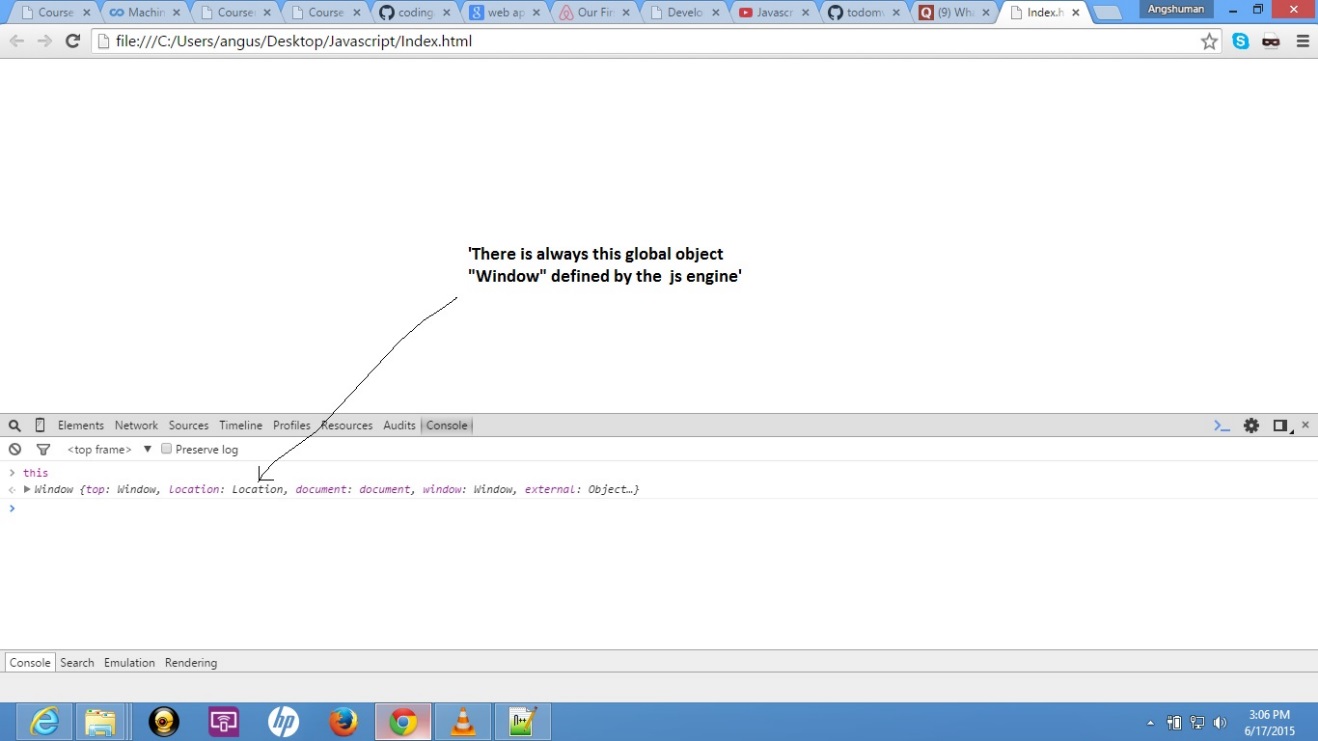
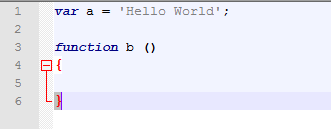
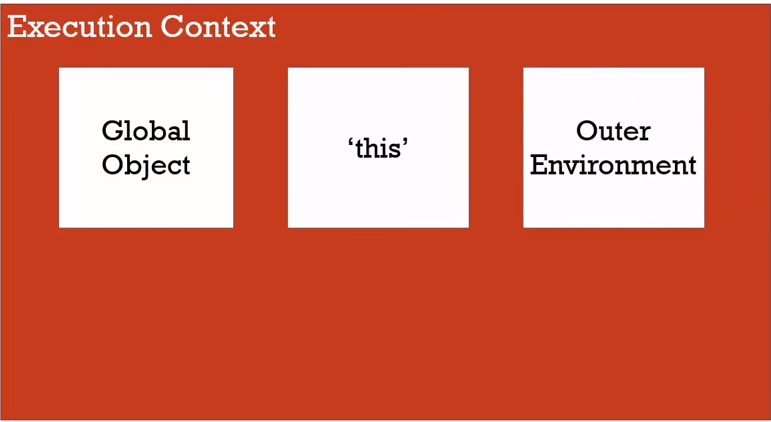
**Chapter 1: Introduction (Preview)**

*-Compiled by: Angshuman Gupta*

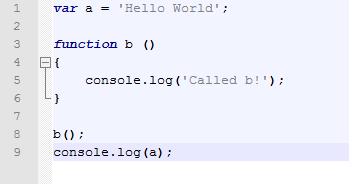


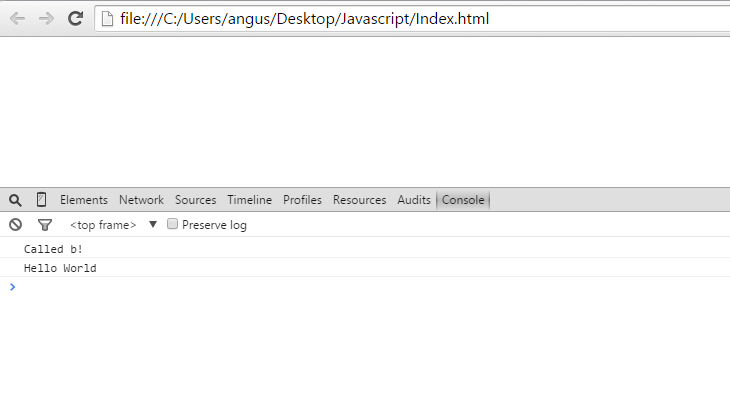




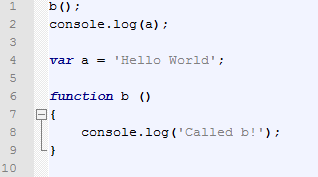


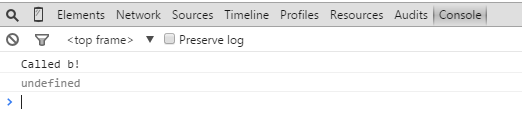
This Execution Context is the wrapper that wraps the code.

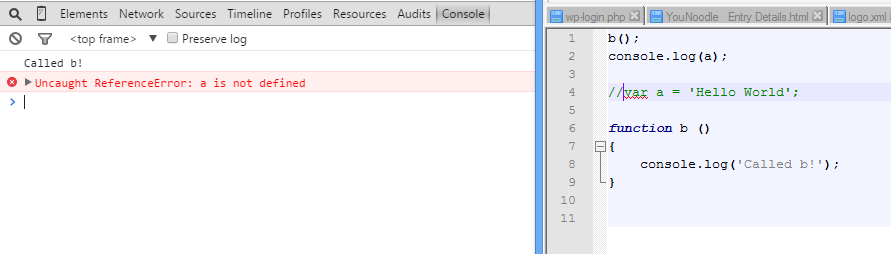




Hoisting: Acts from the top

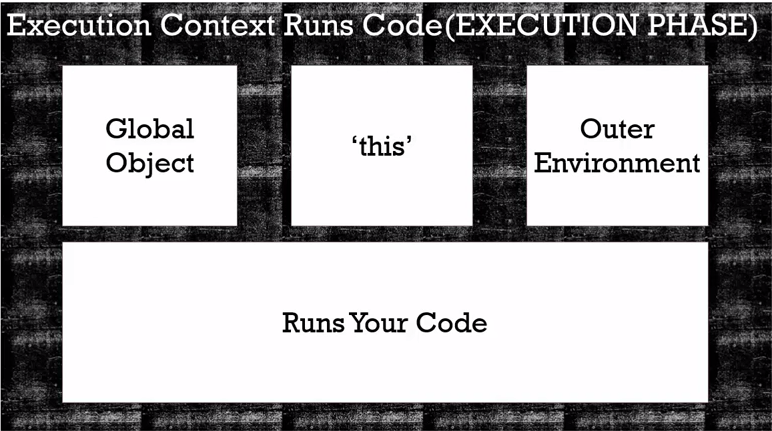






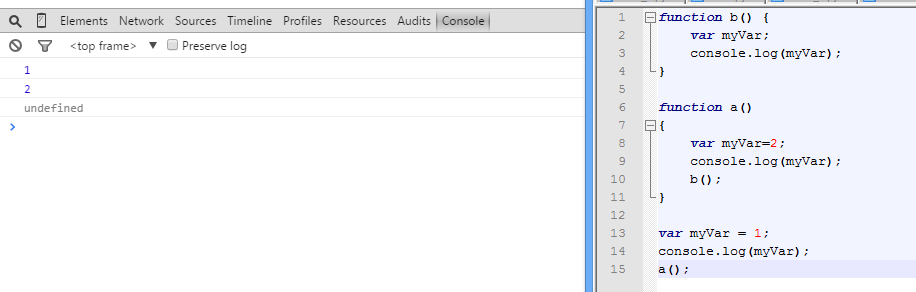


JavaScript makes a placeholder and defines all the variables and functions as ‘undefined’ before its execution.



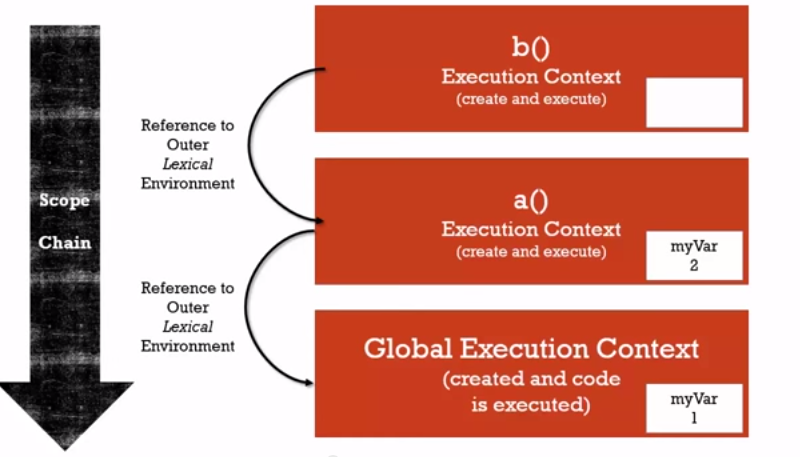


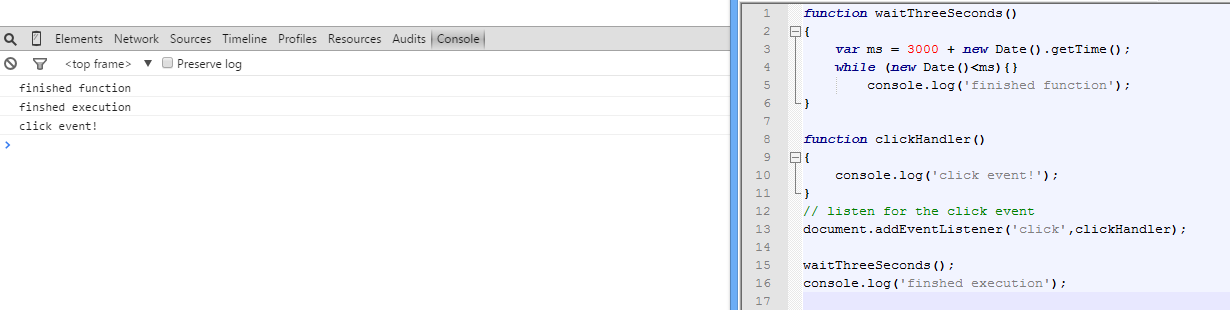
Execution Stack (for function)



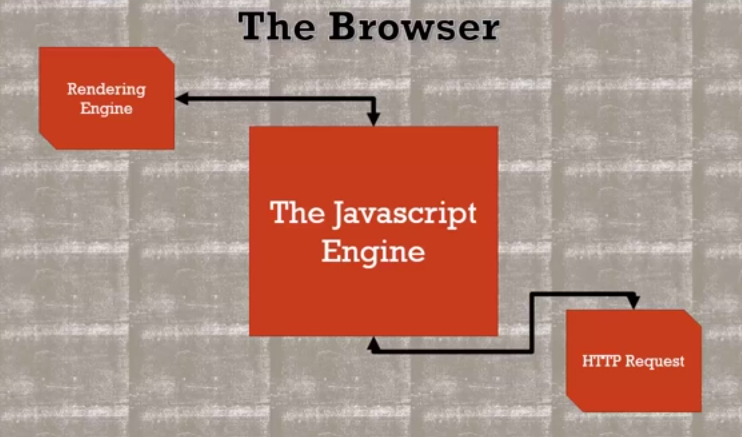


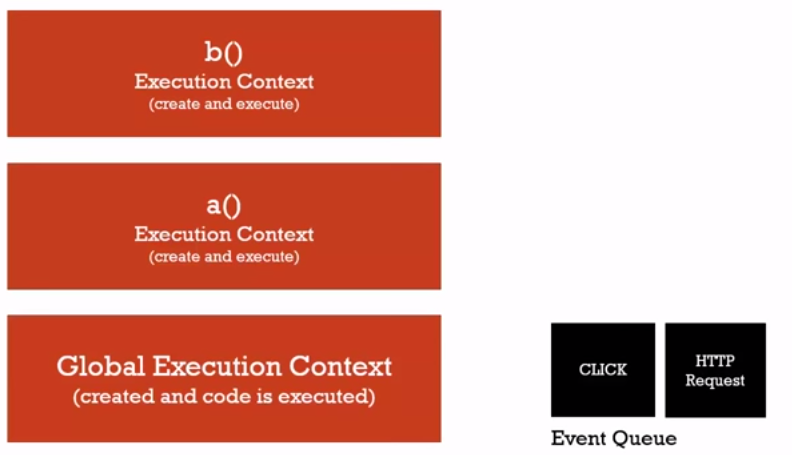


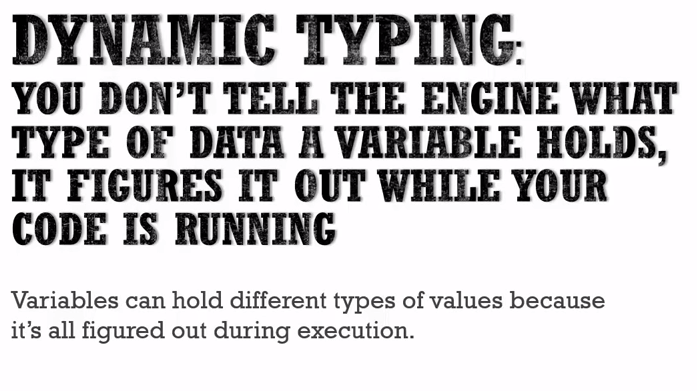
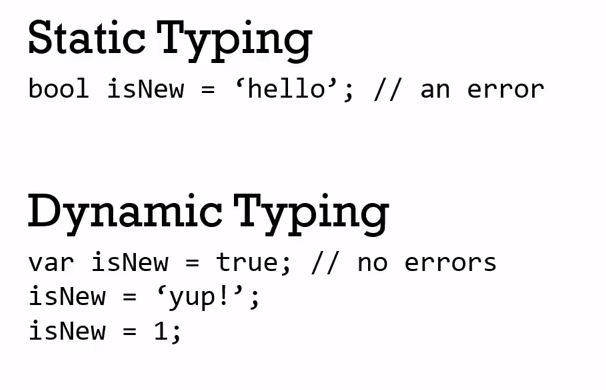




After completing the Execution Stack it looks in the Event Queue.







JS

Other Programming Languages

JS

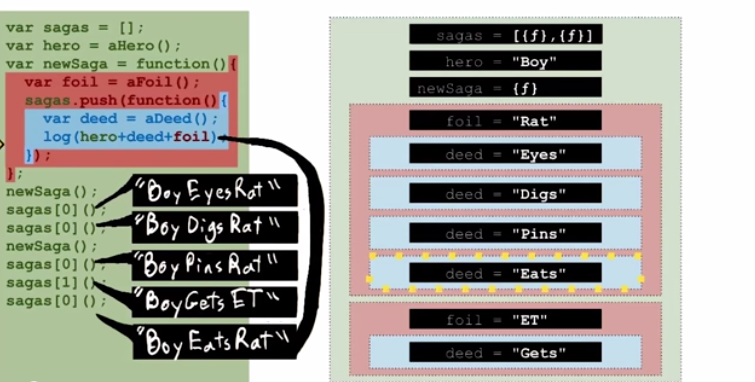
**Chapter 2 : Scopes & Closures**

`

1. Global Scope
2. Lexical Scope
3. Execution Context

Closure: Remains available to all the functions.

* Passing
* Return
* Global save



**Chapter 3: This Keyword**

This:The object found to the left of the dot where the containing function is called

If - object.method(a,y);

so, this = object value aka focal object

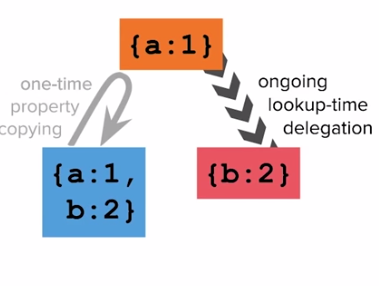
If a function is defined in the global scope, this would refer to the global scope i.e. its lexical scope.

Function.call(r,x,y) 🡪 this would get bind to r

Object.method.call(r,x,y) : this 🡪 r

**Chapter 4 : Prototype Chain**

Its inheritance.



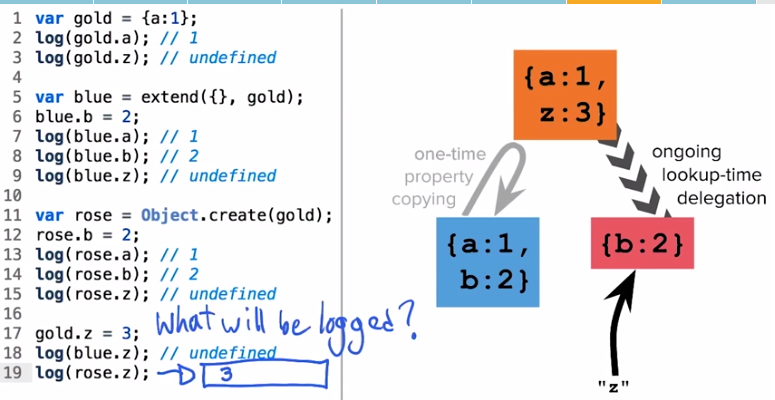
```

Var gold = {a:1};

Var blue = extend({},gold); //copying all aka one time property copying

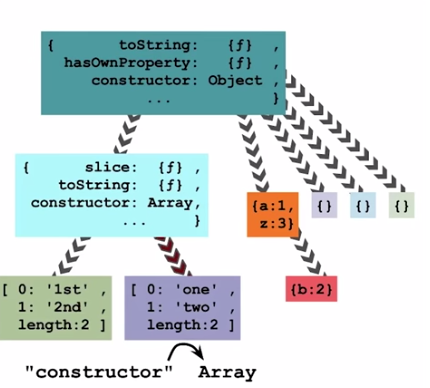
Var rose = Object.create(gold); //linkage aka lookup

```

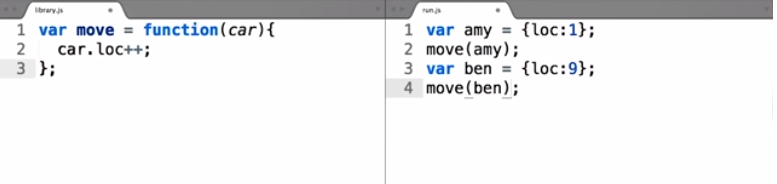


Object Prototype:

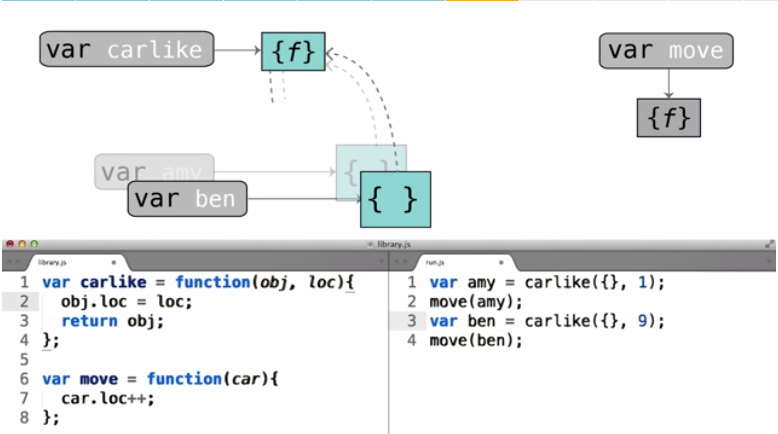
Rose.toString();

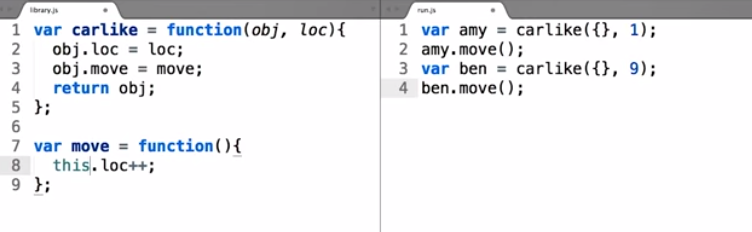


**Chapter 5 : Object Decorator Pattern> Code Reuse**

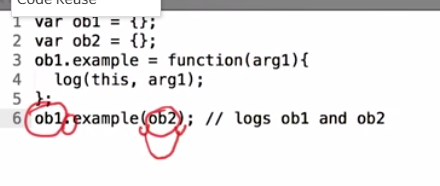


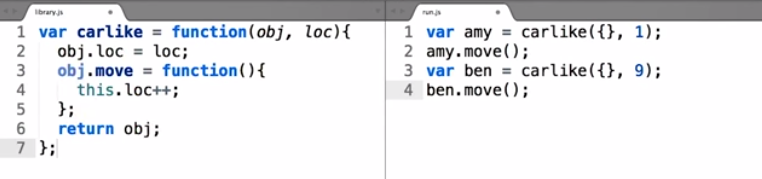
Decorator Function- Naming Convention: Camel Case, here it is CarLike:

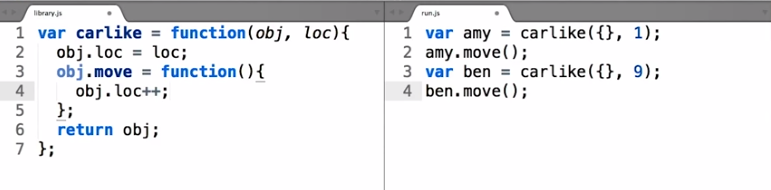




**this:**



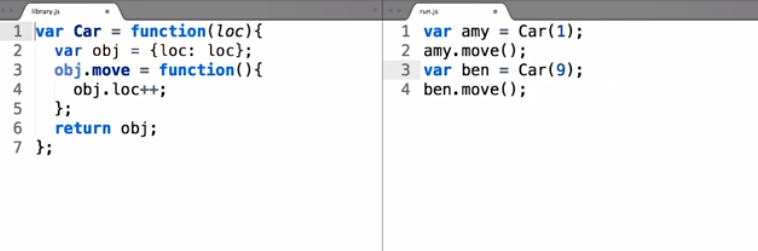




**Chapter 6 : Introduction to Classes**

Difference between decorator function and classes: Decorator function accepts the object which it wants to augments but a class builds the objects which it wants to augments.

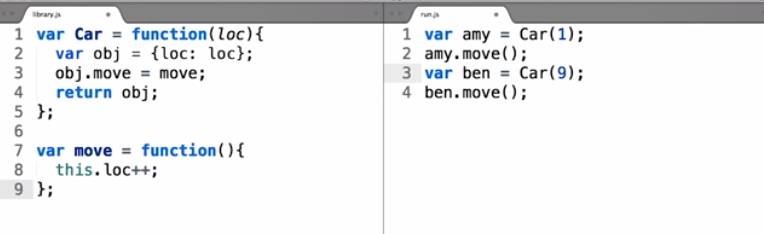
Class: Is a construct that can build a fleet of objects. Naming Convention- Capitalized Noun, Proper Noun



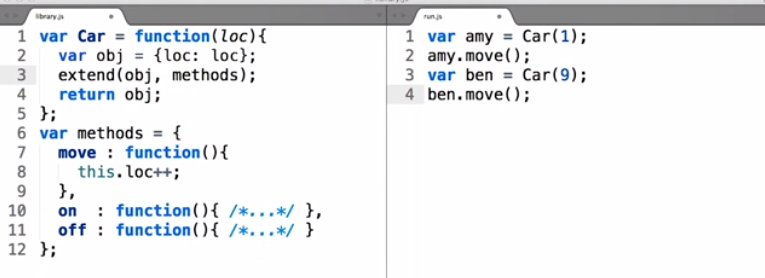
The function of the Class is called a constructor function as it constructs the member of the Class.

The arrow represents instantiating (instance of a class).

To avoid duplicity of creating different object creation for each call.

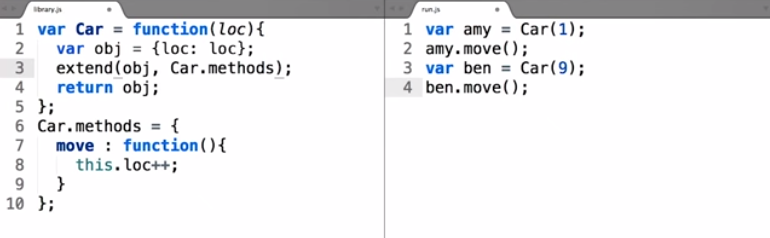


To avoid multiple declaration of a new object inside the class for a new function is better to declare them as object.

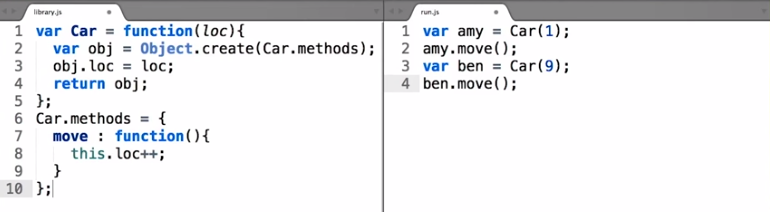


**Note Extend doesn’t work like that refer link:** <http://www.2ality.com/2012/01/js-inheritance-by-example.html>

Using methods as a property of Car:



**Chapter 7: Prototypal Class**

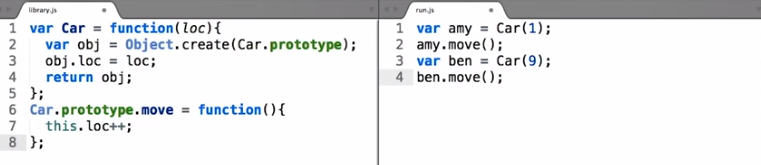


We don’t need to copy properties anymore, prototyping is giving a linkage.

Steps for prototyping:

1. Function that allows to make instances
2. Line in that function that generate the instance object
3. Delegation from new object to some prototype object
4. And some logic to augment that object to make it unique from all other objects of same class.

Since it’s so common, JavaScript creates a default container attached to an object whenever a function is created



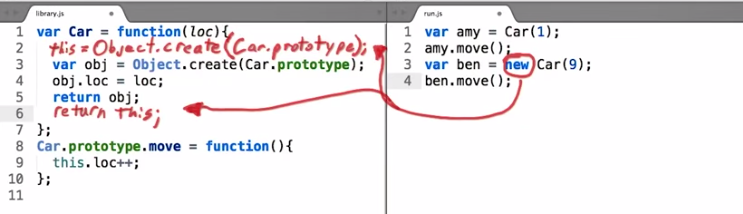
Don’t get confused with the keyword prototype, it doesn’t literally acts as a prototype but as a reference call.



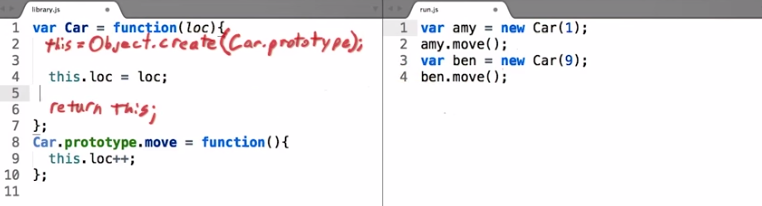
**Chapter 8: Pseudo Classes**

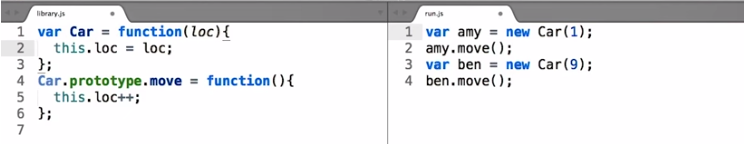
JavaScript doesn't have the traditional "classes" that lower-level languages like C++ and Java have. Instead, JavaScript does some tricks to allow you to write code as though it had these traditional classes. We call these "pseudo-classes".

Use of new operand generate the extra code:



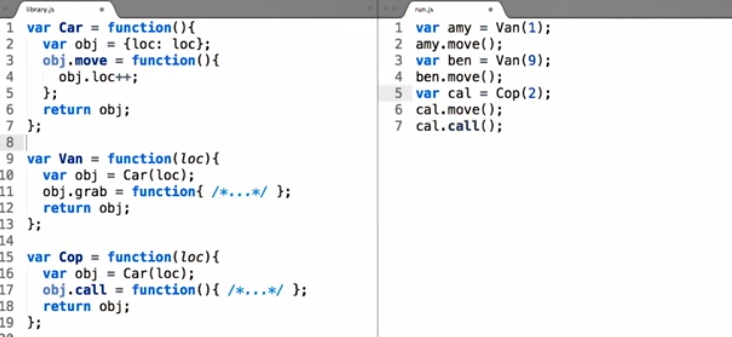
So finally the code turn to:





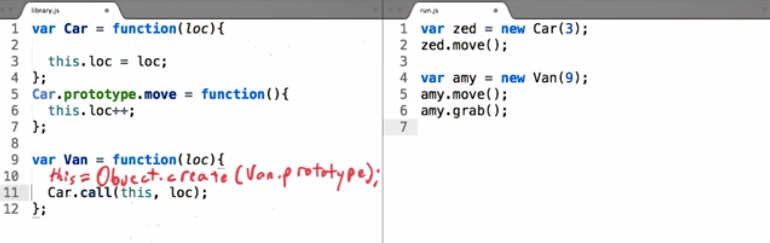
**Chapter 9: Super Classes and Subclasses**

"Subclass" one object to another: This will give our new object the attributes of the original object. It will allow us to make further modifications to the new object without affecting the original object.

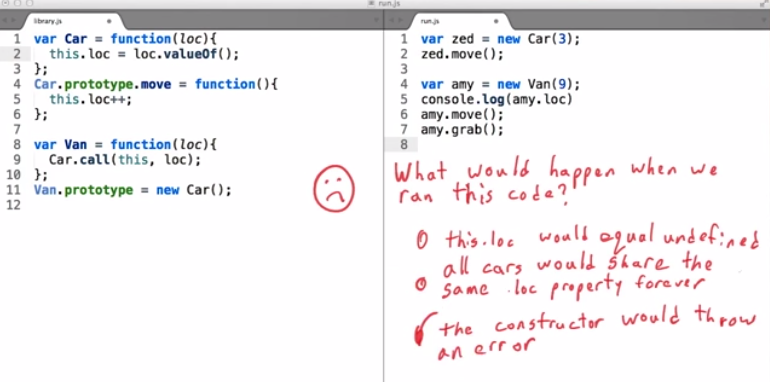


Car here is the Super Class.

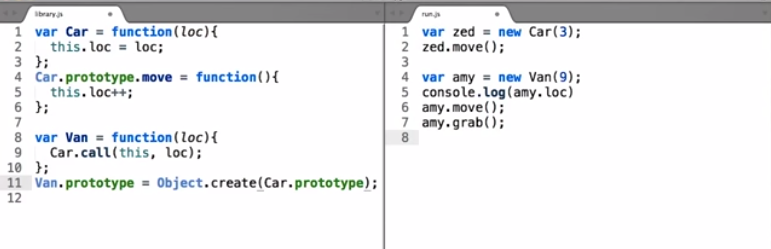
**Creation of subclasses from pseudo classes:**



Below is an incorrect definition:

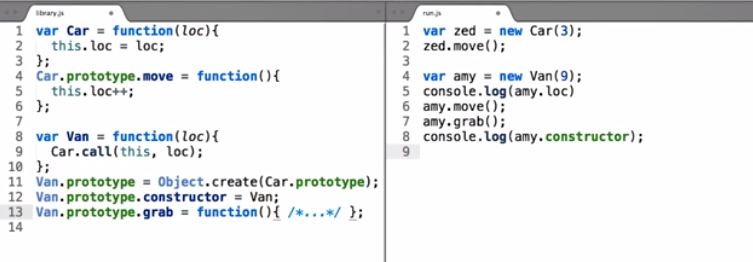


Correct definition:



Now amy.move() can instantiate from Car.prototype.move

Final code:



While we created a new Object. Create it destroyed the default .constructor prototype and its .constructor so new we had to create it again.

**Useful Link:** http://eloquentjavascript.net/