Compiled

- C
- C++

Interpreted

- bash

Hybrid

- Java
- Python
- JS

Scopes → games

↓

define visibility

In programming also, Scopes define visibility.

It determines where all a variable is visible.

Q→ Is JS interpreted ?? → those lang when exec happens line by line, & execution content has no idea beyond the current line.

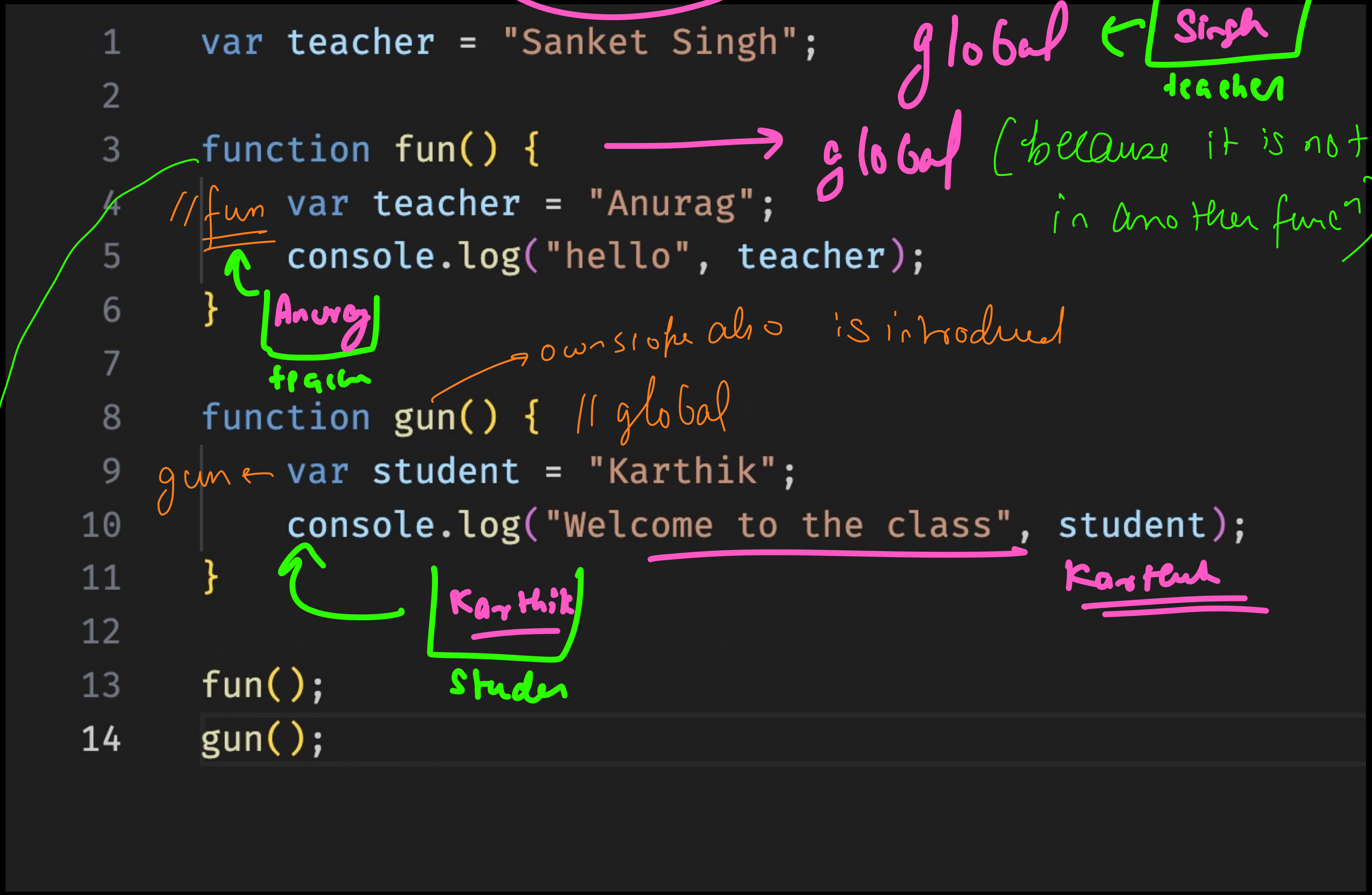
NO → there is a combination of compilation & interpretation.

JS executes your code in 2 phases

- 1st phase → compilation
  - Syntax error
  - Scope resolution
- 2nd phase → execution
  - value assignment to variable
  - code execution

JS → Scope Manager

Phase 1 → no variable gets a value
only scopes are resolved
whenever there is a formal declaration of a variable, it thinks about its scope.



```
var teacher = "Sanket Singh";

function fun() {
    var teacher = "Anurag";
    console.log("hello", teacher);
}

function gun() {
    var student = "Karthik";
    console.log("Welcome to the class", student);
}

fun();
gun();
```

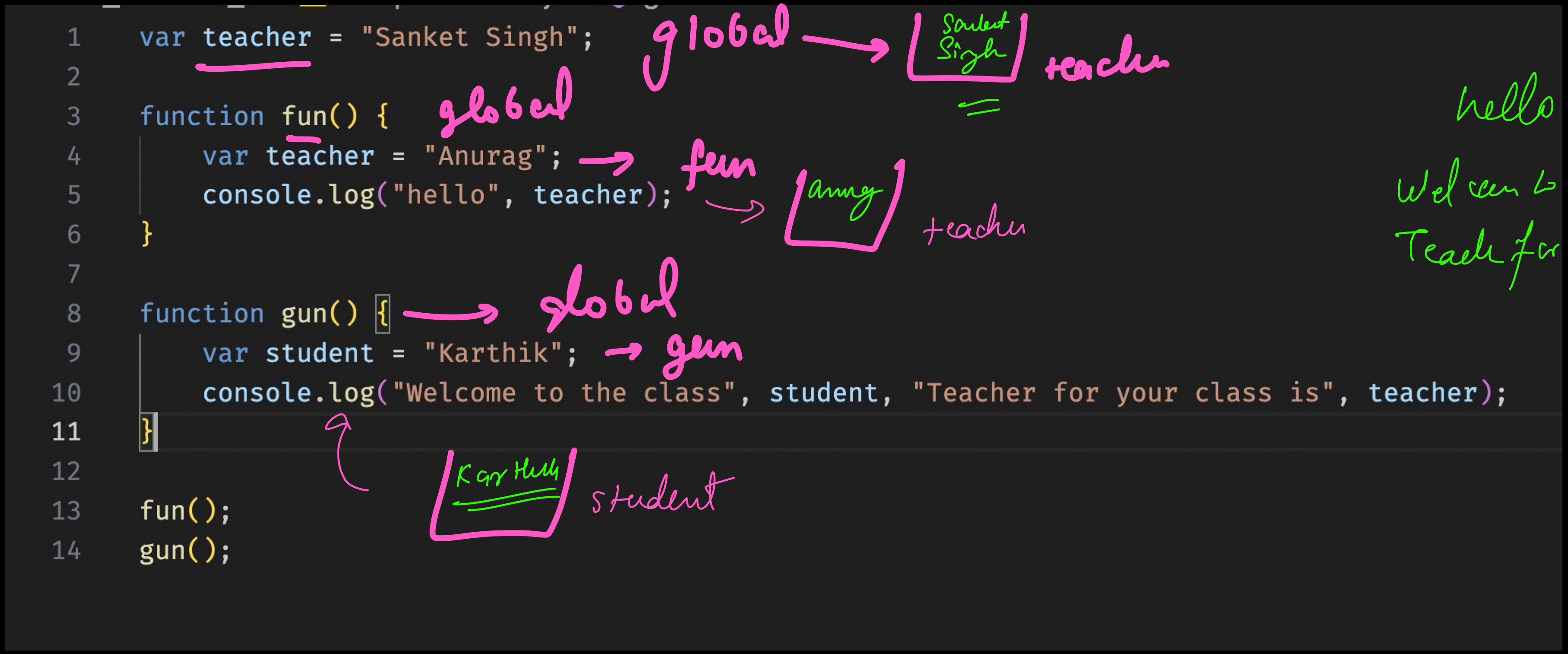
formal declaration } → var x = 10
let y = 2
const z = { }

function

"hello Anurag"

fun introduces it's own scope now

var → function scope (if there is a func)
↳ global scope.



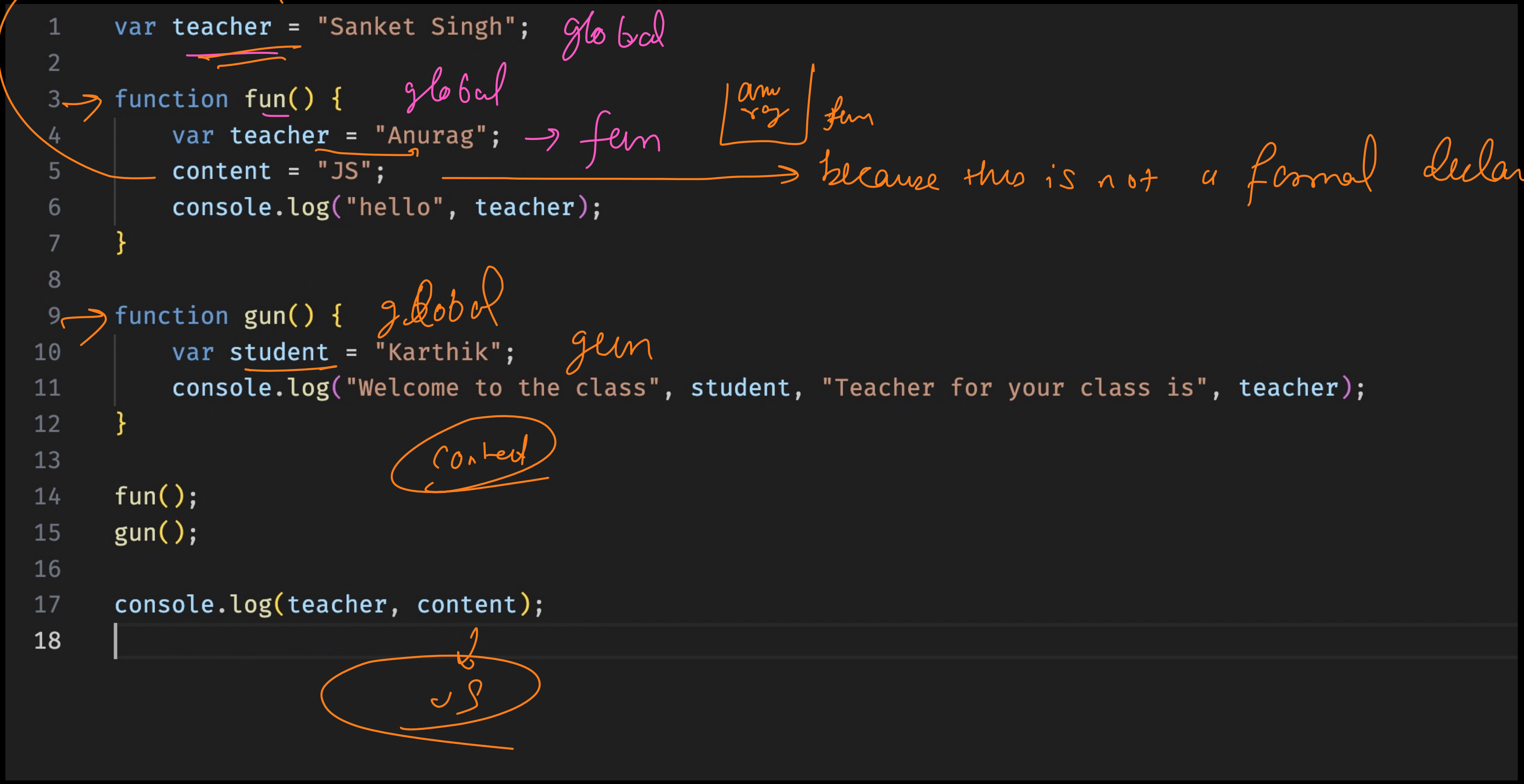
```
var teacher = "Sanket Singh";

function fun() {
    var teacher = "Anurag";
    console.log("hello", teacher);
}

function gun() {
    var student = "Karthik";
    console.log("Welcome to the class", student, "Teacher for your class is", teacher);
}

fun();
gun();
```

hello anurag

Welcome to the class Karthik

Teacher for class is Sanket Singh

Auto Globals



```
var teacher = "Sanket Singh";

function fun() {
    var teacher = "Anurag";
    content = "JS";
    console.log("hello", teacher);
}

function gun() {
    var student = "Karthik";
    console.log("Welcome to the class", student, "Teacher for your class is", teacher);
}

fun();
gun();

console.log(teacher, content);
```

because this is not a formal declaration, So no Scope resolution

Q → How to stop auto global creation

↳ Strict Mode

NOTE

This way of JS to do scope resolution, ahead of time/execution is called as LEXICAL SCOPING (Phase 1 is also called lexical scoping phase)

⇓

Resolving variable scope before execution, is called lexical scoping.

Dynamic Scoping → During runtime we do scope resolution. JS donot support it.

Block Scoping

{

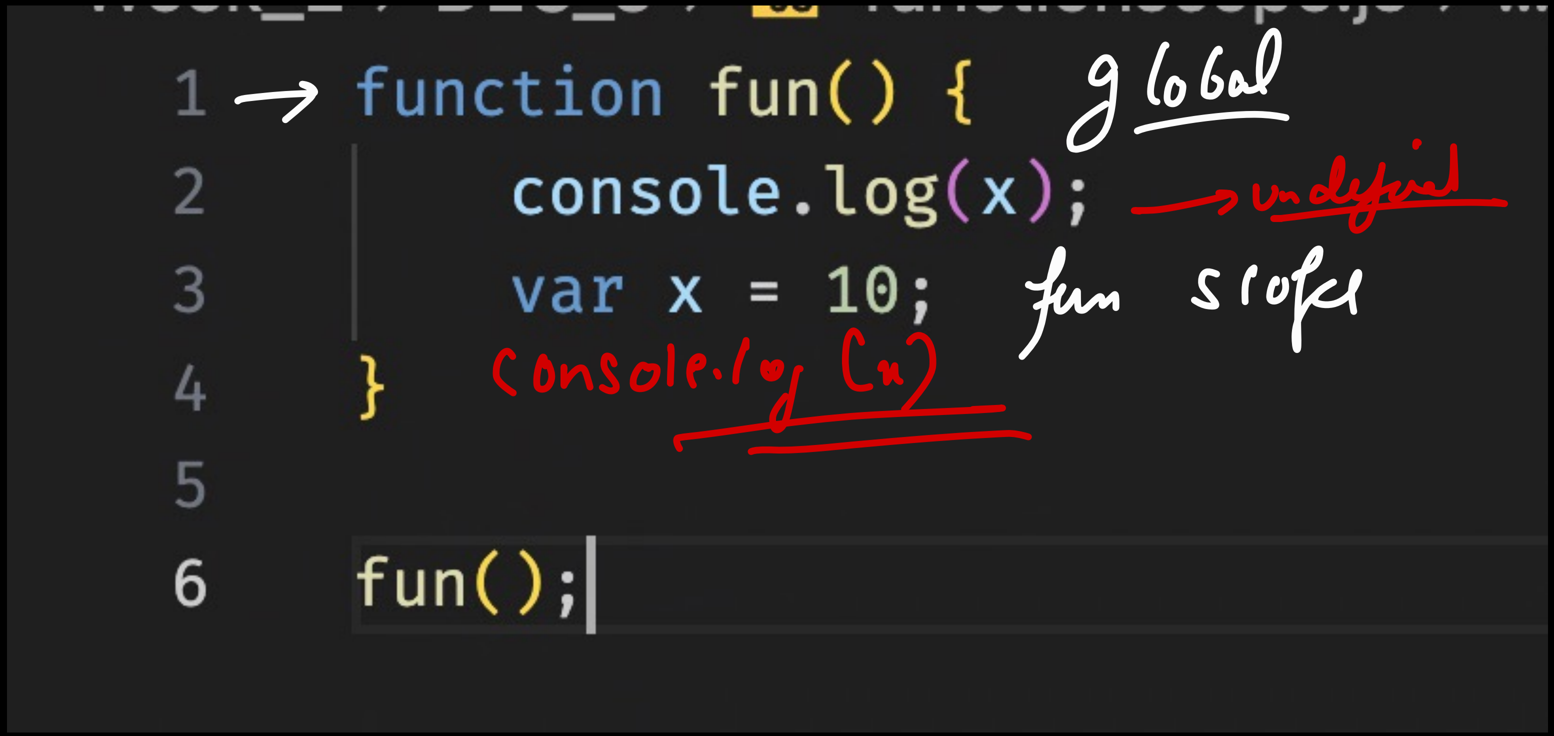
}

pair of curly braces create a new block. except object creation

Block means collection of statement.

what is block scope?

if a variable is only accessible in a block, gets a block scope.



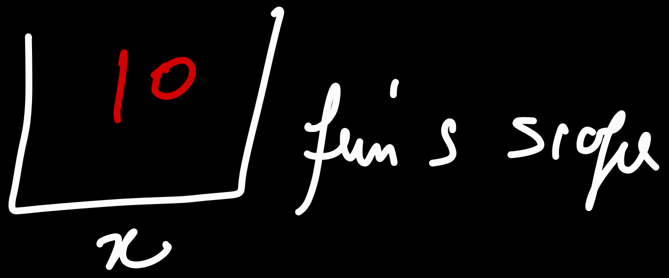
```
function fun() {
    console.log(x);
    var x = 10;
}

fun();
```

global

→ undefined

fun scope

console.log (x)



Temporal Dead Zone (TDZ)

→ It is a term used to describe the state where variables are un-reachable.

When we declare variable using let or const, in the Scope they are they remain in TDZ till the time their declaration hits.

```
if (true) {
    console.log (y); }
    let y = 10;
    console.log(y)
}
```

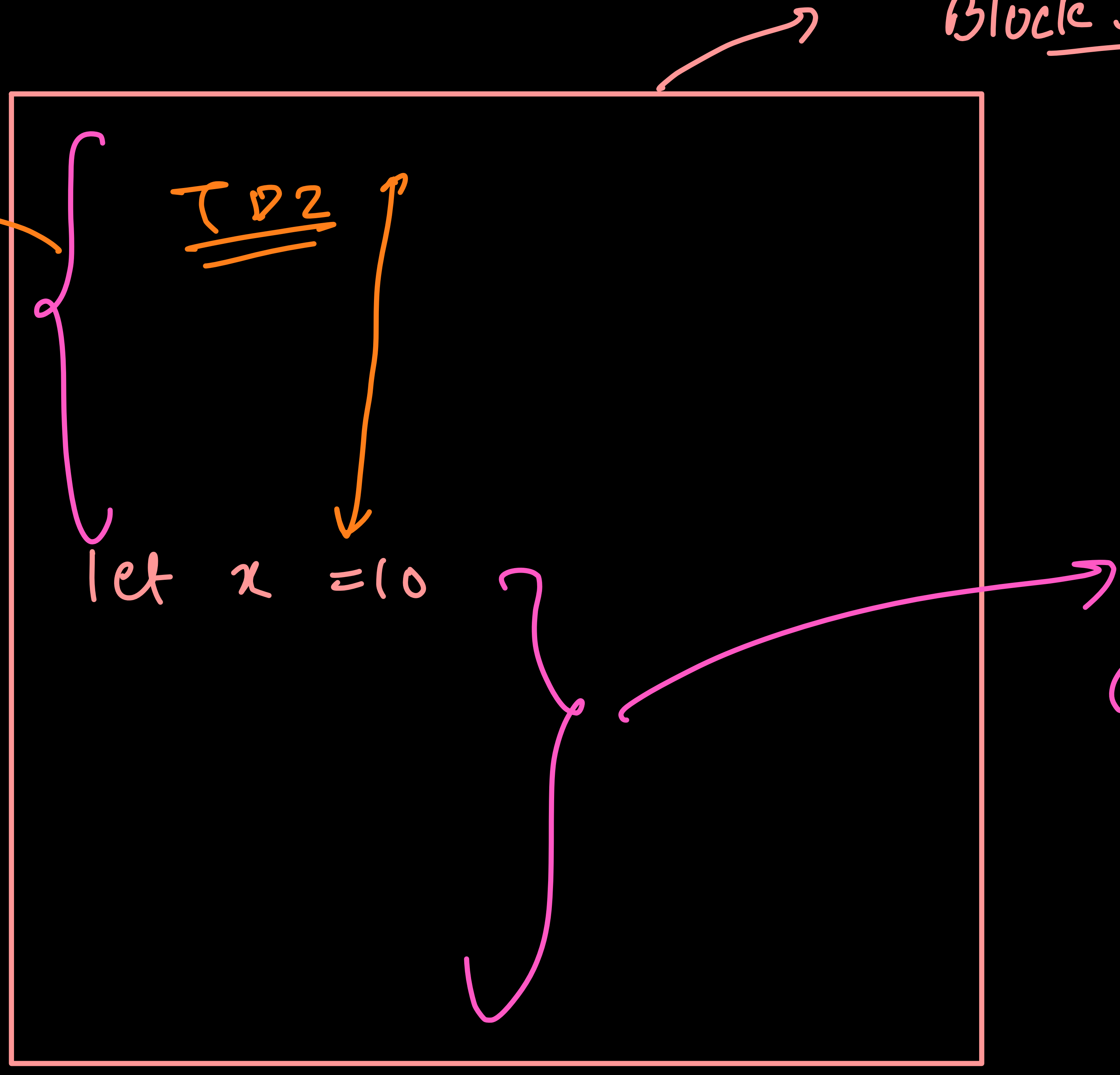
although y has a block scope & here this "if" is a block, we still cannot access y, before the line of its declaration as before that line it is a TDZ for y.

Available

Accessible

Block Scope of x

x is not accessible
It is known
here because the
block is the
scope of x,
but we can't
use it



only this region you
can access x,

var or let ??

readable

```
function f() {
  let x;
  try {
      x = 10
  }
  catch {
      x = 9;
  }
  return x;
}
```

Hoisting

(Phenomenon using which func & variable move at the top)

$\frac{100}{100}$ WRONG

Actually hoisting is a consequence due to which func^ & variable are available before their declaration.

It happens because of lexical Scope.