JavaScript

# This

## In Anonymous function

* This inside an anonymous function is the reference to global scope. To overcome this there exists 2 method than can be used:

**1)Creating new variable like self, \_this to store this value in outer scope**

loadData: function() {  
 **var self = this;**  
 var dataReceivedHandler = function() {  
 console.log(self.logger);  
 }  
 // more stuff  
}

**2)Changing the context of anonymous function by call or apply**

loadData: function() {  
 var dataReceivedHandler = function() {  
 console.log(this.logger);  
 }  
 // more stuff  
 **dataReceivedHandler.call(this);**   
// by passing this as the first argument we make sure the context of the excuted function is our current scope's this  
**}**

## Different ways for declaring functions

var obj = {  
 foo: function(){  
 alert(this === obj);   
 }  
};  
**obj.foo(); // true**  
  
  
function foo(){  
 alert(this);  
}  
**foo(); // window object  
new foo(); // foo object**

# Closure

## Simple encapsulation

var Module = (function () {

var privateMethod = function () {

};

return {

publicMethod: function () {

// has access to `privateMethod`, we can call it:

// privateMethod();

}

};

})();

## Example

### Calling Closure

function myModule() {

var name = "tim", age = 28;

return function greet() {

return "Hello " + name + ". Wow, you are " + age + " years old.";

}

}

var greeter = myModule(); // call `myModule` to get a closure out of it.

alert(greeter()); // Call the closure

//alert(myModule()()); // another option to call the closure

### Private Function

(function () {

var myFunction = function () {

// do some stuff here

};

})();

//myfunction is private here

### Module Pattern

var Module = (function () {

var myModule = {};

var \_privateMethod = function () {

};

myModule.publicMethod = function () {

};

myModule.anotherPublicMethod = function () {

};

return myModule; // returns the Object with public methods

})();

// usage

Module.publicMethod();

# Call And Apply

## Differences

The **call** function requires the arguments to be listed explicitly while the **apply** function allows you to provide the arguments as an array:

**function** user(firstName, lastName, age){

// do something

}

user.call(window, 'John', 'Doe', 30);

user.apply(window, ['John', 'Doe', 30]);

The result of both calls is exactly the same, the user function is invoked in the context of the window and provided the same three arguments.

First parameter(window) tells the function what is **this** and rest is posted to function as parameters.

## Example

var links = document.querySelectorAll('nav li');

for (var i = 0; i < links.length; i++) {

(function () {

console.log(this);

}).call(links[i]);

}

# CallBack

## Example1

# 

getData('http://fakedomain1234.com/userlist', writeData);

document.getElementById('output').innerHTML += "show this before data ...";

function getData(dataURI, callback) {

var myData = getSomeData(); // fake function

callback(myData);}

function writeData(myData) {

document.getElementById('output').innerHTML += myData;

}

function getSomeData() {

// this would make an XHR connection to the server and get, say, some JSON back

var data = 'this is data from the server';

return data;

}

## Example 2

# 

function getInput(options) {

allUserData.push(options,

(function(){

for (var item in options) {

console.log(item + " : " + options[item]);

}

})());

}

getInput({

name: "Rich",

speciality: "js"

});

# Good Practices

var gameController = {

scores :[20, 34, 55, 46, 77],  
 avgScore:null,  
 players :[  
 {name: "Tomy" , playerID:987, age:23},  
 {name: "Pau" , playerID:87, age:33}  
 ]

}

var appController = {  
 scores :[900, 845, 809, 950],  
 avgScore:null,  
 avg: function () {  
 var sumOfScores = this.scores.reduce (function (prev, cur, index, array) {  
 return prev + cur;

});  
 this.avgScore = sumOfScores / this.scores.length;  
 }  
 }

//If we run the code below,

// the gameController.avgScore property will be set to the average score from the appController object scores array

// The avg method "this" keyword will not refer to the gameController object, it will refer to the appController object because it is being invoked on appController

// Don't run this code, we want the appController.avgScore to remain null

appController.avg.apply (gameController, gameController.scores);

// The avgScore property was successfully set on the gameController object, even though we borrowed the avg () method from the appController object

console.log (gameController.avgScore); // 46.4

console.log (appController.avgScore);