* Functions : (Abstraction → hiding Implementation / code details) → we know what that function does, but we dont know how it does

→ DRY - Do not repeat yourself, write the code once use it wherever needed.

→ with loops you can repeat a task at a specific location in the code.

Syntax : → function name (anything)

```
541  function logger(){
542      console.log("Hello, I am Anurag");
543      console.log("I work at Acciojob");
544      console.log("I love teaching");
545  }
546
547  logger();
```

→ when called all the code inside the function will be executed.

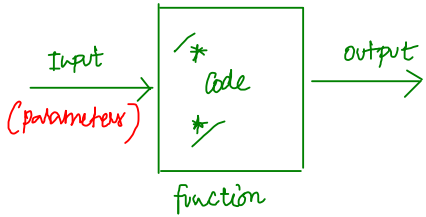
call / invoke / run the function

→ functions can take inputs in form of parameter / arguments.

→ functions can provide response / output using a return statement.

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

①



Eg: addition of Two numbers

→ these are like variables

a = 5, b = 8 (parameters/arguments)

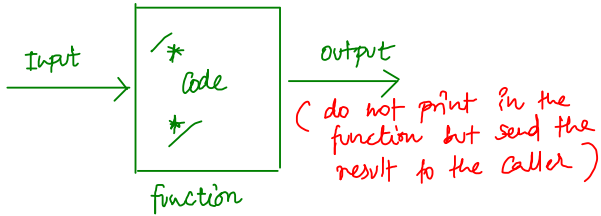
↳ you can any no of parameters

```
568  function add(a, b) {
569      a = Number(a);
570      b = Number(b);
571      console.log(a + b);
572  }
573
574  add(5, 8);
```

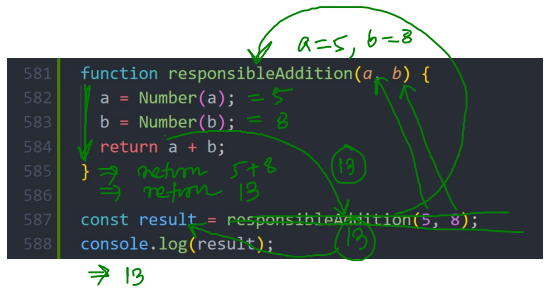
→ console.log (5+8)

→ 13

(2)



Eg: addition of Two numbers



1. you called the function with a=5, b=8
2. function, return 5+8;
   return 13;
3. the value is returned to the place where function is called.
4. const result = funcCall();
   const result = 13;

③

```
590  function incorrectAddition(a, b) {
591    a = Number(a);
592    return;
593    b = Number(b);
594    return a + b;
595  }
596
597  const res = incorrectAddition(5, 8);
598  console.log(res);
```

→ whenever you encounter a return statement stop the function and send response to caller.

return ; → undefined

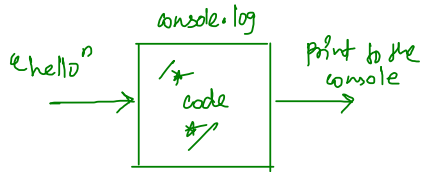
return 45; → 45

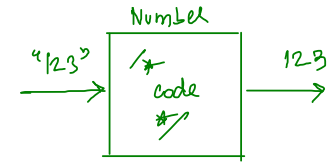
⇒ undefined

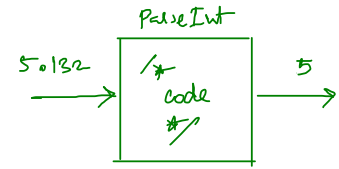
→ we have used many functions without knowing them,

* TV remote,
⊕ volume increaser.
do you know how the sensors work b/w TV and remote ? (physics behind)

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







① A very simple basic fruit processor

→ went to shop
→ asked for juice

```
604  function fruitProcessor() {
605    console.log("Juice with 4 apples and 2 oranges");
606  }
607
608  fruitProcessor();
```

② Now you want mention no of apples, oranges, add the feature of user selecting ↑

→ went to shop
→ asked for juice with your choice of apples, oranges

```
604  function fruitProcessor(apples, oranges) {
605    console.log(`Juice with ${apples} apples and ${oranges} oranges`);
606  }
607
608  fruitProcessor(2, 3);
```

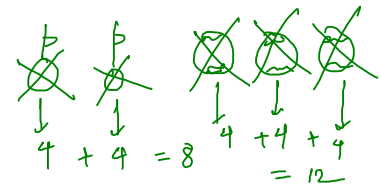
③ cut each fruit into 4 pieces and then make the juice

Eg; apples = 2
oranges = 3

op: "Juice with 8 pieces of apple and 12 pieces of oranges"

```
function fruitProcessor(apples, oranges) {
  const applePieces = apples * 4;
  const orangePieces = oranges * 4;
  console.log(
    `Juice with ${applePieces} pieces of apples and ${orangePieces} pieces of oranges`
  );
}

fruitProcessor(8, 12);
```



④ You can see we are printing inside the function But In general we always expect the functions to respond with results. (return)

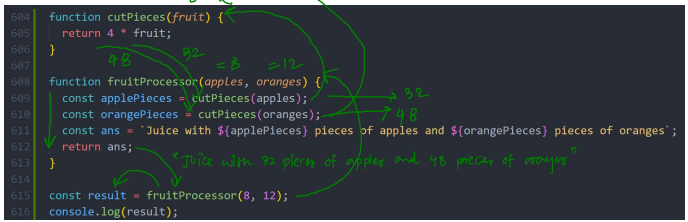
```
function fruitProcessor(apples, oranges) {
  const applePieces = apples * 4;
  const orangePieces = oranges * 4;
  const ans = `Juice with ${applePieces} pieces of apples and ${orangePieces} pieces of oranges`;
  return ans;
}

const result = fruitProcessor(8, 12);
console.log(result);
```

⇒ return sends the value to the caller

→ "Juice with 32 pieces of apples and 48 pieces of oranges"

(5) Make a separate function for cutting fruits,



⇒ "Juice with 32 pieces of apples and 48 piecer of oranges"

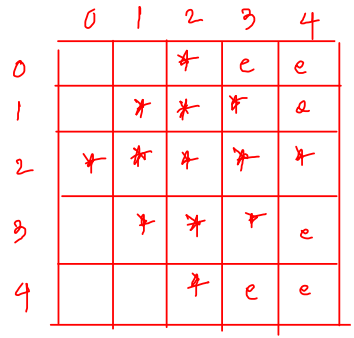
9:35pm
– 9:50pm
BREAK

615. fruitProcessor(8, 12)
609. cutPieces(8)
605. return 4 * 8 ⇒ return 32
609. applePieces = 32
610. cutPieces(12)
605. return 4 * 12 ⇒ return 48
610 orangePieces = 48
611. ans = "Juice with 32 pieces of apples and 48 piecer of oranges"
612. return ans
615. result = "....."

function call & to cutPieces

entire function call

$n = 5$



rows = 5 = n

cols

	# spaces	# stars	# cols
row = 0	2	1	3
row = 1	1	3	4
row = 2	0	5	5
row = 3	1	3	4
row = 4	2	1	3

half = 5/2 = 2.5 = (2)

```
row = row % half;
# spaces = Math.abs(half - row);
# star = 2 * row + 1;
```

continued in tomorrow's class

* Binary to decimal :

eg: num = 1010

(base-2)

1 0 1 0 (binary)

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

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

$\rightarrow 0 + 2 + 0 + 8$

$\Rightarrow 10$ (decimal)

(base-10)

9 3 6 2 (decimal) (0,1,2,3,4,5,6,7,8,9)

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

$\Rightarrow 2*10^0 + 6*10^1 + 3*10^2 + 9*10^3$

$\rightarrow$ 9362

eg: 1 1 0 1 1 1 (binary)

2^5 2^4 2^3 2^2 2^1 2^0

$\Rightarrow 1*2^0 + 1*2^1 + 1*2^2 + 0*2^3 + 1*2^4 + 1*2^5$

$\Rightarrow 1 + 2 + 4 + 0 + 16 + 32$

$\Rightarrow 55$ (decimal)

eg: 1 1 0 1 1 1 (binary)

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

```
let decimal = 0;
let power = 0;
while (n != 0) {
  const digit = n % 10;
  decimal += digit * (2 ** power);
  power ++;
  n = parseInt(n/10);
}
```

dec = 0 1 8 7 7 23 55

Power 0 1 2 3 4 5 6

① 111011 % 10 = 1

decimal += 1 * (2 ** 0)

= 1 * 2^0 = 1

n = 111011 / 10 = 11011

② 11011 % 10 = 1

dec += 1 * (2 ** 1)

= 1 * 2^1 = 2

③ 1101 % 10 = 1

dec += 1 * (2 ** 2)

= 1 * 2^2 = 4

④ 110 % 10 = 0

dec += 0 * (2 ** 3)

= 0 * 2^3 = 0

⑤ 11 % 10 = 1

dec += 1 * (2 ** 4)

= 1 * 2^4

= 16

⑥ 1 % 10 = 1

dec += 1 * (2 ** 5)

= 1 * 2^5

= 32

⑦ 0 X

→ Number in JS has some limit, $(2^{53}-1)$

$2^{10} = 1024$
$\simeq 10^3$

$\Rightarrow (2^{50} \cdot 2^3 - 1)$

$\Rightarrow ((2^{10})^5 \cdot 2^3 - 1)$

$\Rightarrow ((10^3)^5 \cdot 2^3 - 1)$

$\Rightarrow (10^{15} \cdot 2^3 - 1) \simeq 10^{15}$ (max number that you can store)
$\simeq 10^{16}$

$10^{16} \rightarrow$ 17 digits
$10^{15} \rightarrow$ 16 digits

* Inbuilt function, parseInt("1010", 2)

$\Rightarrow 10$