**Call(), apply() and bind() Methods in JavaScript.**

**Basic rules worth remembering:**

1. “this” always refers to an object.
2. “this” refers to an object which calls the function it contains.

Ex.:

var car = {

registrationNumber: "GA12345",

brand: "Toyota",

displayDetails: function(){

console.log(this.registrationNumber + " " + this.brand);

}

}

The above will work perfectly fine as long as we use it this way:

car.displayDetails(); // GA12345 Toyota

But what if we want to borrow a method?

**The bind() Method**

For such cases we can use bind() method of the Function.prototype property. This means bind() can be used by every single function.

var myCarDetails = car.displayDetails.bind(car);

myCarDetails(); // GA12345 Toyota

The bind() method creates a new function where “this” refers to the parameter in the parenthesis in the above case “car”. This way the bind() method enables calling a function with a specified “this” value.

What if we would like to pass a parameter to the displayDetails function? We can use the bind method again. The following argument of the bind() method will provide an argument to the function bind() is called on.

Let me rewrite the car object:

var car = {

registrationNumber: "GA12345",

brand: "Toyota",

displayDetails: function(ownerName){

console.log(ownerName + ", this is your car: " + this.registrationNumber + " " + this.brand);

}

}

var myCarDetails = car.displayDetails.bind(car, "Vivian"); // Vivian, this is your car: GA12345 Toyota

**Another Ex:**

var obj = {name:"Ritik"};

var greeting = function(a,b,c){

return "welcome "+this.name+" to "+a+" "+b+" in "+c;

};

var bound = greeting.bind(obj);

console.log(bound("Valtech","INDIA","JP Nagar")); //call the bound function

/\* the output will be

welcome Ritik to Valech INDIA in JP Nagar \*/

**call() and apply() methods**

Similar but slightly different usage provide the call() and apply() methods which also belong to the Function.prototype property.

This time there is a car object without the displayDetails function, which is located in the global context.

var car = {

registrationNumber: "GA12345",

brand: "Toyota"

}

function displayDetails(ownerName) {

console.log(ownerName + ", this is your car: " + this.registrationNumber + " " + this.brand);

}

**Using apply() and call() function:**

displayDetails.apply(car, ["Vivian"]); // Vivian, this is your car: GA12345 Toyota

Or

displayDetails.call(car, "Vivian"); // Vivian, this is your car: GA12345 Toyota

**Another Ex:**

var obj = {name:"Ritik"};

var greeting = function(a,b,c){

return "welcome "+this.name+" to "+a+" "+b+" in "+c;

};

**Using apply() and call() function:**

console.log(greeting.call(obj, "Valtech", "INDIA", "JP Nagar"));

/\* the output will be

welcome Ritik to Valech INDIA in JP Nagar \*/

OR

var arr = ["Valtech","INDIA","JP Nagar"];

console.log(greeting.apply(obj, arr));

/\* the output will be

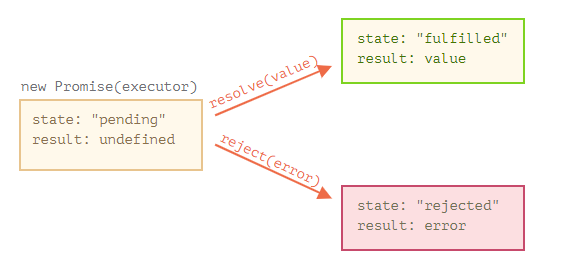
welcome Ritik to Valech INDIA in JP Nagar \*/

**Promise in JavaScript**

A Promise is an object which rejects a single value whether in resolve or reject state.

Promise returns an object in three different states.

1. Resolved
2. Reject
3. Pending

There can be only a single value, either resolve or reject.

Ex:

Const promise = new Promise((resolve, reject) => {

if(true) {

resolve(“ working !!”);

}else{

Reject(“Error it’s not working”);

}

})

Promise.then(result => console.log(result))

**Closure in JavaScript**

A closure is formed when you nest functions, inner functions can refer to the variables present in their outer enclosing functions even after their parent functions have already executed.

Ex :

Const closure = function() {

Let count = 0;

return function increment() {

count++;

return;

)

}

Const incremenfn = closure()

incrementfn()

Incrementfn()

Ex :

function sayHello() {

var say = function() { console.log(hello); }

var hello = 'Hello, world!';

return say;

}

var sayHelloClosure = sayHello();

sayHelloClosure(); // ‘Hello, world!’