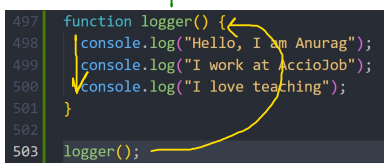
★ Functions : (Abstraction - Hiding Implementation details)

→ DRY - Do not repeat yourself, write the code once and use whenever needed.

Syntax :

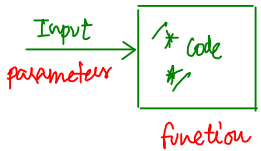


→ When called all the code inside the function will be executed.

→ loop can perform a repeatitive only at a certain location.

1. functions can take inputs in form of parameter/arguments.
2. functions can provide response/output using a return statement.
3. return statement shutdowns/terminates the function, any code after this will not be executed.

①



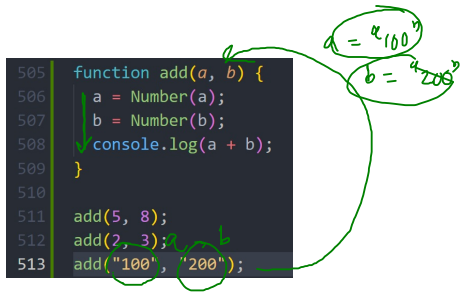
a = "100"

b = "200"

cl (a+b) ⇒ "100" + "200" ⇒ "100200"

add ("A", "B") ⇒ a = NaN , b = NaN

Eg: Addition of two numbers



506. a = Number(5) = 5

507. b = Number(8) = 8

508. cl (5+8) ⇒ 13

---

506. a = Number ("100") = 100

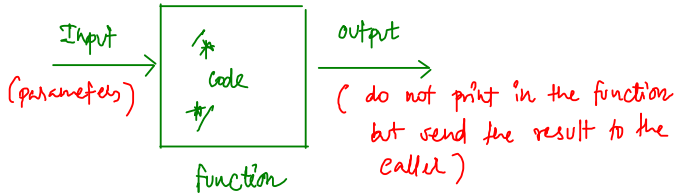
507. b = Number ("200") = 200

508. cl (100+200) ⇒ 300

(a, b) → parameter / arguments , you can have any number of them.

↘ these are like variables

~~2~~



* the value from the function is sent back to the calling location; (a)
  called is replaced with the returned value.

```
519  function add(a, b) {
520    a = Number(a);
521    b = Number(b);
522
523    return a + b;
524  }
525
526  const ans = add(5, 8);
527  console.log(ans);
```

a = 5
b = 8

13

13

520. a = Number(5) = 5
521. b = Number(8) = 8
523. return a+b;
return 13;

const ans = add(5, 8);
ans = 13

(9)

```
533  function add(a, b) {
534    a = Number(a);
535    return;
536
537    b = Number(b);
538    return a + b;
539  }
540
541  const res = add(2, 3);
542  console.log(res);
```

a = 2
b = 3
undefind

→ undefined

→ Whenever you encounter return, stop the function and send response to caller

return ; ⇒ undefined

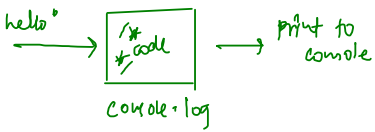
return (45); ⇒ 45

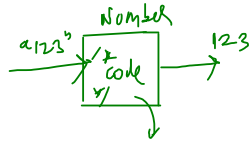
→ we have used many functions without knowing what a function is,

★ TV remote,

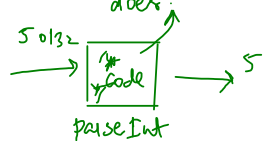
(+) volume increases, do you know how sensors btwn the remote and the TV work? (physics)

```
Const a = Number("123")
Const b = parseInt("5.132")
Console.log("Hello.....")
```





we dont know code but you know what it does.



① A very simple basic fruit processor

```
548  function fruitProcessor() {
549    console.log("Juice with 4 apples and 2 oranges");
550  }
551
552  fruitProcessor();
```

→ went to juice shop
→ asked for juice
(for every customer he makes same juice)

② Now you want to add feature of customers selecting no of apples, oranges

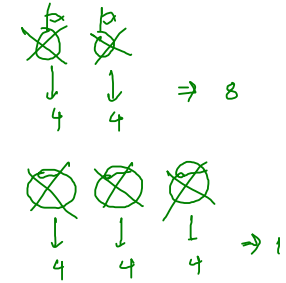
```
548  function fruitProcessor(apples, oranges) {
549    console.log(`Juice with ${apples} apples and ${oranges} oranges`);
550  }
551
552  fruitProcessor(2, 3);
```

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

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

eg: apples = 2, orange = 3

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



```
548  function fruitProcessor(apples, oranges) {
549    const applePieces = apples * 4;
550    const orangePieces = oranges * 4;
551    console.log(
552      `Juice with ${applePieces} pieces of apple and ${orangePieces} pieces of oranges`
553    );
554  }
555
556  fruitProcessor(2, 3);
```

④ Instead of printing inside the function, make the function to respond with the result.
(return statement sends the value to caller)

```
548  function fruitProcessor(apples, oranges) {
549    const applePieces = apples * 4;
550    const orangePieces = oranges * 4;
551    const ans = `Juice with ${applePieces} pieces of apple and ${orangePieces} pieces of oranges`;
552    return ans;
553  }
554
555  const res = fruitProcessor(2, 3);
556  console.log(res);
```

⑤ Make a seperate function for cutting fruits → hiring a worker to cut the fruits

→ you will concentrate on making the juice.

```
548  function cutPieces(numFruits) {
549    return 4 * numFruits;
550  }
551
552  function fruitProcessor(apples, oranges) {
553    const applePieces = cutPieces(apples);
554    const orangePieces = cutPieces(oranges);
555    const ans = `Juice with ${applePieces} pieces of apple and ${orangePieces} pieces of oranges`;
556    return ans;
557  }
558
559  const res = fruitProcessor(2, 3);
560  console.log(res);
```

* Always try to divide the tasks into separate functions, each function should concentrate only on one particular task.

559 · fruitProcessor(2, 3)

553 · cutPieces(2)

549 · return 4 * 2 ⇒ return 8

553 · applePieces = 8

554 · cutPieces(3)

549 · return 4 * 3 ⇒ return 12

554 · orangePieces = 12

559 · ans = "Juice with 8 pieces of apple and 12 pieces of oranges"

560 · return ans

559 · res = "Juice with 8 pieces of apple and 12 pieces of oranges"

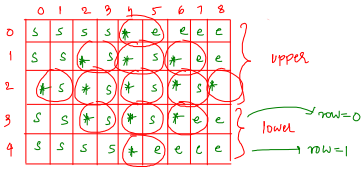
560 · console(res)

function calls to cutPieces

entire fruitProcessor function call.

★ Diamond pattern : "*", "*_" (star pyramid)

Eg: n = 5



n = 5

no. of rows in upper = n/2 + 1

= 5/2 + 1

= 2 + 1 = 3

	col: spaces	col: stars
row = 0	4	1 (2*0+1)
row = 1	2	3 (2*1+1)
row = 2	0	5 (2*2+1)

Rows = n/2 + 1 = 3

stars = 2*row + 1

spaces = (#Rows - row - 1) * 2

(n-r-1)*2

(3-0-1)*2 = 4

(3-1-1)*2 = 2

(3-2-1)*2 = 0

---

no. of rows in lower = n - upperRows

	col: spaces	col: stars
row = 0	2	3
row = 1	4	1

2(n-r)-1

#Rows = n - upperRows = 5 - 3 = 2

stars = 2*(#Rows - row) - 1

2*(2-0) - 1 = 4-1 = 3

2*(2-1) - 1 = 2-1 = 1

spaces = (row+1)*2

(0+1)*2 = (2), (1+1)*2 = (4)

★ general formulas,

n-r, n-r-1, r+1, r, n/2, n/2 + 1, 2r+1, 2(r+1), 2r-1,

2(r-1), 2(n-r), 2(n-r-1), 2(n-r)-1, 2(n-r)+1