Q=what is closer?

<https://www.w3schools.com/js/js_function_closures.asp>

## Ans=JavaScript Closures

Remember self-invoking functions? What does this function do?

Example

var add = (function () {  
    var counter = 0;  
    return function () {return counter += 1;}  
})();  
  
add();  
add();  
add();  
  
// the counter is now 3

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_function_counter3)

## Example Explained

The variable **add** is assigned the return value of a self-invoking function.

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.

This is called a JavaScript **closure.** It makes it possible for a function to have "**private**" variables.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

Q=how to extend object in javascript?

Ans = using prototype

The first thing that we have to do is to create a class. We will start with the constructor:

var BaseClass = function() {

this.name = "I'm BaseClass";

};

Nothing special, just declaring of a function. The body of the class is constructed by using .prototype property of our variable.

var BaseClass = function() {

this.name = "I'm BaseClass";

};

BaseClass.prototype = {

getName:function() {

return this.name;

},

setName:function(str) {

this.name = str;

}

};

So far we have:  
a) a class that is called BaseClass  
b) a public property called "name"  
c) two public methods called "getName" and "setName" that will get/set the "name" property  
And the usage of the class:

var base = new BaseClass();

alert(base.getName());

Q=how to inject one object in another object?

Ans= using protype