* print continuous character pattern :

ip : 4

op :

```
65
A
66 67
B C
67 68 69
C D E
68 69 70 71
D E F G
```

→ shape : ◺

```
for (let r=0; r<n; r++) {
    let ascii = 65+r;
    for(let c=0; c<r+1; c++) {
        psw(string.fromCharCode(ascii));
        ascii++;
    }
}
```

	Start	
row=0	A (65)	65+0
row=1	B (66)	65+1
row=2	C (67)	65+2
row=3	D (68)	65+3

* startChASCII = 65 + rowNum

* handle row change ⇒ 65 + (r % 26);
* handle col change ⇒ if (ascii == 91) {
    ascii = 65
  }

```
for (let r=0; r<n; r++) {
    let ascii = 65+r;          ⇒ 90
    for(let c=0; c<r+1; c++) {
        psw (string.fromCharCode(ascii));
        ascii++;
    }
}
if (ascii == 91) {
    ascii = 65
}
```

r = 26 ⟶ because 26 alphabets

65 + (row % 26);

65 + (26 % 26)

⇒ 65 + 0 = 65

r = 27

65 + (27 % 26)

65 + (1) = 66

This fails if : (N > 26)

⇒ n = 100

* Cyclic nature involved in problem always % operator helps us.

r=0 A
r=1 B
2 C
3 D
4 E
...
r=25 Z

91

1 2 3 4

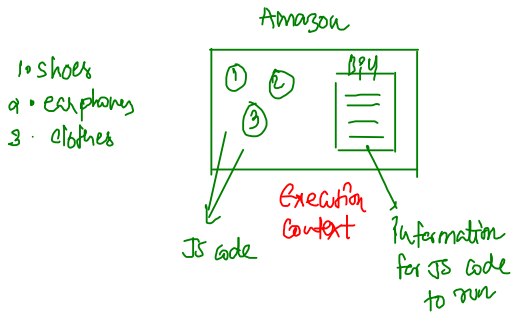
91 92 93 94
90 65 66 67 68
A B C D

r % 26

= 26 → 0 → A 65
= 27 → 1 → B 66
= 28 → 2 → C
= 29 → 3 → D

* 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 run.



Information

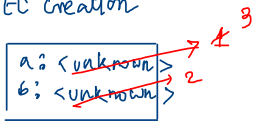
1. variables
   - → let, const, var
   - → function declarations
   - → parameters of functions
2. scope chain

2 phases → Creation phase
         → Execution phase

```
const firstName = "Anurag";

function second() {
  let c = 2;
  return c;
}

function first() {
  let a = 1;
  const b = second();
  a = a + b;
  return a;
}

const x = first();
console.log(x);
```

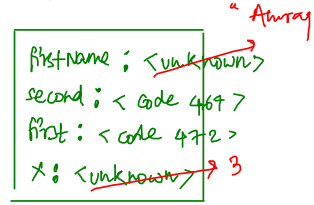
* when first() is called;

① EC creation



② place EC on stack

① Global Execution Context is created.



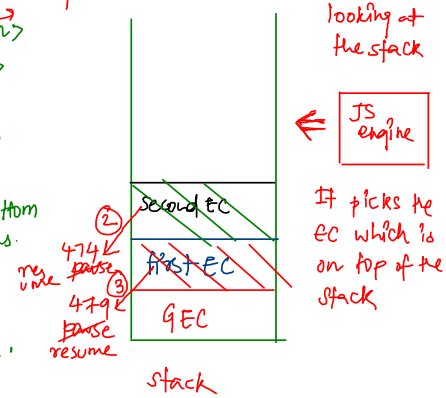
→ scan the code top to bottom for variable and functions.

(Creation phase)

② we will run this EC.

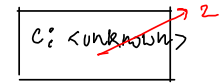
→ this will be placed on to the stack

(running phase)



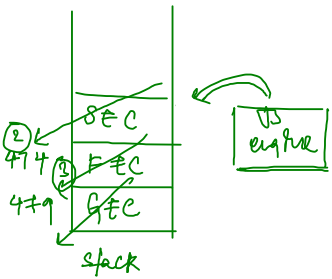
* when second() is called,

① EC creation



② EC place on stack

```
const firstName = "Anurag";

function second() {
  let c = 2;
  return c;
}

function first() {
  let a = 1;
  const b = second();
  a = a + b;
  return a;
}

const x = first();
console.log(x);
```



1. GEC created ad placed on stack
2. GEC is paused at 479
3. first is created and placed on stack
4. FEC is paused at 474
5. SecondEC is creak and placed on stack
6. secondEC is completed, removed from Stack, return (2)
7. Now as FEC is on top of stack it resumes where it got paused ⇒ 474 will recieve (2)
8. FEC is completed, removed from stack, return (3)
9. Now as GEC is on top of stack it resumes where it got paused ⇒ 479 will recieve (1)
10. GEC is complete, removed from stack

* Scope :

Global scope

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

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

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

→ declared out of all the functions, if, else, loops.

→ declared in GEC.

→ they are accessible anywhere in the code.

Function scope

```
function second() {
  console.log(a);
  let c = 2;
  return c;
}

function first() {
  let a = 2;
  const b = second();
  a = a + b;
  return a;
}

const x = first();
console.log(x);
```

→ variables created in the function are accessible within that function only.

→ the above code gives Reference error: a is not defined at line 496

block Scope

```
const birthYear = 1988;
if (1981 <= birthYear && birthYear <= 1990) {
  const oldGen = true;
}
console.log(oldGen);

const birthYear = 1988;
let oldGen;
if (1981 <= birthYear && birthYear <= 1990) {
  oldGen = true;
}
console.log(oldGen);
```

→ variables declared inside if /else /else if /for /while i.e; any block except functions are accessible within those blocks only.

→ function/block scope are almost same but not exactly same.

* undefined vs not defined

- value: Variable is present but no value assigned (empty box)
- error: Variable is not present (box is only not there)

```
// var with block scope
const birthYear = 1988;
if (1981 <= birthYear && birthYear <= 1990) {
  var oldGen = true;
}
console.log(oldGen);

// var with function scope
function test() {
  var firstName = "anurag";
}

test();
console.log(firstName);
```

not defined

* block scope is only applicable for let, const but function scope applies for let, const, var or not. (this behaviour is due to hoisting).

★ Hoisting :

→ It makes variables and functions accessible in the code even before they are declared. ("variables, fractions are lifted to top of their scope/code") ↘ wrong way of understanding

↓ behind the scenes

It is due to the execution context, hoisting is an outcome of creation phase.

	Are they hoisted	Initial value
Functions	yes	Code
var declarations	yes	undefined
let, const declarations	NO	< Temporal Dead zone > <TDZ>

*Example :

```
543  test();
544  demo();
545  console.log(currYear);
546  console.log(example);
547
548  function test() {
549    console.log("Hi, How are you ?");
550  }
551
552  function demo() {
553    console.log("I am a demo function");
554  }
555
556  var currYear = 2023;
557  let example = "some random example";
```

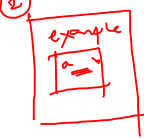
① GEC created,

test : <code 548>
demo : <code 552>
currYear : <undefined> (uninitialized)
example : <TDZ> (=> <value unavailable>

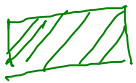
undefined
error

"some random example"

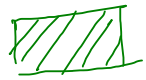
② place GEC on stack



* test()



* demo()



* memory is allocated ('initialized')
for var during creation phase only. but for let, const it is allocated during execution.

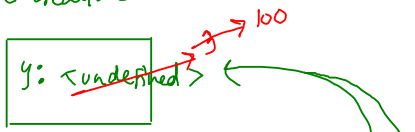
* 546 will give an error :
Cannot access before initialization

Q1:

```
561   y = 3;
562   console.log(y);
563   var y = 100;
564   console.log(y);
```

→ let y = 100;

① GEC creation



② run the EC, place on stack

561. y = 3
562. cl(y) ⇒ 3
563. y = 100
564. cl(y) ⇒ 100

★ what if

569. let y = 100;

① GEC

② run the EC,

561. y = 3 → Error

y is not accessible

Q2 :

```
567  function example() {
568    console.log(a);
569  }
570
571  console.log(a);
572  var a = 1;
573  example();
```

① GEC creation ;

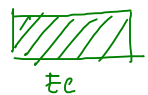
example : <code 567>
a : <undefined>
1

② run ,

571. cl(a); ⇒ undefined

572. a = 1

573. example()



run

568. cl(a); ⇒ 1.

→ a is in my EC ⇒ no?

→ is it there in gEC ⇒ yes

Qs :

```
576  function first() {
577    console.log(1);
578  }
579
580  first();
581
582  function first() {
583    console.log(2);
584  }
585
586  first();
```

① GfC Created,

first : <Code 576> <Code 582>

② run() ;

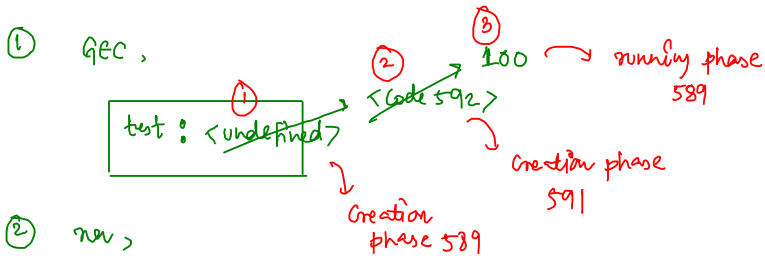
580 : 2

586 : 2

* If you have same function names they will be replaced with last / latest function (top → Bottom)

Q4 :

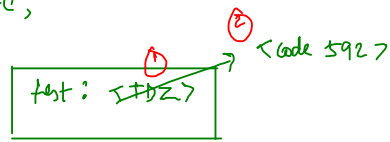
```
589  var test = 100;
590  console.log(test);
591  function test() {
592    console.log("Inside the function");
593  }
594  console.log(test);
```



Q5 :

```
589  let test = 100;
590  console.log(test);
591  function test() {
592    console.log("Inside the function");
593  }
594  console.log(test);
```

(1) GEC,



(2) 589. test = 100;

whenever you try to initialise test, error: It has been already declared.

this will fail because test is already present in EC