

Function Scope V/s Block Scope

We already know about function scope. Everytime we call a function we create a function scope, and we ~~a~~ can only access variables declared inside a function inside it. we can't access them outside the function. Because they're within the function scope



So what's block scope?

Block scope is the scope created inside `{ ... }` (curly brackets)

JavaScript follows ~~block~~ ~~so~~ function scope. And didn't actually have the concept of block scope until ES6 was introduced.

```
if(true)
{
  var name = 'Simran'
}
console.log(name)
```

@codeWithSimran

> Simran

Because if is not a function right? so name is still in global scope and is accessible.

So if we had function sayName()

```
{
  var name = 'Simran'
}
console.log(name)
```

> Reference error

Explanation

Basically when ~~we~~ name is declared inside function a function scope is created and that's why global scope does not have name (it's in function scope)



However, with the introduction of let and const, they also come under block scope.

@codeWithSimran

```
if(true)
```

```
{ var let name = 'Simran' }
```

```
console.log(name)
```

> ~~Simran~~ Reference Error

Because since let and const are block scoped using if statement is like block scope ({ ... }) there those variables are only accessible inside the block scope.

★★ Var still follows function scope.

let and const follow block scope.

→ Therefore in most situations it's better to avoid ~~var~~



Immediately invoked function expressions ~~(IIFE)~~ (IIFE)

Before that let's understand why global variables are bad and then how IIFE can help us :)

→ We already spoke about memory leaks

@codeWithSimran

Okay, if space is the problem, can I use a few global variables?

Let's say we have included multiple script tags in a file

```
<script> var count = 1 </script>
```

```
<script> var count = 2 </script>
```

* Both files have a global variable named "count" so now your count will be overwritten by the latest count value which is 2

THAT'S BAD RIGHT?



As your codebase gets larger
it'll get hard to track these
name collisions.

★ All script tags get combined
to one execution context so
count 'is' used across all
and obviously it can't have
different value for different
files.

@codeWithSimran

IFE will help us here

IFE is a function expression
that works like this

This block says this
is not a function declaration, it's a function
expression

```
(function() {  
  // anonymous func  
  // (no name)  
})();
```

we are immediately
calling this function



Summary

It's an function expression
(we already learnt function
expressions don't get completely
hoisted as IE does not look
at them as function declaration)

That means whatever code
we put inside this function
will not be a part of global
execution context since this
function itself is not part of
global ex

@codeWithSimran



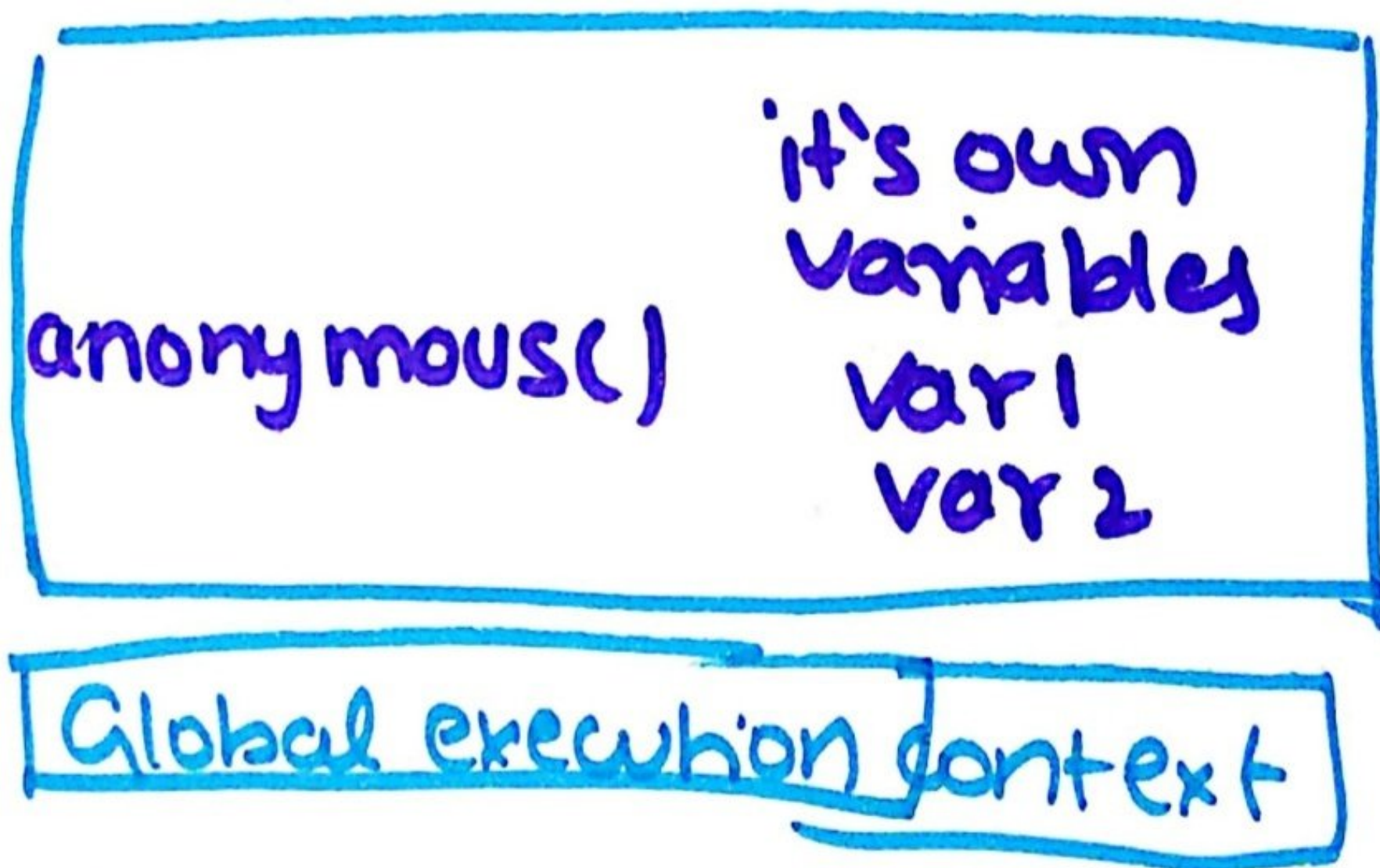
codeWithSimran



codeWithSimran_

So far we know whatever code (variables) we declare inside this IIFE are going to be ~~global~~ local to this function. @codewithsimran

This is how the execution context looks



So a new execution context is created for this anonymous function and it has its own local variables

@codewithsimran



How do we use IIFE to use data (variables or functions) throughout the file?

file.js

@codewithsimran

```
var file1 = (function() {  
  var name = 'Simran'  
  function sayHello() {  
    return 'Simran'  
  }  
  return {  
    name: name,  
    sayHello: sayHello  
  }  
})();
```

★ Since we're immediately invoking 'calling this IIFE file1 variable has the returned data or return value of this IIFE so we can simply do

file1.name

file1.sayHello() etc



But again, we ~~still~~ still have a global variable called `file1` so there is a better solution in modern Javascript called modules we will study about it later.

@codewithSimran

