Private properties using Closures -   
JavaScript allows us to create private properties with help of a prefixed “\_” but it doesn’t prevent user from directly accessing or modifying the same. But this problem can be handled very easily with help of closures. For ex – We are given height and width of a person and secret formula we can calculate weight of a person but we don’t want that property to be accessed by any other one.

function PrintWeightFn() {

This.setWeightFn=function(height,weight){

\_weight=height+20+weight;

};

This.printWeightFn=function(){

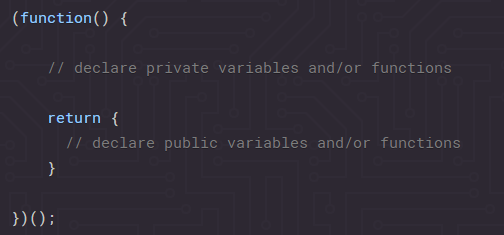
Console.log(\_weight);

}

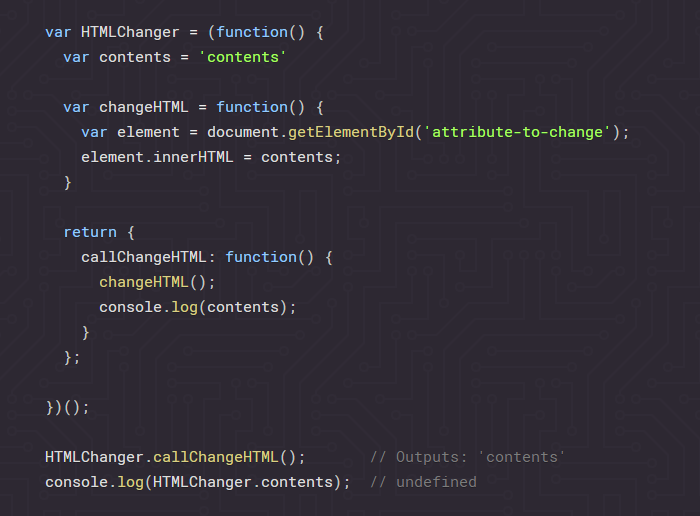
}

Design Pattern in JavaScript: There are mainly four type of design pattern in JS. Module patter, Prototype pattern, Observer Pattern, Singleton pattern.

Module Pattern: JS modules are the most prevailed used pattern for keeping particular pieces of code independent of other. A module should be Immediately Invoked Function expression(IIFE) to allow private scopes – that is a closure that protects variables and methods however it will return an object instead of a function. This is how it looks like

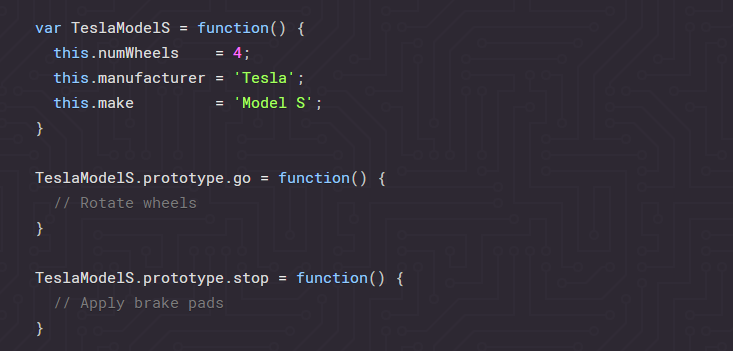


Lets take a concreter example

  
  
Here we can see that contents can’t be accessed directly but can be accessed by the function changeHTML().

Some of the obvious disadvantages of this design pattern are, a.) as we can’t access the private method we hence can’t test them and hence poses unit testing problems. B.) Also the public behaviors are non over ridable.

Prototype Design pattern - The Prototype design pattern relies on the JavaScript prototypical inheritance. The prototype model is used mainly for creating objects in performance-intensive situations.



Singleton Pattern – A singleton only allows for a single installation but many instance of the same object. It restricts client from creating multiple objects, as after the first object is created it will return instance of itself. For example – Using an office printer. If there are 10 people in office and they all use on printer, ten computer share one printer(instance).

var printer = (function () {

var printerInstance;

function create () {

function print() {}

function turnOn() { }

return { print: print, turnOn: turnOn };

}

return {

getInstance: function() {

if(!printerInstance) {

printerInstance = create();

} return printerInstance;

}

};

})();

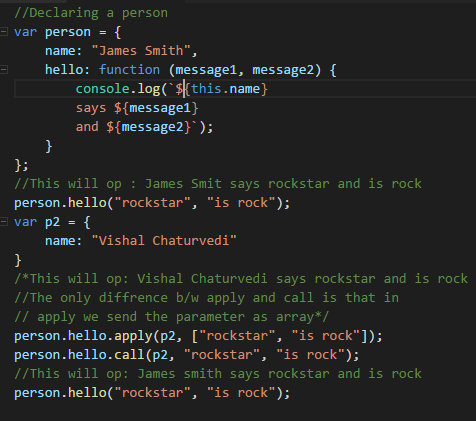
We can see that create method is private for people but every person can access get an instance with help of getInstance function. As we can see there also we are creating the instance only once. In angular JS singleton is very much used in services, factory and providers.   
Some of the disadvantages which can occur with this are a.) Singleton can rarely be prone to conditions like two instances trying to access the instance which will lead to two thread creating two objects. Hence developers must be privy to synchronization when implementing singleton in multithreading application.

apply call and bind methods :

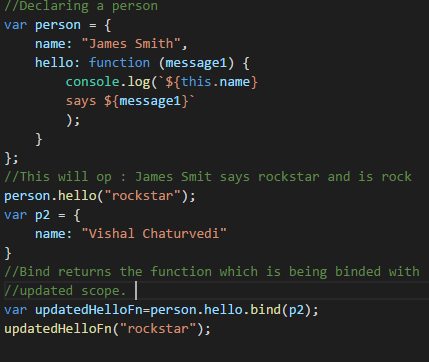
Here we can see How a Call Function works.

Call basically calls the function with for the current object but doen’t change the orignal scope.

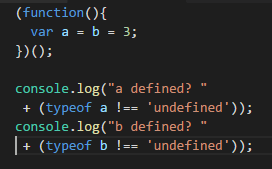
It is same as invoking a function by passing a current context.

The apply is same as call i.e. It is similar to calling a function for a given context.

The only difference b/w apply and call is that we need to pass arguments as array in apply where as in call we can pass them separately.

Bind basically returns the function on which bind has been called with an updated perceptive of the same.

Here we can see in “updatedHelloFn” as we are biding with p2 hence the perceptive of this.name changes from james smith to vishal chaturvedi.

Some of other question:

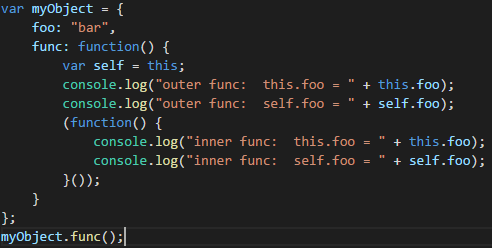
Here mostly developers can misunderstood this statement as equivalent to

Var b=3; Var a=b;

Wherein this is not the case rather it is equivalent to

B=3; var a=b;

And hence b becomes a global variable and hence it is not undefined.

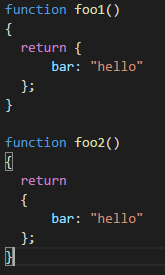
Here in outer function both this and self are referring to myObject and hence this.foo and self.foo there will be equal to bar.

Whereas in IIFE( Immediately invoked function expression) this no more refer to myObject and hence this.foo will be undefined whereas because of the concept of closure self still will be defined and hence self.foo will print bar.

Significance of wrapping up the entire JS in a single wrapper –

This technique creates a closure around the entire contents of the file which, perhaps most importantly, creates a private namespace and thereby helps avoid potential name clashes between different JavaScript modules and libraries.

Opening brackes in new line or same line:

We may consider that these two function will return same thing but this will not happen. Although semicolon is optional in JS but still this case explains why they should be put properly.

In function foo2() as in return statement there is nothing in that line no parenthesis anything hence a semicolon is automatically inserted which lead to foo2() returning undefined and foo1 returning object.

This behavior also argues for following the convention of placing an opening curly brace at the end of a line in JavaScript, rather than on the beginning of a new line.

<https://www.toptal.com/javascript/interview-questions> ( For JS coding question)