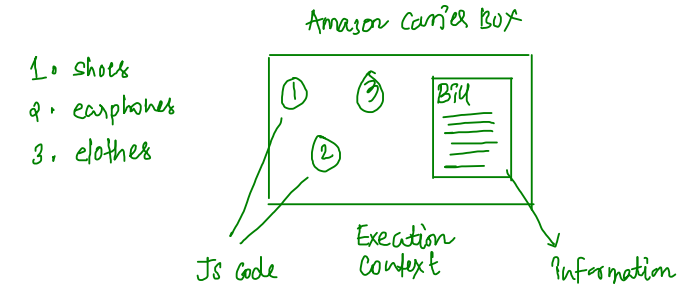
✱ Execution Context :

→ 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 func's
2. Scope chain

2 phases → Creation phase
2 phases → Execution phase

```
563  const firstName = "Anurag";
564
565  function second() {
566    let c = 2;
567    return 2;
568  }
569
570  function first() {
571    let a = 1;
572    const b = second();
573    a = a + b;
574    return a;
575  }
576
577  const x = first();
578  console.log(x);
```

① Global Execution Context,

★ look for variable declaration and functions

firstName : ~~<unknown>~~ "Anurag"
Second : <code>
first : <code>
x : ~~<unknown>~~ 3

② we will run the GEC,

→ GEC is placed on top of the stack.

★ when first is called,

① EC is created

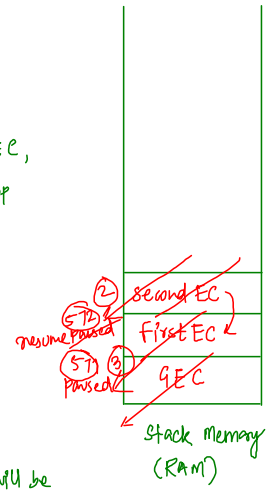
a : ~~unknown~~ 1 3
b : ~~unknown~~ 2

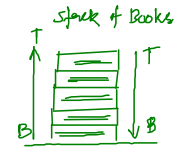
★ return statement, EC is completed hence remove from stack

★ when second is called,

① EC is created

c : ~~unknown~~ 2

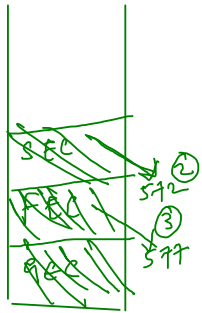




★ At any point of time, only one EC will be running, remaing EC's are paused.

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

```
563  const firstName = "Anurag";
564
565  function second() {
566    let c = 2;
567    return 2;
568  }
569
570  function first() {
571    let a = 1;
572    const b = second();
573    a = a + b;
574    return a;
575  }
576
577  const x = first();
578  console.log(x);
```



1. GEC is created
2. GEC is paused at 577
3. first EC is started
4. first EC paused at 572
5. second EC is started
6. second EC is completed, return 2 to previous EC
7. first EC resumed at 572, recieved (2)
8. first EC is completed, return 3 to previous EC
9. GEC is resumed at 577, recieved (3)
10. GEC is 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 block

→ declared in GEC

→ They are accessible everywhere in the code

Function Scope

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

function first() {
  const a = 2;
  const b = second();
  console.log(c);
  a = a + b;
  return a;
}
```

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

→ Reference error not defined

block Local Scope

```
const birthYear = 1998;
if (1981 <= birthYear && birthYear <= 1990) {
  const oldGen = true;
}
console.log(oldGen);
```

→ Variables are accessible only inside if/else/else if/for/while

→ function, local/block scope are almost same.

→ applies only for let and const variables, var declarations are not block scoped.

```
// Local/Block Scope
// Var does not follow local/block scope
const birthYear = 1998;
if (1981 <= birthYear && birthYear <= 2000) {
  var oldGen = true;
}
for (let i = 0; i < 10; i++) {
  var lastName = "Kapoor";
}

function demo() {
  console.log(oldGen);
  console.log(lastName);
  var firstName = "Anurag";
}
demo();
console.log(firstName);
```

true

Kapoor

Not defined → Reference Error

* Hoisting :

→ It makes variables & functions accessible in the code before they are even declared.

"variables, functions are lifted to top of their scope"

⇓ Behind the scenes

Before execution, the creation phase is responsible for this behaviour.

Functions → Yes

var declarations → Yes, undefined

let, const declarations → NO, < temporal dead zone >

< TDZ >

Initial value

Zone where the variable is created but "uninitialised"

⇒ not yet ready to use.

Example :

```
630  test();
631  demo();
632  console.log(currYear);
633  console.log(example);

634
635  function test() {
636    console.log("Hi, How are you ?");
637  }
638
639  function demo() {
640    console.log("I am a demo function");
641  }
642
643  var currYear = 2024;
644  let example = "some random text";
```

632 → undefined

633 → Error

① GEC Created

test : <code> 635

demo : <code> 639

currYear : undefined

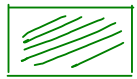
example : <TDZ> → uninitialised

② place GEC on to stack

* memory is allocated for var during GEC creation but for let, const it is not allocated

* test EC

630 .



632 . undefined

* demo EC

631 .

633 . Error ; example is not yet initialised

Q1:

```
647  y = 3;
648  console.log(y);
649  var y = 100;
650  console.log(y);
```

→ let y = 100 (or) const y = 100

① GEC creation,

y: ~~undefined~~ 3 100

② run the EC,

(647) y = 3

→ Is y present in GEC

(648) (3)

(649) y = 100

(650) (100)

★ what if ① Create

let y = 100 y: <TDZ>

② run,

(647) y = 3

→ Reference Error: Cannot access y before initialization.

★ var is hoisted, but let/const are not hoisted

Q2 :

```
653  function example() {
654    console.log(a);
655  }
656
657  console.log(a);
658  var a = 1;
659  example();
```

1

undefined

① GEC creation,

example : <653>
a : ~~undefined~~ 1

② run,

(651) undefined
(658) a = 1
(659)

① 
example EC

② run ExampleEC
(654) console.log(a)
→ a is not present in EEC, check in GEC → ①

Q3 :

```
662  function first() {
663    console.log(1);
664  }
665
666  first();
667
668  function first() {
669    console.log(2);
670  }
671
672  first();
```

① GEC Creation,

② run,

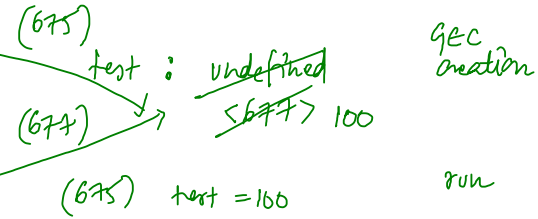
(666) first() ⇒ ②

(672) first() ⇒ ②

★ If you have same function names, they will be overridden/replaced with the last/latest func (top → bottom)

Q4:

```
675  var test = 100;
676  console.log(test);
677  function test() {
678    console.log("Inside function test");
679  }
680  console.log(test);
```



① GEC creation,

test : undefined <677> 100

② run,

(675) test = 100

(676) 100

(680) 100

* Binary to decimal :

Eg: 1010

Op: 10

eg: 1111

op: 15

$n = 1111 \,\%\, 10 = (1) \times 2^0$

$111 \,\%\, 10 = (1) \times 2^1$

$11 \,\%\, 10 = (1) \times 2^2$

$1 \,\%\, 10 = (1) \times 2^3$

(0)

$10^3\ 10^2\ 10^1\ 10^0$ → Base

1000100 10 1

1 2 3 4

$\Rightarrow 1 \times 1000 + 2 \times 100 + 3 \times 10 + 4 \times 1$

$2^3\ 2^2\ 2^1\ 2^0$ → Base

1010

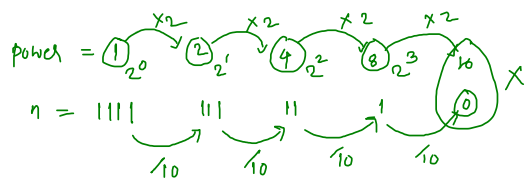
$\Rightarrow 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$\Rightarrow 8 + 0 + 2 + 0 = 10$

1111

$\Rightarrow 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$\Rightarrow 8 + 4 + 2 + 1 = 15$



→ The previous approach fails for large strings,

ip: "10100010101101011010001111111100001"

⇓

converting to a Number (at max it can only convert to Number for 16/17 len string)

$2^{10} = 1024$

$\simeq 10^3$

$2^{10} \simeq 10^3$

$(2^{10})^5 \simeq (10^3)^5$

$2^{50} \simeq 10^{15}$

⇒ Number har max limit value $= 2^{53} - 1$

$\simeq 10^{15}$ or 10^{16}

how digits in 10^{15} → 16 digits

⇓

16 len string

* you can use inbuilt JS,

parseInt("10101011", 2)

⇓ → Base

decimal equivalent