Name :Mayur Purushvani

Function returning :

The function returns another function as an argument.

We have one generic function and used of that function we can return another function like below example :

```
function InterviewQuestions (job)
{
   if(job === 'teacher')
   {
      return function(name)
      {
         console.log("What subject do you teach "+ name + " ?");
      }
   }
   else if (job === 'designer')
   {
      return function(name)
      {
         console.log("What is UI designing "+name + " ?");
      }
   }
   else
   {
      return function (name)
      {
         console.log("What do you do "+ name + " ?");
      }
   }
}

var teacherQuestion = InterviewQuestions('teacher');
teacherQuestion('mayur');

//OR

InterviewQuestions('teacher')('mike');
InterviewQuestions('designer')('jhon');
InterviewQuestions('developer')('ellon');
```

We can call the function in 2 ways :

Like I've mentioned above.

Immediately Invoked Functions Expressions (IIFE) :

The above function is simple, while the below function is called as IIFE
Immidiate Invoke Function Expression.
The above function is known as simple function declratation but we don't need to only declare the function but we need the function as an expression way. Syntax must follow..!

```
function game()
{
   var score = Math.random() * 10;
   console.log(score >= 5);
}
game();


/*————————————————————*/

(function (goodluck) {

   var score = Math.random() * 10;
   console.log(score >= 5 - goodluck);
})(5);
```

IIFE also helps in data privacy.

Closures :

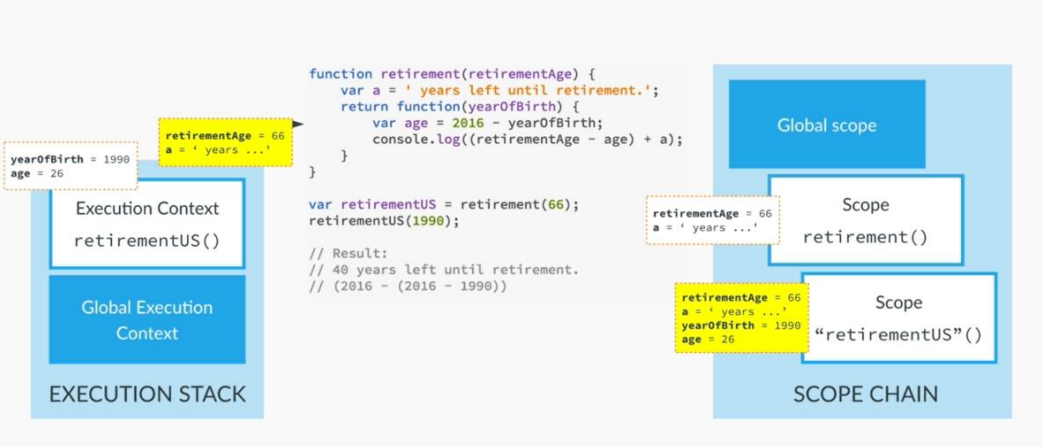
An inner function has always access to the variables and parameters of its outer functions, even after the outer function has returned.

Let's take an example :

```
function retirement(retirementAge)
{
   var a = " years left until retirement";
   return function(yearOfBirth)
   {
      var age = 2020 - yearOfBirth;
      console.log((retirementAge - age )+ a);
   }
}
var retirementInIndia = retirement(65)(1999);
```

The outer function retirement has returned but also the inner function can access that parameters or variables. This is known as clouser.

How it’s work ? :


function retirement(retirementAge) {
var a = ' years left until retirement.';
return function(yearOfBirth) {
var age = 2016 - yearOfBirth;
console.log((retirementAge - age) + a);
}
}
var retirementUS = retirement(66);
retirementUS(1990);
// Result:
// 40 years left until retirement.
// (2016 - (2016 - 1990))
retirementAge = 66
a = ' years ...'
yearOfBirth = 1990
age = 26
Execution Context
retirementUS()
Global Execution Context
EXECUTION STACK
Global scope
retirementAge = 66
a = ' years ...'
Scope
retirement()
retirementAge = 66
a = ' years ...'
yearOfBirth = 1990
age = 26
Scope
“retirementUS”()
SCOPE CHAIN

Bind, Call and Apply :

```
        var mayur = {

    name : 'mayur',
    age : 22,
    job : 'teacher',
    presentation : function (style, timeOfDay){
        if(style === 'formal')
        {
            console.log('Good ' + timeOfDay + ' ladies and gentalman ' + 'I\'m '+ this.name);
        }
        else if(style === 'friendly')
        {
            console.log('Hey! What\'s up? I\'m '+ this.name + ' Good ' + timeOfDay + ' everyone!');
        }
    }
}
var mike = {
    name : 'mike',
    age : 22,
    job : 'designer'
};

mayur.presentation('formal', 'morning');
mayur.presentation('friendly','evening');

mayur.presentation.call(mike,'formal','morning');

mayur.presentation.apply(mike,['formal','morning']);
/*This will not work in this code because our code doesn't
get any array argument. but this is only for your reference that the 'apply' is called something lik
e this.*/

var mayurFriendly = mayur.presentation.bind(mayur,'friendly');
mayurFriendly('morning');
mayurFriendly('night');
```

The bind method is used for bind something In our function and that will also be used further.

The call method is used when we create a generic function and we can also declare another variable function like above example, so mayur has some functions and below ive created one other variable called ‘mike’. So I can also access or call the variables and methods of mayur’s function.

The apply syntax is given above. This will not work in this code. Because apply need an aray argument. So, We don’t have any array argument in our code.

The call method has 3 argument, one is ‘this’. So, Which function do you want to call or execute. Then second and third argument is ‘style’ and ‘timeOfDay’.

JavaScript Versions :

ES6 / ES7 / ES8

- Well supported In all modern browsers.
- No support in older browsers.
- Can use most features in production with transpiling and polyfilling (converting to ES5)

ES6 new Features :

- Variable Declaration with let and const
- Blocks and IIFEs
- Strings
- Arrow Functions
- Destructuring
- Arrays
- The Spread Operator
- Rest and Default parameter
- Maps
- Classes and subclasses

Variables Declration with let and const :

- A let and const is a block scope.
- In ES5 we can access an variables inside the functions, but in ES6 with let and const we cannot access the variables inside the functions. It is a block scope.
- Like below example :

```
//ES5 variables
var name = 'mayur';
var age = 22;
name = 'mike';
console.log(name);


//ES6 variables
const name1 = 'mayur';
let age2 = 22;
//name1 = 'mike';
console.log(name1);
```

```
//ES5 functions
function drivesLicece(passedTest)
{
   if(passedTest)
   {
      //console.log(firstName); //This will retuen undefined.
      var firstName = 'mayur';
      var yearOfBirth = 1999;

   }
   //console.log(firstName + ' born in ' + yearOfBirth + ' is now officially allowed to drive a car.');
   //ES5 is a function block so, it will work fine
}
drivesLicece(true);

//ES6 functions
function drivesLicece6(passedTest)
{
   if(passedTest)
   {
      //console.log(firstName); //This will return an ERROR.
      let firstName = 'mayur';
      const yearOfBirth = 1999;

   }
   //console.log(firstName + ' born in ' + yearOfBirth + ' is now officially allowed to drive a car.');
//
   //ES6 has block scope so it will not allowed to access the variables outside the block scope.
ERROR

}
drivesLicece6(true);


//Let's take another example of block scope in ES6

let i = 23;
for ( let i = 0; i < 5; i ++)
{
   console.log(i);
}
console.log(i); //This value remain same which is 23.
//But when i remove the let from for loop, Te scope of i is only one which is 1 2 3 4 5 .
```

Blocks and IIFEs :

A block is defined as :
{
}

Let's take an example of block scope :

```
{
   const a = 1;
   let b = 2;
}
console.log(a + b); //Error
```

```
(
   function () {
      var c = 3;
   })();
console.log(c);  //Error
```

This will also gives an error. Because
The 'var' clause is function scoped. So when we declare var, we can access that variable inside whole function scope. But the 'let' and 'const' is the block scope. So when {},(),[] brackets are closed, then that variables scope is also removed or closed.

Strings in ES6 :

We have a too many Strings methods in ES6 like log, repeat, startsWith, endsWith, includes, etc.
Let's look at the below example:

```
let firstName = 'mayur';
let lastName = 'purushvani';
const yearOfBirth = 1999;
function calcAge(year)
{
   return 2020 - year;
}

//ES5
console.log("This is " + firstName +" " + lastName + " I was born in " +  yearOfBirth + " and i'm "
+  calcAge(yearOfBirth) + " years old!");

//ES6
console.log(`This is ${firstName} ${lastName} I was born in ${yearOfBirth} and i'm ${calcAge(ye
arOfBirth)} years old`);
//Use back ticks..


const m = `${firstName} ${lastName}`;

console.log(m.startsWith('m'));
//When i search 'M' Capital, it will return false because javascript is case sensitive.

console.log(m.endsWith('ni'));

console.log(m.includes('yu'));
//search from string that it will exists or not!

console.log(`${firstName} `.repeat(5));
//Repeat the string as many times you want!
```