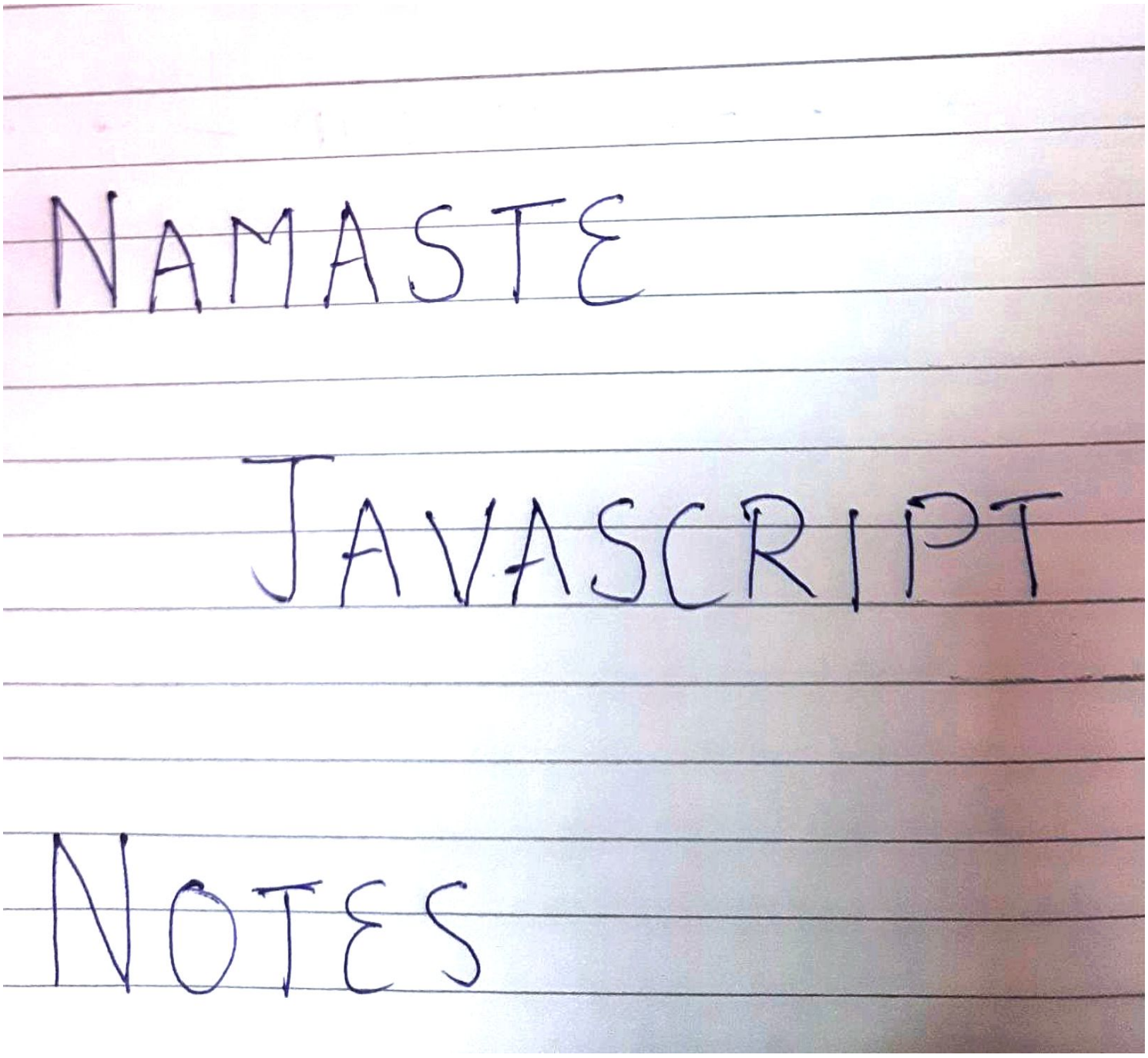
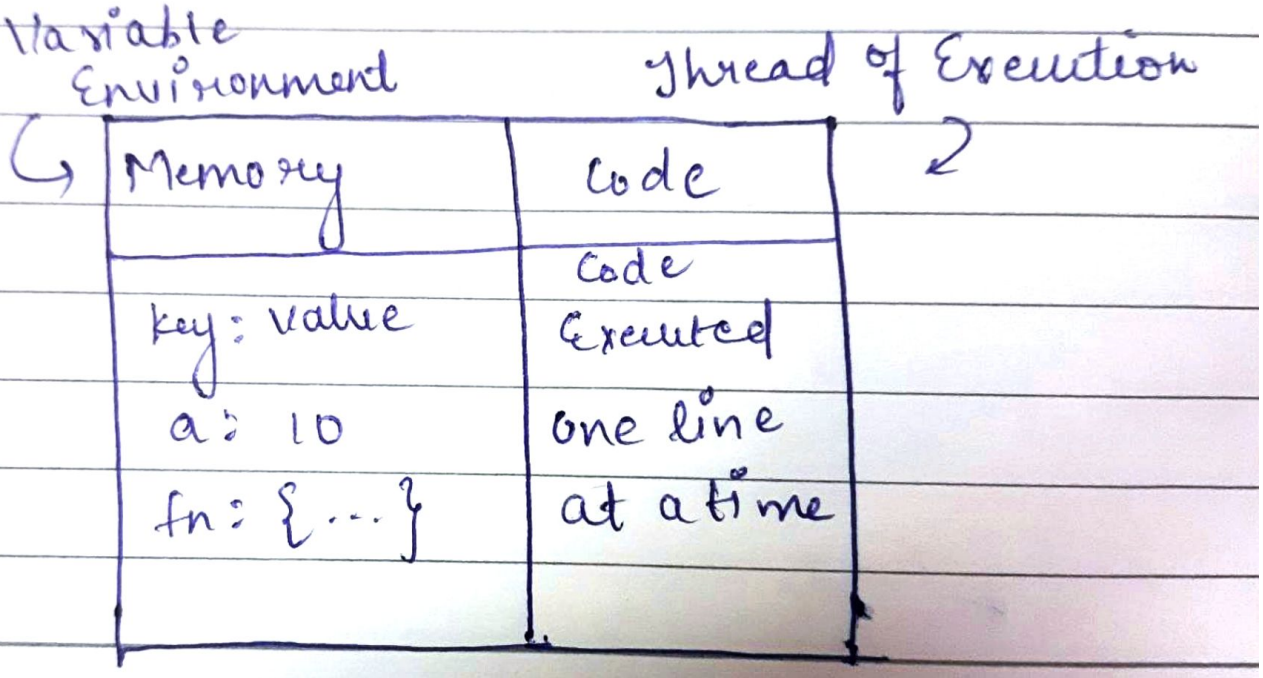

NAMASTE
JAVASCRIPT
NOTES

Lecture-1 How JavaScript works?

* Everything in javascript happens inside the execution context



DATE

* Javascript is a synchronous single-threaded language

Lecture-2 How Javascript is executed?

* Execution happens in two phases:-

(i) Memory Creation Phase

In this phase, all variables are initialized with undefined and function's code is stored in memory.

(ii) Execution Phase

In this phase, assignment of values ~~take place~~ to the variables take place. Function invokation is also done in this place

DATE

* A new execution context is created for every function invokation and is deleted after the execution of that function.

* The creation and deletion of various execution contexts is managed by the call stack.

* Call stacks maintain the order of execution of execution contexts.

Lecture - 3 Hoisting in Javascript

* Definition of Hoisting

It is a phenomenon of accessing variables or functions before even intializing it.

DATE

* Before starting the execution of code variables are initialized with undefined and the whole fn's body is stored in the fn name (Excluding arrow fns)

Lecture-4 How functions work in JS?

* When the program starts executing global execution context is created

* When a fn is invoked a new execution context is pushed into the call stack and variables created inside that fn has local scope

DATE

Lecture-5 Shortest JS Program

* When the program starts executing a global object (window object) is created in global execution context.

* this points to the global object in global execution context

* all the variables and f^n's created in global space are attached to the window object

Lecture-6 undefined vs not defined in JS

* undefined is allocated to variable while memory allocation phase

* Not a good pratise to assign a variable as undefined

DATE

Lecture-7 The Scope Chain, Scope & Lexical environment

* Scope is where you can access a variable or a function inside the code

* Whenever, an execution context is created, a lexical environment is also created.

* Lexical environment is the local memory along with the lexical environment of its parent.

* While accessing a variable, javascript engine looks for it in the local scope, if its not present then it checks the lexical env. of the parent. This check goes till the global scope.

DATE

* This chain of lexical environment is also known as the scope chain

Lecture-8 let, const and TemporalDeadZone

* Are let and const declarations hoisted?

⇒ Yes, they are. They are assigned memory just like var before code execution. But var declarations are attached to global object whereas let and const are attached to script object.

* What is temporal dead zone?

The time taken from hoisting till the initialization of variable (let & const) is called temporal dead zone

DATE

- If you access the variable (let or const) in the temporal dead zone you will get a reference error
- let and const are not attached to window object
- redeclaration not allowed in let and const (SyntaxError is thrown) but allowed in var.
- TypeError

  Happens when you try to change const variable later in the code
- SyntaxError

  Happens when const variable is not ~~defined~~ intialized.

* Reference Error
When you try to access let or const variable without initializing it

Lecture-9 Block Scope and Shadowing

* What is a block?
⇒ A block is used to combine multiple javascript statements into one group. It is used to combine multiple statements where javascript expects a single statement. For eg if statement

* What is block scope?
⇒ It means what all variables and functions we can access inside this block.

* let and const are block scoped whereas var isn't

DATE

```
{
    var a=10;
    let b = 20;
    const c = 30;
}
```

⇒ In the above code let and const are allocated memory which is reserved for this block during hoisting and var is allocated memory in global space

* What is shadowing in javascript?

```
var b = 100;
{ var b = 20;
  console.log(b);
}
console.log(b);
```

b is shadowed in second initialization, i.e the b of global scope is modified in the block.
(Does not happen with let or const)

DATE

* Scope chain works the same way for arrow functions as well

Lecture - 10 Closures (Most imp.)

A function bundled with its lexical scope is called closure

```
function x(){
  var a=7;
    function y(){
      console.log(a);
    }
  y();
}
x();
```

↳ In the above code, the local space of x forms a closure for y.

DATE

* The fn forms closure with its parent, parent's parent and so on.

Lecture-11 setTimeout + closures Interview Question

Q
```
function x() {

  for (var i=1; i<=5; i++) {
    setTimeOut (function() {
      console.log(i);
    }, i*1000);
  }
  console.log("Namaste Javascript");
}
x();
```

Output:-

6 // after 1 sec
6 // " 2 sec
6 // " 3 sec
6 // " 4 sec
6 // " 5 sec

=> The fn inside the setTimeout has the reference to the i (closure) By the time the for loop completes all the iterations the value of i becomes 6. After the timer of all individual fn ~~expire~~ expires it prints 6.

=> This problem can be fixed with using let instead of var as a new copy of i is created ~~of~~ for every iteration

=> It works with let because let is block scoped and var isn't

DATE

* How to implement the above functionality without using let?

↳

```
function x() {
    for (var i=1; i<=5; i++) {
        function close(n) {
            setTimeout(function() {
                console.log(n);
            }, n*1000);
        }
        close(i);
    }
    console.log("Namaste Javascript");
}
x();
```

↳ for every iteration f^n close has a new parameter

Lecture - 13 First class Functions

* What is a function statement?

```
function a() {
  console.log("a called");
}
```

* What is a function expression?

```
var b = function() {
  console.log("b called");
}
```

↳ Major difference b/w both is 'a' can be hoisted but 'b' will have undefined while hoisting.

* What is Function Declaration?

↳ Function declaration is same as fⁿ statement

* What is an anonymous function?

↳ Function without a name is called anonymous function.

They are used when fⁿ are used as values, same as in fⁿ expression

* What is Named Function Expression?

```
var b = function xyz() {
    console.log("b called");
}
```

Q. var b = function xyz() {
    console.log(xyz); // prints xyz()
}
xyz(); // reference error

Why does reference error show on calling xyz()?

↳ Because xyz is not declared in global scope. It can be only accessed inside xyz().

Q. Difference between Parameters & Arguments?

parameters

→ var b = function (param1, param2) {
    console.log("b called");
}

b(1,2);
↳ arguments

DATE

* What are first class functions?

↳ The ability to use functions as values are known as first class functions.
(functions can be passed as an argument to another fn and can be returned by a fn)

* First class citizens is same as first class functions.

Lecture-14 - Callback Functions in JS ft. Event Listeners

* What is a callback function?

↳ When a fn is passed as an argument to another fn, it is called callback function.

DATE

For eg

```
setTimeout (function () {
    console.log ('timer');
}, 5000);
```

↳ the first parameter passed to setTimeout is the callback function.

↳ Callback functions gives us the power to run f^n's in an asynchronous manner.

* Blocking the main thread.

↳ When a f^n takes a lot of time to execute another f^n's cannot be executed in the javascript engine, at the same time. The main thread is blocked for the time being.

DATE

* Event listeners

↳ Whenever an event occurs, the callback f^n is passed to the javascript engine to get executed.

Q Why do we remove Event listeners?

↳ Because they consume memory as it needs memory to hold the variables and f^ns in its closure.

Lecture-15 Asynchronous Js & Event Loop

* Javascript is a single threaded language which can only run one thing at a time

DATE

↳ To use certain superpowers of the web browser we need the aucess of WEB APIs.

Some of the web apis are:-

* setTimeout()
* DOM APIs
* fetch()
* localStorage
* console
* location

* Event Loop:-

↳ The job of event loop is to continuously monitor the javascript engine.
If the call stack becomes empty it pushes the callback f^n from callback queue onto the call stack

DATE

* Callback queue:- (Task Queue)

It stores the callbacks which needs to be executed in javascript engine.

For eg

↳ In setTimeout, whenever the timer expires, the callback fn is pushed into the callback queue.

↳ In event handlers, whenever the event is performed, the callback fn is pushed into the callback queue

* MicroTask queue

It is same as callback queue, but has a higher priority.

DATE

Q What all callbacks goes into the microtask queue?

↳ (i) promises returned by api calls

(ii) mutation observer (callbacks which gets called ~~to~~ on DOM tree mutation).

* Event loop will only give opportunity to the callback queue only after all the callbacks are executed in microtask queue.

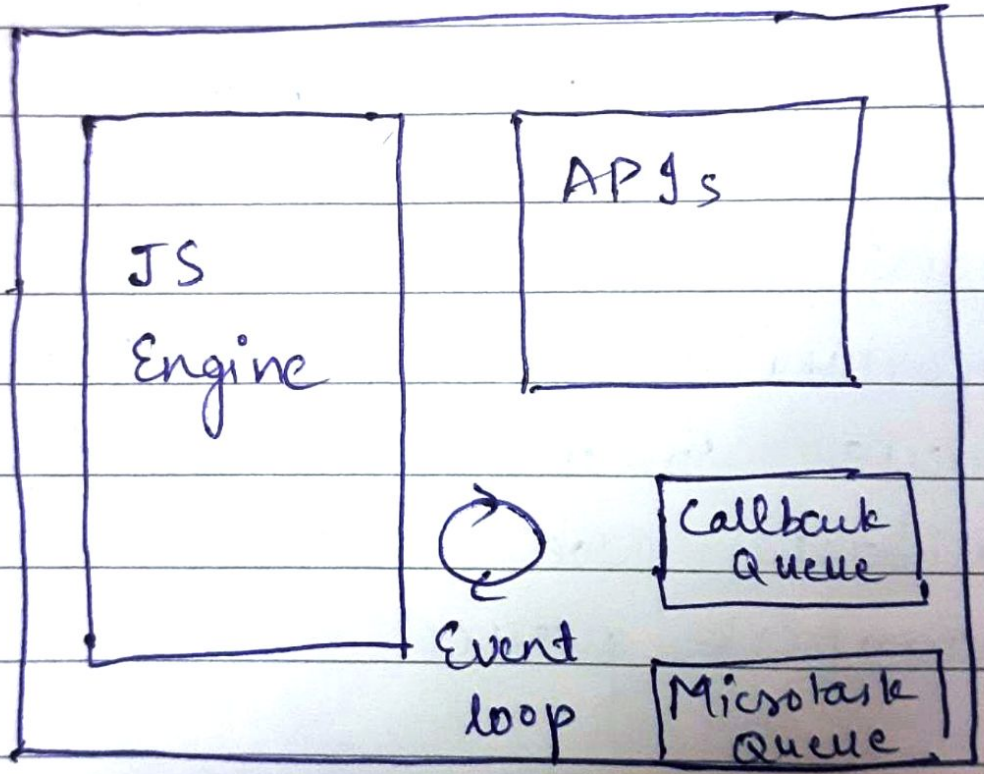
Q What is starvation?

↳ When the callback queue doesn't get a chance to execute due to large no. of functions in microtask queue for a long time, this condition is called starvation

DATE

Lecture-16 JS Engine Exposed

- To run a javascript code on any platform we need a Javascript Runtime Environment

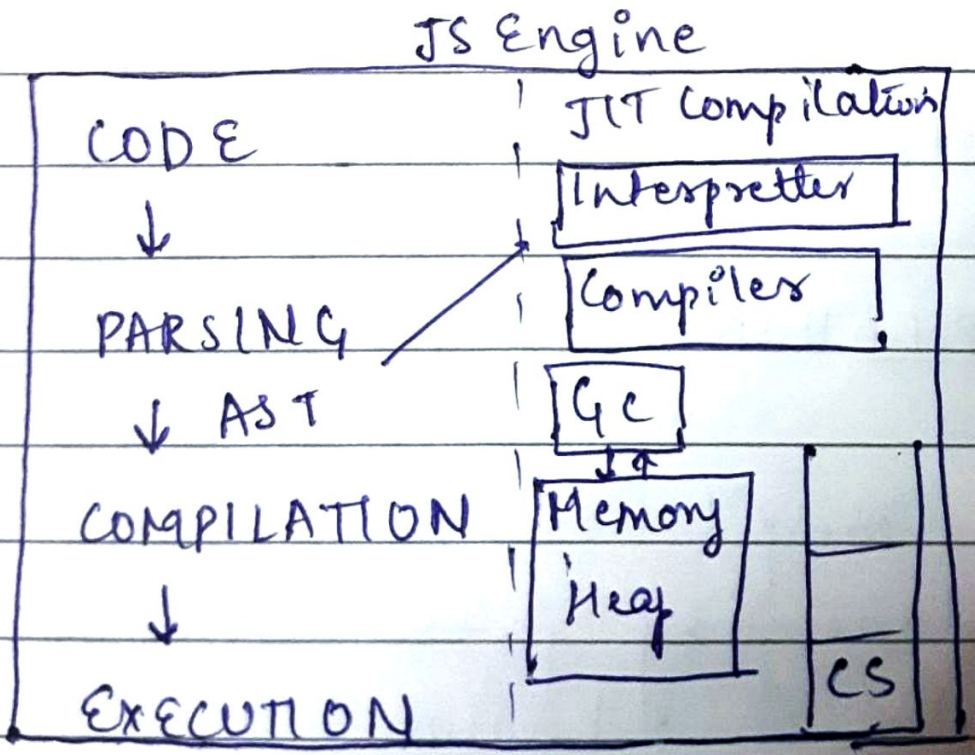


Javascript Runtime Environment

* Javascript Engine is the heart of Javascript Runtime Environment

↳ V8 is the javascript engine inside chrome and nodejs.

Javascript Engine Architecture



① Parsing :-

↳ In this phase, the code is broken down into tokens and is passed through a syntax parser.

↳ Syntax Parser takes the code and converts it into an AST (Abstract Syntax Tree)

② COMPILATION and Execution

↳ Both compilation and execution is done by javascript engine which is known as J.I.T Compilation (Just In Time).

↳ Interpreter converts our high level code to byte code

DATE

along with the compiler which optimises the code. and then it proceeds to execution phase

↳ Execution is done with the help of memory heap and the call stack.

↳ Garbage Collector inside Memory Heap uses Mark and Sweep Algorithm to free up memory.

Lecture-17 Introduction to Functional Programming and Higher-Order Functions

Q What is a Higher Order Function?

↳ A function which takes a function as an argument or returns a function from it is known as Higher Order Function.

Lecture-18 Map, Filter and Reduce

① Map function

↳ Transforms each element of the array and returns a new array

For eg

```
const output = arr.map(function
                double(n){
                return n*2;
                });
```

↳ the function passed to the map doubles each and every element in arr and returns a new array with modified elements.

DATE

(2) Filter Function.

↳ Used to filter elements inside the array.

For eg

```
const output = arr.filter((n) =>
                              n%2);
```

// this filter method returns all the odd numbers as an array.

(3) Reduce Function

↳ Iterates on every elements to perform a certain operation and return a final value.

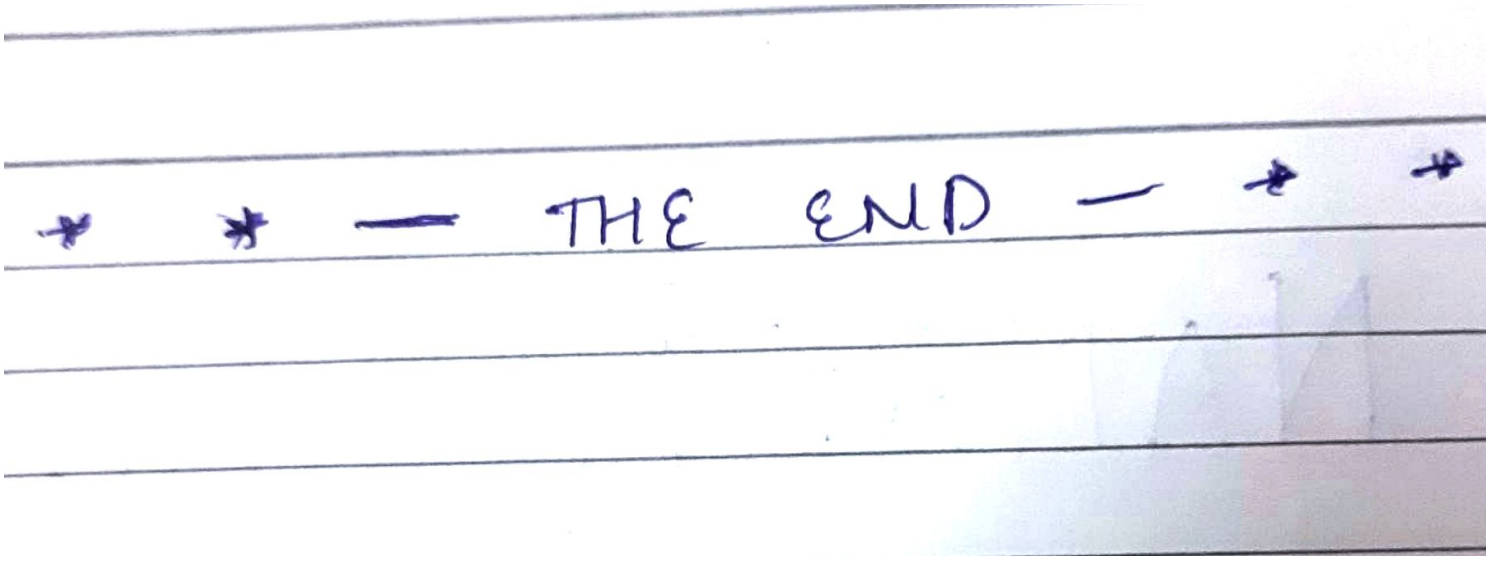
For eg

```
const output = arr.reduce(function
                (acc, curr) {
                acc = acc + curr;
                return acc;
                },0);
```

// curr is the current element and accumulator stores the sum upto curr for every iteration.

↳ It finally returns the sum of all the elements in the array

↳ Second parameter of reduce fn is the initial value given to the accumulator.


— THE END —