* Functions : (Abstraction – hiding implementation details) → we know what a function does? but how it does we dont know.

→ solves the problem of DRY, provides reusability.

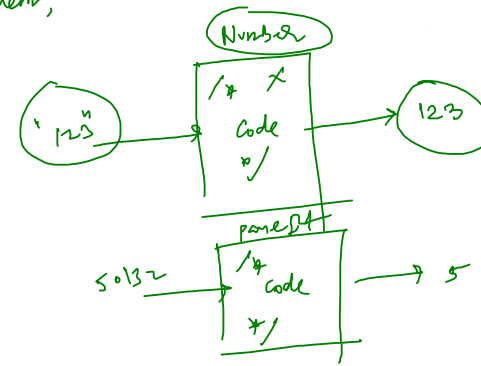
→ It can take Inputs by parameters / arguments.

→ It can send some response by return keyword.

→ return shutdowns / terminates the function, any code after this will not be executed.

→ we used many functions without knowing them,

```
const a = Number("123")
const b = parseInt(5.132)
console.log("...")
```



```
485  function cutPieces(fruit) {
486    return 4 * fruit;
487  }
488
489  function fruitProcessor(apples, oranges) {
490    const applePieces = cutPieces(apples);
491    const orangePieces = cutPieces(oranges);
492    const ans = `Juice with ${applePieces} pieces of apples and ${orangePieces} pieces of oranges`;
493    return ans;
494  }
495
496  const res = fruitProcessor(4, 2);
497  console.log(res);
```

1. 496 called

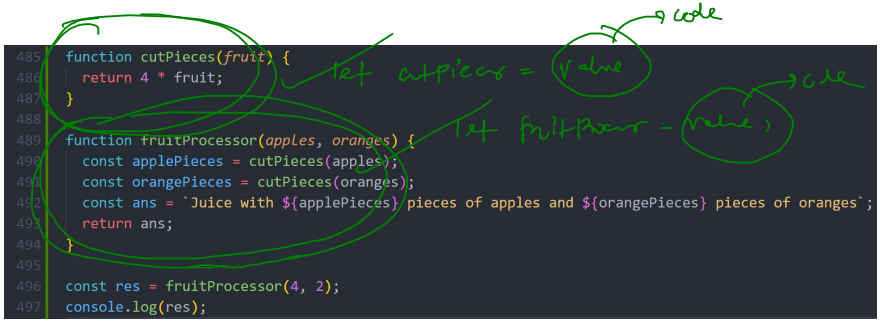
9. 489 (apples = 4, oranges = 2)
3. 490
4. 485 (fruit = 4)
5. 486 (4 * 4 = 16)
490. applePieces = 16
491. orangePieces = 8
493. return a

496. res = a (received)

9:20 pm
– 9:35 pm
BREAK

fruitProcessor(3, 1)

496. fruitProcessor(3, 1)
489. apples = 3, oranges = 1
490. cutPieces(3)
485. fruit = 3
486. return 4 * 3 = 12
490. applePieces = 12
491. cutPieces(1)
485. fruit = 1
486. 4 * 1 = 4
491. orangePieces = 4
493. return "..."
496. res = "..."


code
let cutPieces = value
code
let fruitProcessor = value
function cutPieces(fruit) {
return 4 * fruit;
}
function fruitProcessor(apples, oranges) {
const applePieces = cutPieces(apples);
const orangePieces = cutPieces(oranges);
const ans = `Juice with ${applePieces} pieces of apples and ${orangePieces} pieces of oranges`;
return ans;
}
const res = fruitProcessor(4, 2);
console.log(res);

★ Binary, decimal :

base2 (0,1)

base10 (0-9)

10^3 10^2 10^1 10^0

1 2 3 4

⇒ $1*1000 + 2*100 + 3*10 + 4*1$

= $1000 + 200 + 30 + 4$

= 1234

2^3 2^2 2^1 2^0

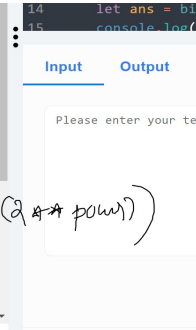
1 1 0 1 Binary

⇒ $1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

⇒ $8 + 4 + 0 + 1$

⇒ 13 (decimal)

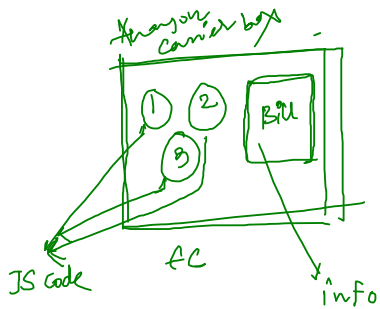
```
Sum = 0
pow = 0
while (num > 0) {
  dig = num % 10
  sum = Sum + (dig * (2 ** pow))
  num = Math.floor(num/10);
  power++;
}
```



★ Execution Context :

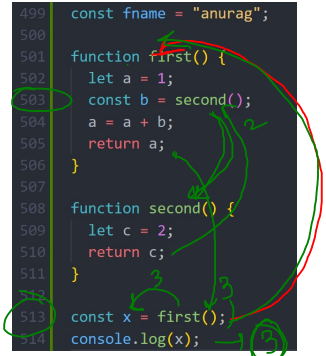
→ It is like a black box, where a piece of JS code is executed, It contains all the information for the code to be executed.

1. shoes
2. earphones
3. clothes



Information

1. Variables
   - → let, const
   - → function
   - → parameters



① Execution Context

GEC

fname : 'anurag'

first : <code>

second : <code>

x : ~~unknown~~ 3

② we will run the EC

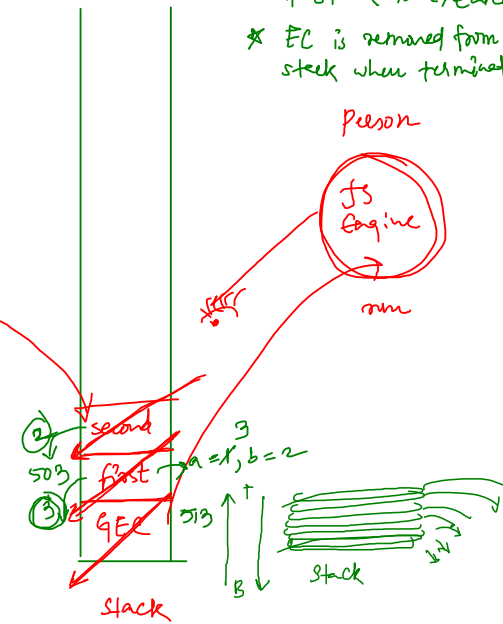
→ EC is placed on to the stack.

* always EC on top of stack is executed
* EC is removed from stack when terminated

Person

JS Engine

run



first

a : ~~1~~ 3

b : ~~unknown~~ 2

second

c : 2

* EC is created for every function.

* At any point of time only one EC will be running.
* only one GEC is created per program.

get started
get paused
first started
first paused
second started
second completed/returned
first resumed
first completed/returned
gec resumed
gec completed/returned