2.

PAGE NO :
DATE : / /

Scopes and Closures In-depth.

a. leverage the Code Quality
b. debug quicker

Unit 01
Understanding Scopes

Unit 02
Compilation and Interpretation

Unit 03
Closures

Material Recommended
1. You don't know JS
by kyle Simpson
[free]

PAGE NO :

DATE : / /

//function declaration

Function foo {

    Console.log ("function foo called");

}

//function Expression

[first-class citizen]

Var bar = Function () {

    Console.log ("function bar called");

};

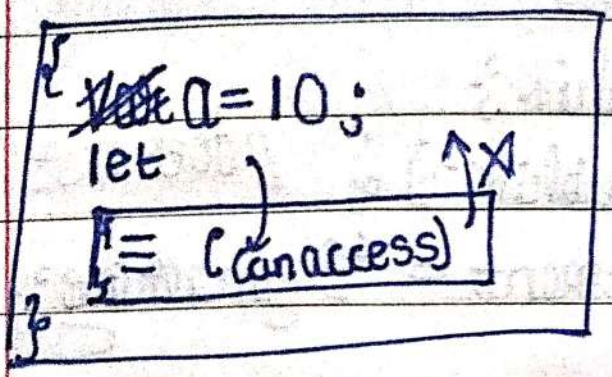
foo();

bar();

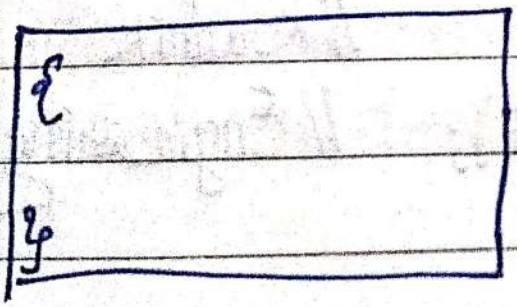
PAGE NO :
DATE : / /

What is Scope ?

Scope Contains a portion of program where particular var is accessible. Outside Scope it don't exist



What inside the box is not accessible Outside the box



Bunch of way to Create block

① if else

etc

but {} does not Create Scope here in JS. for var

We assume we are only using var

PAGE NO:
DATE: / /

* Blocks does not Create Scope, function does. (Function Scoping)

```
Var name = 'Koushik';
if (name == 'Koushik') { // does not Create S
    Var department = 'Engineering';
}
Console.log (name);          // Koushik
Console.log (department);    // Engineering
```

there are more addition to JS, to Create Scope but For most cases our assum will Work. (var)

PAGE NO :
DATE : / /

Code2

```
var name = "Koushik";

function allocateDepartment(){
    if(name == "Koushik"){
        var department = "Engineering";
    }
}

allocateDepartment();
console.log(department); // Error
```

Create Scope

Whatever out is acc look inside than outside

// Error
acc out of scope

Scope Excercise

```
Var top=10;
function foo(){
    Var inner=20;
    Console.log(inner);
}
```

What will be o·p if we execute above code? Ans. Nothing

Add

```
foo();
```

Ans 20 is Output

Modified Code

```
Var top=10;
Var inner=50;
function foo(){
    Var inner=20;
    Console.log(inner);
}
foo();
```

Output

20.

PAGE NO :
DATE : / /

Add below

```
Console.log(inner);
```

Outputs

20
50

```
Var a=10
Var b=10;
Console.log(a+b);
```

← By default global
I don't want
how to modify?

Ans wrap in function

```
function myFn(){
    Var a=10;
    Var b=10;
    Console.log(a+b);
}
myFn();
```

What is O/P?

```
Var name = "Koushik";
function printGreeting (name) {
        console.log (name + " Hello");
}

printGreeting ("Arthur Dent");
```

Output

Arthur Dent Hello.

Hope this help

PAGE NO :
DATE : / /

IIFE

We avoid using global Variable due to multiple files running

```
Var a = 10;
Var b = 10;
Console.log (a+b);
```

global

What if there are 10 files and using same name for global Var.

they must be Overriding Same memory

- better way to wrap around func and make call

```
function myFn () {
    Var a = 10;
    Var b = 10;
    Console.log (a+b);
}
myFn();
```

But here as well we are creating global function myFn.

Same problem

- Better to not give name

```
(function (){
    var a = 10;
    var b = 10;
    console.log(a+b);
})();
```

← Immediately Invoked Function Expression (IIFE)

Read vs Write Operations

```
var a = 10;          // Write Opn
console.log(a);      // Read Opn
```

```
var a = 10;
var b;
b = a;               // Read a, write to b.
```

RHS is Read.
LHS is Write.

PAGE NO :
DATE : / /

```
function greet (name) { //Write Opn
    Console.log (name); //read Opn
}
greet ("Kaushik");
```

Implication of read and Write Opn

① Var foo;
Console.log (foo); //undefined

② //Without declaring (Independent code ahead)
Console.log (foo); // Syntax Error read opp^n without dec

③ //without declaring
Foo = 10; //allowed write Opn (Global Scope Created)
Console.log (Foo);

(Global)

The Window Object

```
Var a = 10;        ← global
Function foo() {
    Var b = 20;    ← not global
}                    Scope to func foo
```

global(root Object) depends on which runtime you are using.

Browser

→ Window Object

↑ that Opens

Whenever you create global variable, its get attached to global Object

```
Var abc = 100;
Var def = "Hello Window Object !!";
```

def or window.def gives same result "Hello Window Object"

PAGE NO :
DATE : / /

```
function myFunc(){
        Console.log("Hello");
}
```

Window.myFun() or myFunc() will gave same result.

Runtime	global Object
node	global.name Object
browser	Window

But according to JS Spec there will be atleast one global Object of runtime

Unit 2

PAGE NO:
DATE: / /

Compilation and Interpretation

↳ Nature of JS

Which is both Compiled and Interpr

JS Code ⟶ Interpretor (Runtime)

- execute directly
- no file in middle

☆ Small Compilation

[Aim:- not to generate middle file, but to look for Signs and note of some things which it needs to execute Program]

Both steps happen Quickly and togather when browser get JS file

JS is both Compiled and Interpreted

L

Compile
looks
at

PAGE NO :
DATE : / /

Understanding the Compilation phase

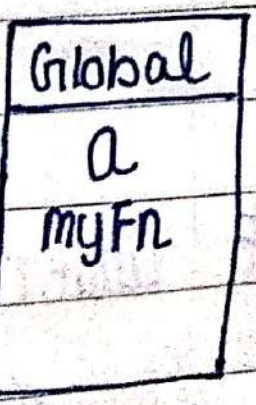
Compiler looks at

```
Var a = 10;
Var b = 20;
Console.log (a+b);
```

Compilation step

Compilation step

```
Var a = 10;
function myFn() {
    Var b = 20;
    Var c = b;
}
myFn();   //40
```



PAGE NO:
DATE: / /

```
var myName = "Koushik";
function greet (name) {
    console.log ("Hello, " + name);
}
greet (myName);
```

Understanding the interpretation Step

```
var myName = "Koushik";
function greet (name) {
    console.log ("Hello, " + name);
}
greet (myName);
```

Compilation Step

Execution Step (based on where it is)

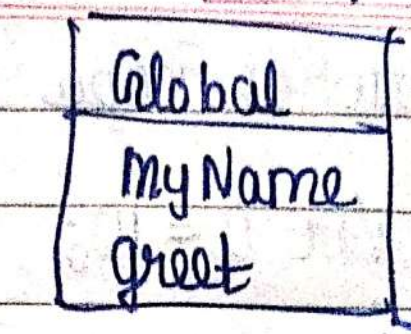
Console is defined in global Scope we Just mentioned things we wrote

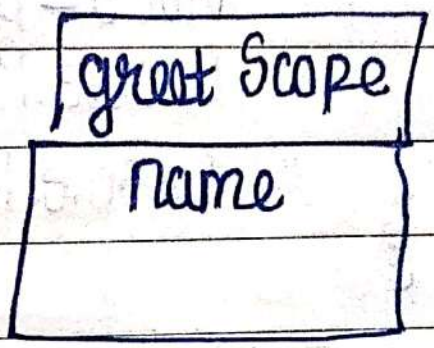
Compilation Step work

```
myName = "Koushik"
greet(myName);

// Executing
This are happening at
global Scope
function greet(name){
  console.log("Hello"+name)
}

Executing at greet func
      Scope
```





~~##~~ Looks from down to up

What if we don't find anything till we reach global Scope? → look at this with our knowledge of Scope chain ahead

(based on where it is)

PAGE NO:
DATE: / /

Global Scope Problem

```
var a = 10;
function myFn() {
    var b = a;
    console.log(b); //10
    console.log(c);
}
myFn();
```

Compiler Step

Global
- a
- myFn

myFn
- b

Interpretation step

```
console.log(c);
```

Hey myFn you have c? no
global you? no
is it read opr
throw error

Case2

```
va
fu
}
my
Co
```

Compilation Step

Case2

```
var a = 10;

function myFn() {
    var b = a;
    console.log(b);
    c = 100;
}

myFn();

console.log(c);
```

Global
a
myFn

myFn Scope
b

Hey myFn You have c? no
global you? no
its write opn
Hence lets Create
it in global Scope

Interp Step

Global
a
myFn
c

myFn Scope
b

PAGE NO :
DATE : / /

~~The Global Scope problem~~

Moral

always use var. to declare variable

Compiler Interpreter

PAGE NO :
DATE : / /

Comp

```
Var a=10;

function outer() {
    Var b=a;
    console.log(b);
    function inner() {
        Var b=20;
        Var c=b;
        console.log(c);
    }
    inner();
}
outer();
```

Hey outer give me b

Hey inner give me b.

Hey inner give me c.

Global
a
outer

Outer Scope
b
inner

inner Scope
b
c

Interpretor Step

10
20.

PAGE NO
DATE : / /

```
Var a = 10;

function outer() {
    Var b = a;
    Console.log(b);
    function inner() {
        Var c = b;
        Console.log(c);
        Var b = 20;
    }
    inner();
}
```

Hey inner → give me b
b is undef →
Set my b to 20. (in chain)

Compiler

Global
a
outer

Outer Scope
b
inner

inner Scope
c
b

Here is

w

w

ex interpretor step

10
undefined

PAGE NO :
DATE : / /

Console.log (a)
Var a = 10;

Interpretor Step
undefined

~~Undefined~~
Compiler Step

Global
a

This behaviour is nothing but Hoisting in JS. We will discuss ahead in detail.

PAGE NO
DATE : / /

Hoisting —

(Knowledge we have about Compilation Step and Interpretation Step, the Concept of Hoisting should make Sense (Self-evident))

Idea is Wherever you write Variable declaration in JS, its almost as if When the interpretor runs Variable declaratin are hoisted to top, its move to top.

↓ Its using that var only at interpretor Step

```
a=10;
console.log(b);
c++;
var a;
var b;
var c;
```

moved upword (top)

```
// a is written with 10
// b undefined
// c hold NAN
```

Compilation Step

a
b
c

PAGE NO :
DATE : / /

Same goes for function

```
myFn1();

function myFn1(){
    ---
}
```

Work fine

Helps in recursive func

```
function recurse(){          //ok
    recurse();
}
```

```
function fnA(){              //ok, due to hoisting
    fnB();
}
function fnB(){
    fnA();
}
```

PAGE NO.
DATE: / /

✡✡

This happens only for function declaration and not for function expression

While execution.

funA(); // funA is undefined

Job of interpretor

Job of Comp

var funA = function () {

}

Remember: A function declaration results in a function object created being created in the compilation step itself

PAGE NO:
DATE: / /

Using Strict mode

Remember:- if you are using a variable that is not declared for a read opn it gives error but if you are using undeclared variable for write opn, its create var in global scope.
(said earlier global var are bad)

Assume I did typo

```
var myName = "";
myname = "kaushik";
```

It create var at global scope in interpreter step. this is bad

you can prevent this behaviour from happening and thats called strict mode.

always refer mdn

PAGE NO
DATE : / /

"use strict"; // first line if you want strict in entire script

var myName = "";

myname = "koushik"; // error [Reference error: assignment to undeclared var]

thats not only thing there are many thing you want JS would have worked like ideally to avoid conf.

function myCode() {
"use strict"; // only this function execute in strict mode.
var myName = "";
myname = "koushik";
}

read on mdn

PAGE NO :
DATE : / /

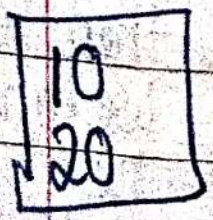
Introducing Closures

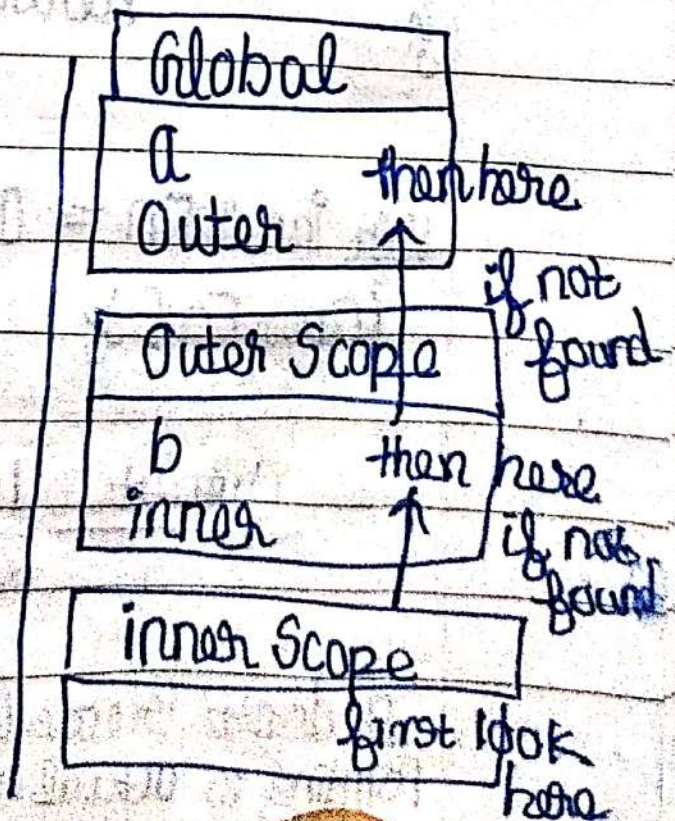
(everything) is easy when Kaushik Sir teaches ~~anything~~.

Code 1

```
Var a=10;
function outer(){
    Var b = 20;
    function inner(){
        Console.log(a);
        Console.log(b);
    }
    inner();
}
outer();
```

ask a to inner





PAGE NO
DATE : / /

Code2 (closure in action)

(even from where used (called) in diff Scope from where it is declared)

```
Var a=10;

Function outer() {
    Var b=20;

    Var inner = function() {
        Console.log(a);
        Console.log(b);
    }
    return inner;
}

Var innerFn = outer();
innerFn();
```

Output 10
20

Once Scope of outer ends then too we can access b. thats closure because of (It form Scope chain)

It don't create copy, its pointing to actual a and b.

PAGE NO :
DATE : / /

Closure in depth

```
Var a=10;
```

every time I run this script a new copy of a is created.

Code

```
Var a=10;
function outer(){
    Var b=20;
    Var inner=function(){
        Console.log(a);
        Console.log(b);
    };
    return inner;
}

Var innerFn=outer();
innerFn();
Var innerFn2=outer();
innerFn2();
```

← a new copy is created each time you make a call to outer

// Note b created won't be garbage collected, as we still have reference to b in inner function. (entire scope chain)

// refer to same a but diff copy of b.

PAGE NO
DATE : /

→ to get real understanding, here is modified Code..

```
Var a = 10;
function outer() {
        Var b = 20;
        Var inner = function() {
            a++;
            b++;
            console.log(a);
        }; console.log(b);
        return inner;
}

Var innerFn = outer();
innerFn();
Var innerFn2 = outer();
innerFn2();
```

Output

```
// 11
// 21
// 12
// 21
```

PAGE NO :
DATE : / /

odified

Closures in Callbacks

JS is Single threaded.

You don't have something like wait(1000)
What you have instead is SetTimeOut( )

```
Var a=10;
Var fn = function () { Console.log(a); }
//wait for 1 second and then execute function
SetTimeOut(fn, 1000);
Console.log("Done");
```

Output

Done
10

PAGE NO
DATE : / /

The Module Pattern (Usecase of closures)

Nothing like public, private methods as in some other language.

```
var person = {
    "firstName" : "Koushik",
    "lastName" : "Kothagal",
    "getFirstName" : function(){
        return this.firstName;
    },
    "getLastName" : function(){
        return this.lastName;
    }
};

person
  {...}

person.getFirstName()
  "Koushik"

person.firstName
  "Koushik"
```

← this is directly accessible

PAGE NO :
DATE : / /

there is a way we can restrict direct access this is what we call a module Pattern

Idea is to use Scope to hide things.

Exactly same thing as above we can access fN and LN

```
function createPerson(){
    var returnObj = {
        "firstName" : "koushik",
        "lastName" : "Kothagal",
        "getFirstName" : function(){
            return this.firstName;
        },
        "getLastName" : function(){
            return this.lastName;
        }
    }
```

```
var person = createPerson();
person.getFirstName();   person.getLastName();
Person.firstName         person.lastName
```

But we can reuse function to create multi Object

```
var person2 = createPerson();
```

PAGE NO
DATE : / /

To Restrict accessibility

```
function createPerson(){
    var firstName = "Kaushik";
    var lastName = "Kothagal";

    var returnObj = {
        "getFirstName": function(){
            return firstName;
        },
        "getLastName": function(){
            return lastName;
        }
    };

    return returnObj;
}

var person = createPerson();
Console.log(person.firstName // undefined
Console.log(person.getFirstName()); // Kaushik
```

need to use Setter and getter

// local function scope

only they can access firstName, lastName

no access directly

Ex→ Add in returnObj.

```
"getFirstName": function(name){
    lastName = name;
}
```

You can add both same way to returnObj.

Remember:- They are working on closure Variable and not working on Obj Property.

```
Var person = CreatePerson();
Console.log(person.getFirstName());
Person.setFirstName("Foo");
Console.log(person.getFirstName());
```

Output

Koushik

Foo.

There is no direct access possible.

* Note:-

Any Variable declared in a function get created everytime the function is called.

PAGE NO :
DATE : / /

Output

10
10
10
:
:
10 (10 was printed 10 times)

Because they all were holding ref to global var i which was modified as loop keep running

after timeout when printed they all printed current val global i was holding i-e 10.

Comment Sec → there is nothing to do with time and speed its about priority. Even if you set setTimeOut(print,0) the print function will always log out last iteration value of i.

This is because the argument func of the setTimeOut (in above case → Print) is always executed at last: Once the entire code of file is executed. (check first comment)

Solving the Problem

~~Var~~ ~~i~~

One way

```
Var i;
for(i=0; i<10; i++){
  (function(){
    var currentValueOfI = i; //Closure
    setTimeOut(function(){
      console.log(currentValueOfI);
    }, 1000);
  })();
}
```

0
1
2
3
:
:
9

Conclusio

PAGE NO :
DATE : / /

Another Way _ take as argument

```
var i;
for (i=0; i<10; i++) {
  (function (currentValueOfI) {
    setTimeOut (function() {
      console.log (currentValueOfI);
    }, 1000);
  }) (i);
}
```

```
0
1
2
3
.
.
.
9
```

Conclusion

Summary

PAGE NO
DATE :

Closures in async callbacks.

```
Var i;
for (i=0; i<10; i++) {
    Console.log (i);
}
```

Output

0
1
2
3
4
5
6
7
8
9

→ Assume I don't want to print directly but rather wait for 1 sec before print.

```
Var i;
Var print = function (){
    Console.log(i);
};
for (i=1; i<10; i++) {
    SetTimeOut( print, 1000);
}
```