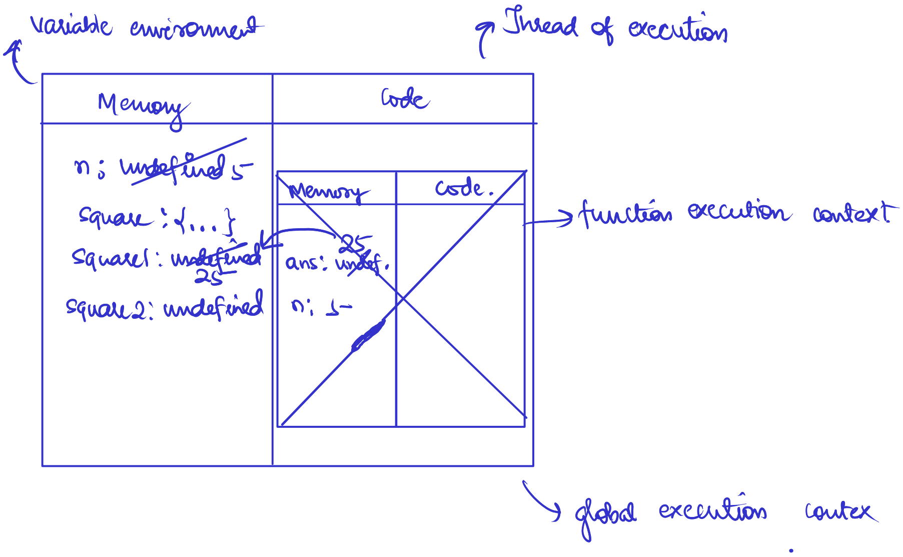
Execution Context:-

```
var n = 5;

function square(n) {
  var ans = n * n;
  return ans;
}

var square1 = square(n);
var square2 = square(8);

console.log(square1)
console.log(square2)
```



```
let n = 5;

function square(n) {
    let ans = n * n;
    return ans;
}

let square1 = square(n);
let square2 = square(8);

console.log(square1);
console.log(square2);
```

→ let & const are not hoisted. They lie in the temporal dead zone.

Hoisting in JS :-

→ Hoistable declarations include function, function*, async function async, function* & var

→ let, const & classes are also hoisted but they don't have default initialization.

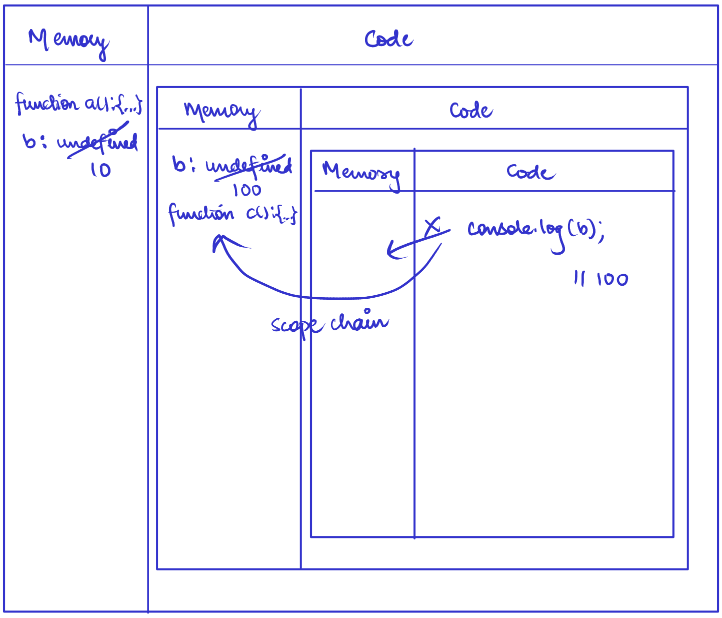
'undefined' v/s 'not defined'

↓ Declared but not initialized (Memory is allocated but no value)

↓ Not declared. (Memory is not allocated)

Scope chain, scope & Lexical Environment

```
function a() {
    var b = 100;
    c();
    function c() {
        console.log(b); // 100
    }
}
var b = 10;
a();
```



Types of Error: Syntax, Reference, and Type.

Uncaught ReferenceError: x is not defined at …
This Error signifies that x has never been in the scope of the program. This literally means that x was never defined/declared and is being tried to be accesed.

Uncaught ReferenceError: cannot access 'a' before initialization
This Error signifies that 'a' cannot be accessed because it is declared as 'let' and since it is not assigned a value, it is its Temporal Dead Zone. Thus, this error occurs.

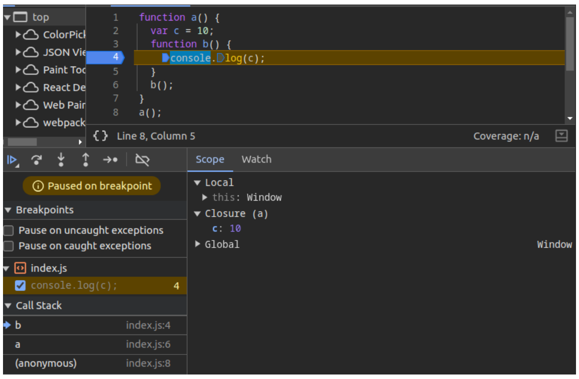
Uncaught SyntaxError: Identifier 'a' has already been declared
This Error signifies that we are redeclaring a variable that is 'let' declared. No execution will take place.

Uncaught SyntaxError: Missing initializer in const declaration
This Error signifies that we haven't initialized or assigned value to a const declaration.

Uncaught TypeError: Assignment to constant variable
This Error signifies that we are reassigning to a const variable.

let & const are blocked scope. & var is function scoped.

Closures :-



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

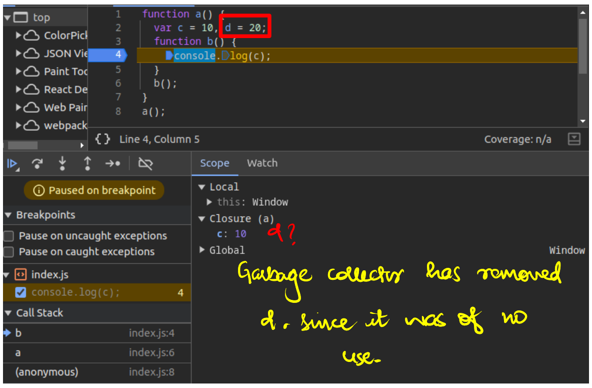
Line 8, Column 5 Coverage: n/a

Paused on breakpoint

Breakpoints
□ Pause on uncaught exceptions
□ Pause on caught exceptions
index.js
☑ console.log(c); 4

Call Stack
b index.js:4
a index.js:6
(anonymous) index.js:8

Scope Watch
Local
this: Window
Closure (a)
c: 10
Global Window



```
function a() {
  var c = 10, d = 20;
  function b() {
    console.log(c);
  }
  b();
}
a();
```

Line 4, Column 5 Coverage: n/a

Paused on breakpoint

Breakpoints
□ Pause on uncaught exceptions
□ Pause on caught exceptions
index.js
☑ console.log(c); 4

Call Stack
b index.js:4
a index.js:6
(anonymous) index.js:8

Scope Watch
Local
this: Window
Closure (a)
c: 10 d ?
Global Window

Garbage collector has removed d. since it was of no use.

→ Advantages :-

- Module Design patterns
- Currying
- Data Hiding
- SetTimeouts

→ Disadvantages :

- Over consumption of memory.
- Memory leak
- Freeze browser.

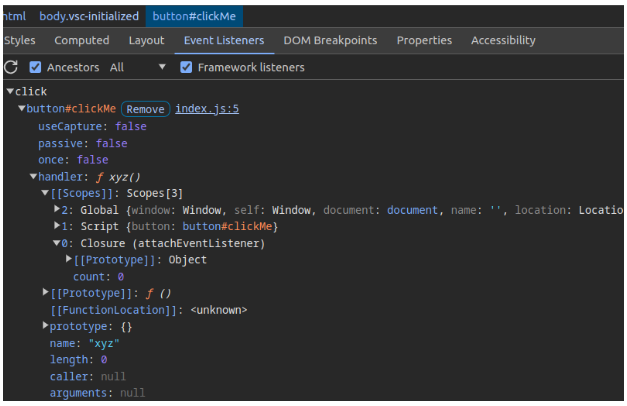
Currying

```
function add(a) {
  return function test(b) {
    return function rest(c) {
      return a + b + c;
    };
  };
}

console.log(add(6)(7)(5));
```

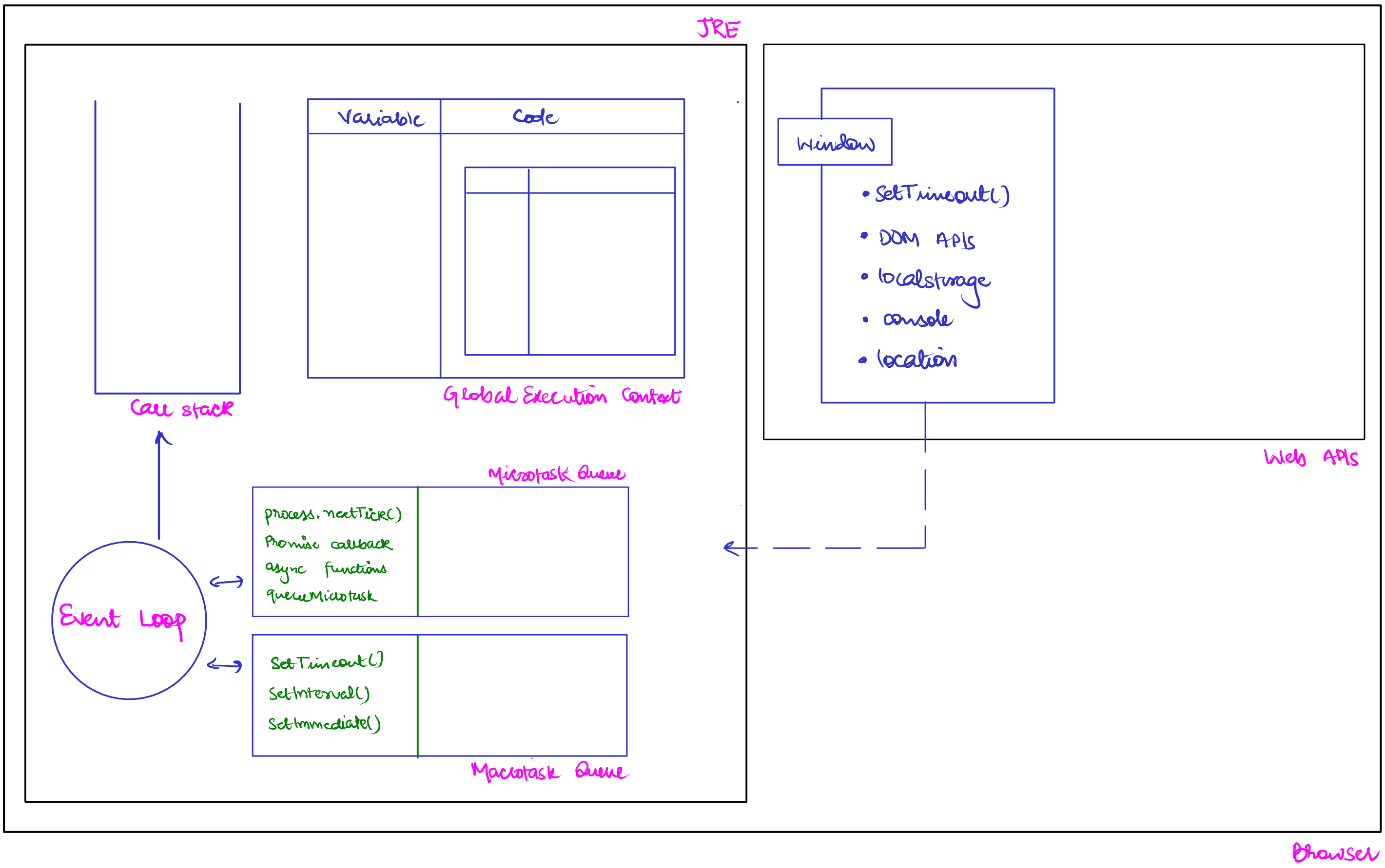
```
function x() {
  function close(i) {
    setTimeout(function () {
      console.log(i);
    }, i * 1000);
  }
  for (var i = 1; i ⩽ 5; i++) {
    close(i);
  }
  console.log("Namaste Javascript");
}

x();
```

```
function a() {
  console.log("Function Statement / Function Declaration");
}
a();

var b = function () {
  console.log("Function Expression");
}
b();

// function () {
  // anonymous function
  // used when functions are used as values
// }

var b = function xyz() {
  console.log("Named Function");
}
b();

var b = function(param1, param2) {
  console.log(param1, param2);
}
var arg1 = 10, arg2 = 10;
b(arg1, arg2);

var b = function(param1) {
  console.log(param1, "First Class Function / Citizens");
}

b(function(){})
```

```
const button = document.getElementById("clickMe");

function attachEventListener() {
  let count = 0;
  button.addEventListener("click", function xyz() {
    console.log("Button clicked", ++count);
  });
}

attachEventListener();
```



```
▼click
 ▼button#clickMe Remove index.js:5
    useCapture: false
    passive: false
    once: false
   ▼handler: ƒ xyz()
     ▼[[Scopes]]: Scopes[3]
       ▶2: Global {window: Window, self: Window, document: document, name: '', location: Locatio
       ▶1: Script {button: button#clickMe}
       ▼0: Closure (attachEventListener)
         ▶[[Prototype]]: Object
          count: 0
     ▶[[Prototype]]: ƒ ()
      [[FunctionLocation]]: <unknown>
     ▶prototype: {}
      name: "xyz"
      length: 0
      caller: null
      arguments: null
```



JRE
Variable
Code
Call stack
Global Execution Context
Window
• SetTimeout()
• DOM APIs
• localstorage
• console
• location
Web APIs
Microtask Queue
process.nextTick()
Promise callback
async functions
queueMicrotask
Event Loop
SetTimeout()
SetInterval()
SetImmediate()
Macrotask Queue
Browser