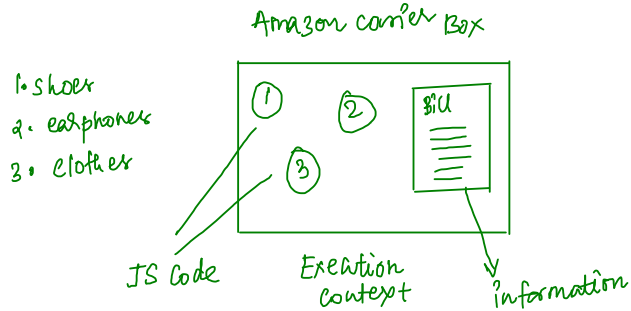
* Execution Context :

→ It is like a black box, where a piece of JS code is executed, It contains all the information for the code to be executed.



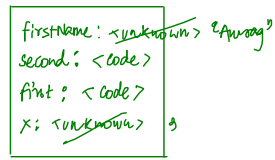
Information

1. variables
   - → let, const, var
   - → function declarations
   - → parameters of functions
2. scope chain

2 phases → Creation phase ('GEC `creation')
→ Execution phase ('stack')

```
621  const firstName = "Anurag";
622
623  function second() {
624    let c = 2;
625    return c;
626  }
627
628  function first() {
629    let a = 1;
630    const b = second();
631    a = a + b;
632    return a;
633  }
634
635  const x = first();
636  console.log(x);
```

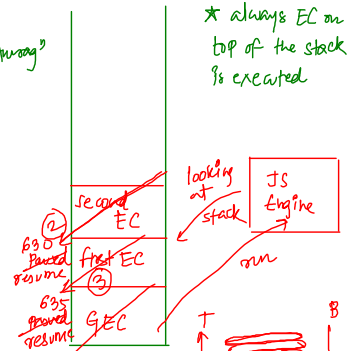
① Global Execution Context



② we will run the EC

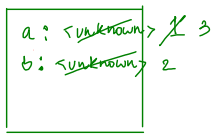
→ EC is placed on to top of the stack.

★ always EC on top of the stack is executed



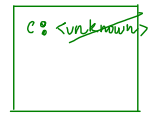
★ when you called first();

① EC is created



★ when you called second();

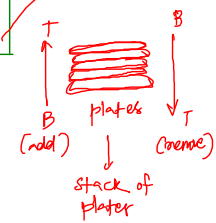
① EC is created



★ EC is completed on return, hence remove from stack

★ At any point of time only one EC will be running. remaining all are paused.

★ there will be only one GEC per JS program. remainig all EC's are created upon function calls.



```
621  const firstName = "Anurag";
622
623  function second() {
624    let c = 2;
625    return c;
626  }
627
628  function first() {
629    let a = 1;
630    const b = second();
631    a = a + b;
632    return a;
633  }
634
635  const x = first();
636  console.log(x);
```

1. GEC Started
2. GEC paused at line 635
3. first EC started
4. first EC paused at line 630
5. Second EC started
6. Second EC completed, return to previous EC
7. first EC resumed at line 630
8. first EC completed, return to previous EC
9. GEC resumed at line 695
10. GEC completed

★ Scope :

Global scope

```
const a = 5;
let firstName = "Arjun";
const year = 2023;

function test() {
  console.log("test", a, firstName, year);
}

test();
console.log("outside test", a, firstName, year);
```

→ declared outside of a function or a block.

→ They are accessible everywhere in the code.

function Sope

```
function calcAge(birthYear) {
  const currYear = 2024;
  const age = currYear - birthYear;
  return age;
}

console.log(calcAge(2001));
console.log(currYear, age);
```

→ variables are accessible only inside the function

→ also called as local scope.

→ Reference Error: not defined

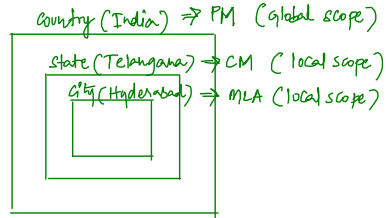
block scope

```
const birthYear = 1998;
if (1981 <= birthYear && birthYear < 2000) {
  const oldGen = true;
}
console.log(oldGen);
```

→ variables are accessible only inside if/else/else if/for/while.

→ function scope and block scope are almost same.

→ this only applies to let and const variables, var declarations are not block scoped (TWIST)



* Hoisting ?

→ It makes variables and functions accessible/usable in the code before they are even declared. "variables and functions are lifted to the top of their scope".

(It is just a feel but actually EC is doing all the JOB)

⇓ Behind the scenes

Before execution, code is scanned for variable/function declarations and it decides whether to lift or not.

functions → Yes

var declarations → Yes, undefined

let, const declarations → No, < Temporal dead zone >

< TDZ >

⇓

the zone where the variable is actually created but **un initialized** and you cannot access it. * *

Example :

```
668  test();
669  demo();
670  console.log(currYear);
671  console.log(example);
672
673  function test() {
674    console.log("Hi, How are you ?");
675  }
676
677  function demo() {
678    console.log("I am demo function");
679  }
680
681  var currYear = 2024;
682  let example = "some random value";
```

① GEC will be created (declarations)

test : <code>
demo : <code>
currYear : undefined
example : <TDZ> (uninitialized)

② place GEC on top of stack

stack

668. test();

669. demo();

670. undefined

671. example is not yet initialized : Reference Error

★ hoisting is actually an out come of EC creation.

★ memory allocated for vars during GEC creation but for let, const it is not allocated.

Op: Hi, How are you ?
I am demo function
undefined
Error ✗

Q1 :

```
685  y = 3;
686  console.log(y);
687  var y = 100;
688  console.log(y);
```

① GEC creation

y : ~~undefined~~	~~3~~ 100

② running EC,

(685) y = 3

(686) 3

(687) y = 100

(688) 100

* what if

var y = 100;

to let y = 100;

y : <TDZ>

(685) y = 3

Reference Error : Cannot access y before initialization.

* var is hoisted but not let/const.

Q2 :

```
691  function example() {
692      console.log(a);
693  }
694  console.log(a);
695  var a = 1;
696  example();
```

① GEC creation

example : <code>
a : ~~undefined~~ 1

② running GEC,

(694) undefined

(695) a = 1

(696)

① example EC

② run example EC

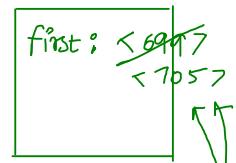
(692) * a is not present in Ex EC
hence look in GEC.

⇒ ①

Q1 :

```
699  function first() {
700    console.log(1);
701  }
702
703  first();
704
705  function first() {
706    console.log(2);
707  }
708
709  first();
```

① GEC creation.



② running GEC,

(709) first(); ⇒ 2

(709) first(); ⇒ 2

★ If you have same function names they will be overridden for the latest (top → bottom)

Q4 :

```
713  var test = 100;
714  console.log(test);
715  function test() {
716    console.log("Insisde function test");
717  }
718  console.log(test);
```

① GEC creation,

test : ~~undefined~~ ~~f 715~~ 100

② running GEC,

(713) test = 100

(714) 100

(718) 100

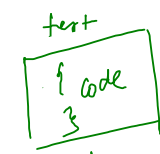
★ during creation,

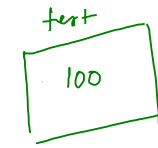
(713) var test

(715)

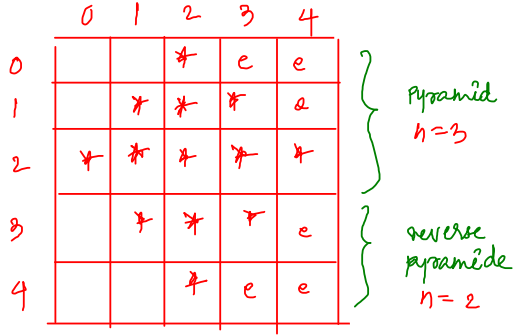
★ during execution,

(713) test = 100

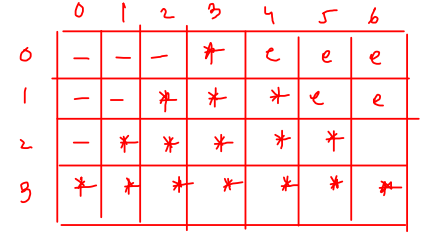




n=5

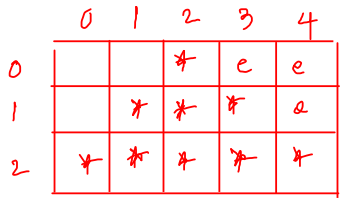


n=4



Q1: print pyramid

n=3



rows = 4 = n

cols

	# spaces	# stars
row = 0	3	1 (2*0+1)
row = 1	2	3 (2*1+1)
row = 2	1	5 (2*2+1)
row = 3	0	7 (2*3+1)

spaces = n - row - 1

stars = (2 * row) + 1

Q2: print reverse pyramid

n = 4

	0	1	2	3	4	5	6	7
0	_	*	*	*	*	*	*	*
1	_	_	*	*	*	*	*	e
2	_	_	_	*	*	*	e	e
3	_	_	_	_	*	e	e	e

rows = 4 = n

Cols

spaces = row

stars = 2 * (n - row) - 1

row	# spaces
row = 0	1
row = 1	2
row = 2	3
row = 3	4

# stars	
7	2 (4 - 0) - 1 ⇒ 8 - 1 = 7
5	2 (4 - 1) - 1 ⇒ 6 - 1 = 5
3	2 (4 - 2) - 1 ⇒ 4 - 1 = 3
1	2 (4 - 3) - 1 ⇒ 2 - 1 = 1