EGMA :—

DATE
PAGE

latest version:— 2025 (ES : — Egma Script)

JAVASCript → why, what, How.

1995 (initial version in market)
At that time, it is only client side

- ⇒ It is Both a client side (more recomanded) & Server-side language (need to install Node.js)
- It is verstalie & dynamicalley typed Programming language.
- Dedeloped By Brenden Eich

Features :—

1) • Javascript is function-Oriented, Objed-Oriented. multiparadigmm Scripting language.
2) • It is client-side as well as server-side Scriptling language.
3) • Dynamicalley-typed Scriptling language.
   {no need to declare the data type}
   {type of is of variable resolved at Runtime}
   {No keyword available in JS}

• Typed Datat	• Dynamicalley-typed Data.
resolve at Compile time	resolve at Runtime.

JS :- very Simple language

DATE
PAGE

In dynamically-typed :- it is difficult to resolve the error, we can't identify the nature of data at specific time.

(JavaScript)

4) • It has Capability to Convert HTML (Static web-page) into DHTML.

↓

Dynamic-web-page (with help of JS)

"JS is Synchronous (one-thread at a time) By-Default"

Interview-question :-

- Asynchronous / Synchronous ?
- Interpreted / Compiled ?
- How Single-threaded ?

Ans :- it Depends on the Browser.

5) • JS is a Single-thread Language.

How to write JS in HTML Document ?

Using {<Script> Tag} ← HTML Tag.

```
<script>
    JS code.
</script>
```

where to write JS in HTML

In <body></body> tag.
just before the closing of Body tag.

<body>

<script>....
</script>
</body>

{Agr upper likhege TO vo HTML Code ko load hone, me time lgega, kyuki JS on single threaded HE TO Jabtk vo thread run hogo, Baki code Block Ho jata Hai.}

So, :-

HTML
JS

#format :-

<HTML>
<head> ... </head>
<body>
≡ ← HTML

<Script>
≡ ← JS
</Script>
</body>
</body>
</HTML>

NOTE :-

Browser → Software (JAVAScript Virtual Machine [JAVAScript Engine]

⇒ it responsible to execute the JS code in HTML and if error occure, it deted it.

LiveScript → (converted into) JAVAScript
inspired by Java (when Java is very popular in 1995)

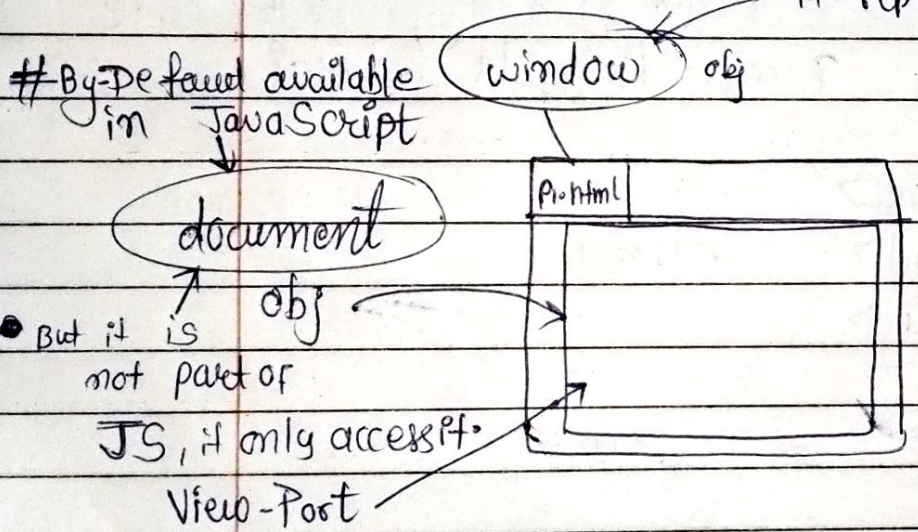
⇒ JAVAScript Virtual Machine ⇐

JAVAScript Engine

- → Ex → Chakra
- → ME → Chakra-Core
- → GC → VS - engine
- → MF → Spider-Monkey

⇒ Interview question

How JavaScript Engine Internally work?



Document → object → It has some properties and Behaviours

- → write()
- → writeln()
- → getElementById()
- → querySelector()
- → CreateElement()
- → getElementClassName()

```
<html>
<body>
<script>
document.write("JAVA");
</script>
</body>
</html>
```

DIF functions & methods.

Why it is called Scripting Language ?

DATE
PAGE

A scripting language that:
- is interpreted (run line by line, not Compiled first).
- is used to control or automate another Program.
- usually runs inside a host environment (like browser)

Why JAVAScript :-
- It runs inside the browser.
- It is interpreted, not Compiled
- It was made to add behaviour to web Pages.
- No main method or entry Point.

Today JS is also programming language, used for Backend (Node.js)

But name "Scripting Language" stayed bcoz of its origin.

- NOTE :-

① " " } ' ' } Both treats As a String

② ; (optional). if you using (;), than use in whole Program, otherwise donot use it, Bcoz it is bad Programming Practice.

③ HTML code in JS, than we write in into " " String format.

for example :-

```
<Script>
    document.write("JAVAScript")
    document.write("<br JAVA")
    document.write("<h1> Hello <\h1>")
<\Script>
```

{Console.log("Hello") :- Print message on Browser Console}

BrowseConsole → console obj

It is object of Browser Console.

(HTML Document) factory of document obj

↓

"<br>" + document.Constructor.name. " // print the factory name of document

Window

↓

"<br>" + window.Constructor.name " //

factory ⇒ Window.

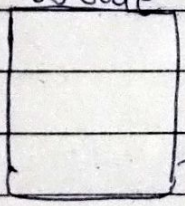
BrowserConsole

↓

"<br>" + Console.Constructor.name" // factory ⇒ BrowserConsole.

Declare variable in JS.

```
<Script>
var a = 20;
var b = 10;
var c = a+b;
document.write("addition"+c);
</Script>
```



Internally How it executes the JS code

① Parsing
- → lexical analysis
- → Syntax analysis

lexical analysis :- Converting the JS code into small Token (Tokenization)

↓

words.

Syntax analysis :- Token → AST (Abstract Syntax Tree)
Converted into

After Syntax analysis, we get

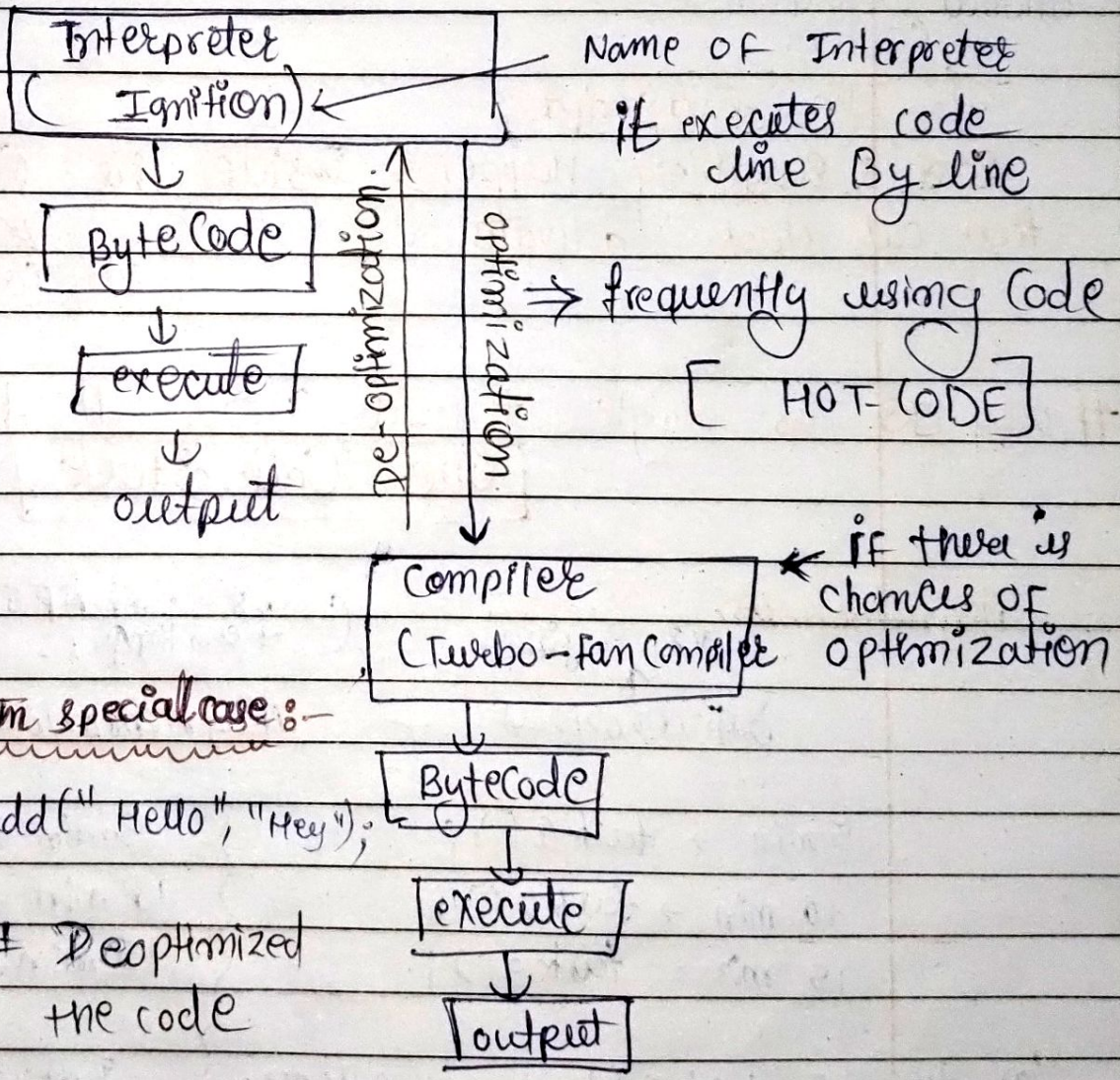
⇒ AST (it executes finally)

Working flow of Execution of JavaScript Code.

* Before execution *

AST

Comes into execution




```
add(a,b){
  ≡
}
add(10,20);
add(20,30);
.....
n
```

In special case :-

add("Hello", "Hey");

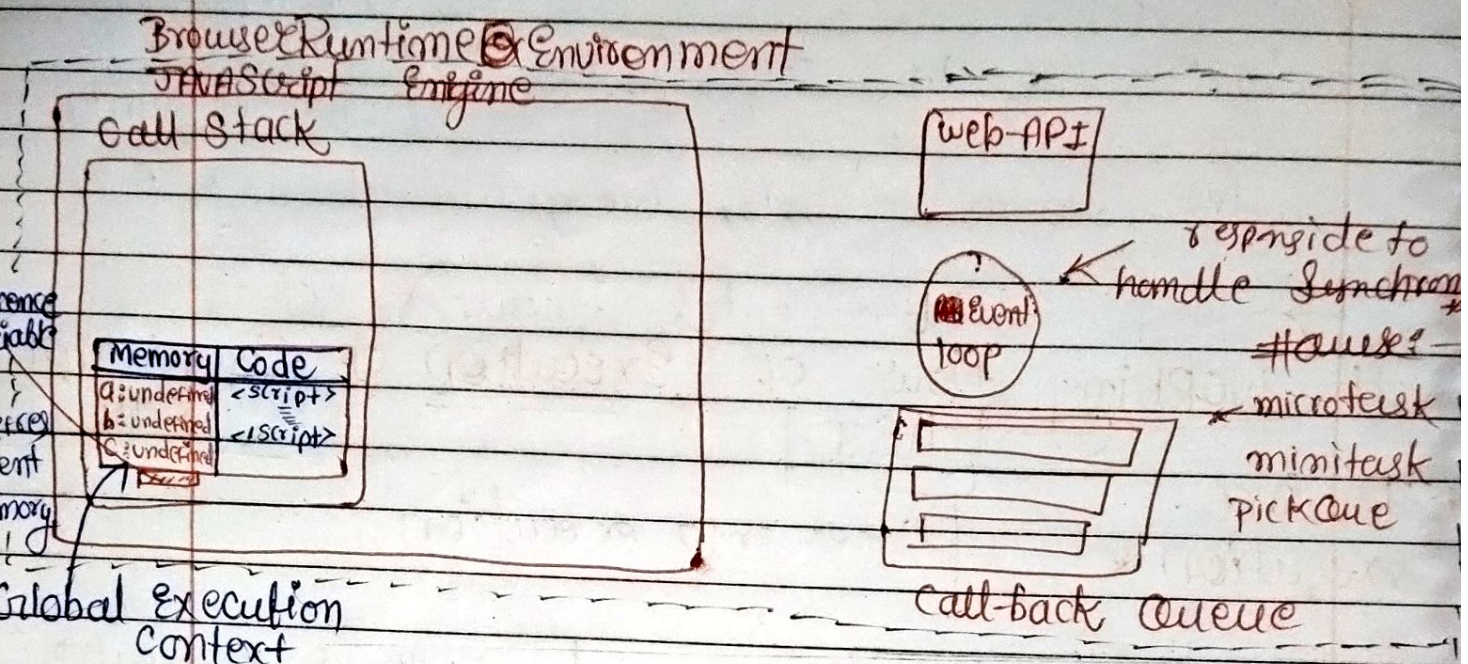
it Deoptimized the code

● NOTE :-

tum frequently numeric data bhej rahe the to code optmized Ho gaya, lekin humne agr String Bhejo to vo. fir de-optmized Hona jarruri He.

refroll

⇒ what is CallStack ?.
⇒ use of eventloop ?
⇒ internal working of execution of JS code ?



In JavaScript :-
Each & everything Happens inside the CallStack.
first call-stack activates

Key Component OF Javascript

Why JS is Single-thread Broz it has only one Call-Stack.

Asynchronous (v8 engine) (v8 + web API + eventloop) = execution

Synchronous + ASynchronous = JSA

```
5min → task1();
10 min → task2();
15 min  task3();
```

in this if task1() is executing, than other thread have to wait.

```
5min →  task1()
10 min →  task 2()
15 min→  task 3()
```

But in this all threads runs parallely.

V8 + libuv = Node JS.

{ if we executes & writes the Asynchronous code in Node }

DATE
PAGE

(1) V8 Engine is not capable to execute to the Asynchronous Code.

(2) V8 Engine executes only Synchronous code.

* V8 ⇒ Responsible to executes Synchronous code.

* (V8 + web-API + eventloop) ⇒ Responsible to executes Asynchronous Code.

NOTE :-
If we executing & writing the Asynchronous code in Node.js environment than, if executes By V8 + libuv

Undefined :- Data-type.
if variable is declare But not ~~undefin~~ initialized.

Note :-
{ when we call any function, it creates a different & Separate context corresponding to each function. }

{ if the execution is done, than the GEC poped-up from Call-Stack : }

In js :-

var :- Keyword

Primitive Data Type :-

- Number (10, 3.5)
- String ("Hello")
- Boolean (true, false)
- Null (let y = null)
- unDefined (let x)
- BigInt (123n)
- Symbol (Symbol("id"))

Non-Primitive :-

- Class
- Array
- Enum

(parent) Global-Execution-Context

↓

(child) other execution-Context

Interview question :- Lexical Environment ? loopchain ?

"CONCEPTS"

Variables :-

Variable Store data.

Keyword	Scope	Can change value
var	Function	Yes
let	Block	Yes
const	Block	NO

Example :-
```
let name = "Neha";
const age = 22;
```

Data Types :-

JavaScript is dynamically typed.

Primitive :- String, Number, Boolean, Null, undefined, BigInt, Symbol

Operators :-

(1) Arithmetic Operators :- +, -, *, /, %, **

(2) Comparison Operators :- (==) Compares value only, (===) compares value as well as data type, !=, !==, >, <, >=, <=

(3) logical Operators :- &&, ||, !

Conditional Statement :-

(1) if - else
(2) else if

Loops :- loops repeat Code.

(1) for loop
(2) while loop
(3) do - while loop

Functions :-

functions are reusable bbocks of code

(1) Normal function :-

```
Function add.(a, b) {
    return a+b;
}
Console.log(add(2,3));
```

(2) Arrow function :-

```
const multiply = (a,b) => a*b;
```

Arrays :-

Array stores multiple values in one variable

Ex :- let fruits = ["Apple", "Banana", "orange"];

(1) Accessing elements :-

Console.log (fruits[0]);

(2) Modifying elements :-

fruits[1] = "orange";

(3) Array length :-

fruits.length;

Common Array Methods :-

(1) Push() :- add at end.

(2) POP() → remove from end

(3) unshift() → add at start

(4) shift() → remove from start

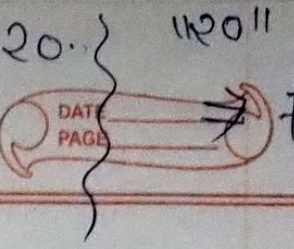
(5) forEach() →

fruits.forEach(fruit ⇒ Conse.log(fruit));

Number — 5.12, 5.0, 5

"20" == 20 ⇒ True

"20" === 20 ⇒ false



Datatype

Primitive

- Number
- String
- boolean
- undefined
- Null
- BigInt

(# important questions in interview)

Non-Primitive

- Array
- object
- Class.

function.

⇒ typeOF(a); // Identify the type of a (variable)

operator :- (==) compares only value.

(===) compares value & Datatype

Undefined

① TypeOF(undefined) is undefined.

② Both represents lack of value

* (undefined == value) ⇒ true.

Null

typeOF(null) is object.

* undefined === null ⇒ false

Type of NaN = Number.

DATE
PAGE

③ (20 + undefined);
⇒ NaN

④ undefined + null;
⇒ Nan (Result)

SOME MCQs ⇒

(A) (20 + null);
⇒ 20
in this case.
• null is Converted into 0

(B) 20 * null;
result ⇒ 0

(C) '20' + undefined;
result: 20undefined

(D) '20' + null;
Result:- Null

(E) 0 == null;
:- false

(F) false == 0;
Result :- true

(G) 20 + true;
⇒ 21

(H) 1 == true;
⇒ true;

(I) Nan == NaN
false

NaN ko jab bhi == se compare karenge To vo By default hi false hi return karega.

(j) NaN != NaN
Result ⇒ true.

(k) '20' + 10;
Result :- 2010

(L) '20' - 5 ;
Result :- 15

(M) '20' * 5';
Result ⇒ 100

(N) '21' % 2 ;
Result ⇒ 1

(O) '20' == 20 ;
true

(P) '20' == '20';
Result ⇒ true

(<) ⇒ <
(>) ⇒ >

• "< br >" •

Those Operations are of Arithmetic Operators.

In javaScript :-

Logical operators return the last value.
like

* 200 && 100 ;
Result ⇒ 100

* 0 && 100 ;

* false && 100 ;
⇒ false

* True && 100 ;
⇒ 100

⇒ IF else ⇐

In javaScript :-

① in JS "0" means false.

② var x = 20;

```
if (x)
  =
else
  =
```

Result ⇒ True.

```
var x = 0;
if (x)
  = (True)
else
  = false
```

Result ⇒ False.

③ ['' , " "]

Result :- falsy value.

```
var x = '' | " ";
if (x)
  (true);
else
  ( false);
```

Result = false.

④ var x = undefined;

```
if (x)
  True
else
  false
```

Result :- false.

⑤ var x = null;

```
if (x)
  True
else
  false
```

Result :- false

DATE
PAGE

(6) var x = 1n
if(x)
True;
else
false;
Result :- True

(7) var x = On;
if (X)
True.
else
false
Result :- false.

(8) var x = Nan;
if (x)
True;
else
fals e;
Result :- false.

* 1/0
Result :- Infinity.

Inhi 6 ka nth. false ⟹
Baki Sab true;

① 0
② "" / '' ''
③ undefined
④ null
⑤ false
⑥ Nan.

Alert Box :-
Prompt(" Enter the value "); // always String value in prompt.
⇒ It is a function.
⇒ [it is Basically a ~~Pop~~ Alert Box]

⇒ it uses "===" operator :

// JAVAScript Coercion.
n = n * 1; // It treats it like Number

⇒ ① + Prompt
② Parse Int
③ n = n * 1

DATE
PAGE

* JavaScript Coercion :-

Converting numeric String into Number using * by 1

n = n * 1 ;

Case label ko variable bhi bna sakte.

a = 2 , b = 1

Like.

Case a+b : - - - - - - - ;

Duplicates bhi kar sakte ⇒

Case 2 : - - - -
Case 3 : - - - // Pehle wala case 3 chl jayenga
Case 3 : - - - - -

* In javaScript, the case label can be number, string, boolean, bigInt, undefined, null, ~~NaN~~ 1 = 1 (switch (true))

Switch Case "===" operators use karta hai :

x = 100 ; (scanning nhi Hoga, jab yeh line execute Hogi tab existance me ayega.)
var x = 100 ;
let x = 100 ;
Const x = 100 ;

⇒ NaN isliye match nhi hoyega kyunki vo By default false ⇓

var let Const

Variable Concept's :—

DATE
PAGE

* Hoisting *

* Let And Const are also Hosted.

* Temporal DeadZone :—

① me variable uninitialize Hota

② variable ko access nhi kar sakte,

③ Agr karenge to error generate karega

```
Console.log(x);
let x = 404;

Result = Error
```

⇒ TDZ

⇒ x : < uninitialize >

① var x = 404;
Console.log(x); Result ⇒ 404
⇒ 404

② ~~var~~ x
Console.log(x);
var x = 400; // Result :- undefined
var is Hosted.

Hosting is the default behaviour OF JAVAScript,

In case of Hosting, All declaration move to the Top OF screen.

var x ; ← declaration)
~~var~~ x = 100 ; definition

var x = 100; // declaration + definition.

Temporal Deadzone :-

Basically a time period.

Variable declaration ——— variable Initialization

Temporal Deadzone

```
Console.log(x);
≡
let x = 100;
≡
console.log(x);
```

x = 100

→ Temporal Dead zone.

Variables Concepts :-

var x = 100;

↓

Global space

x enters in global space.

we can change the value

let, Const ← constant

↓

creates their own memory space..

① # var x = 100;

var x = "Hello";

// we can redefine the variable with same name

Harmfull

var is Hosted, let, and const is also Hosted But they enter in Temporal dead zone

So, if you are using let, const, first initialized it than use it.

DATE
PAGE

② # let x = 100;
let x = "Hello";

// cannot redefine the let

③ # if (true){
var x = 100;
}
console.log(x);

// Bcoz it is function at Scope.

Function-level-Scope.

output:- 100;

④ # if (true){
let x = 100;
}
console.log(x);

// Bcoz, let is Block-level.

output :- generate error

⇒ Global-Execution-Context
(anonymous)

* Array *

- It is flexible in size and in storing data.
- It is Based on indexing.
- In JavaScript, the Array is Dynamic in size.

Syntax :-

* let arr = [10, 20, 30, 40];

* let ar = [10, "Hello", true];

(1) for In loop for traversing each element :-

```
for (let index in arr)
document.write("<br>" + index + ": " + arr[index]);
```

(2) forOff loop for access the direct element

```
for(let element of arr)
document.write("<br>" + element);
```

Type of array is Object

DATE
PAGE

(3) for Data manipulation :-
Here are some Methods :-

(1) push (1000); // It add the element at the end.

(2) unshift (5000); // add element at the start.

(3) pop() ; // remove the element at the last

(4) Shift(); // remove the element at the start.

(4) # for Adding the element at particular index
Splice

* Splice () ; // you can add and remove element.

" * splice (2, 0, 500) ;
↓ ↓ ↘ element add at second position
inder position → Delete Count "

" * Splice (2, 2) ; // it will delete 2 element from the second index.
↓ ↳ Delete Count
index "

(5) :- for searching :-

includes ( ) ; // return type Boolean

Predicates, functions

(6) indexOF(5, 1); // for getting the index of particular element.

(7) Delete arr[2]; (Delete Operator)

it delete the element at 2nd index, But it does not delete the index position.

example :-

let arr = [10, 20, 30, 40, 50];

delete arr[2];

output :- 10, 20, , 40, 50

* filter Method *

{ it internaley creates an array.

let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let output = arr.filter((element) => { return element % 2 == 0 })}

Definition :-

{ The filter() method in JS is used to create a new Array that contains only those elements which satisfy a given Condition. }

" In javaScript :- functions are the objects "

Predicates function return either true & false, if the given condition is satisfied or not.

for loop Dif Map. Dif for In and for OF loop

Dif filter vs map.

DATE
PAGE

(1) find( ); // we can check is the element present or not. if not present than return undefined.

(2) findIndex( ); // Return the element Index. if not found than return (-1).

(3) Some( ); // checks if any element in array meets a condition and returns true or false

(4) every( ); // All elements must pass and meets a condition.

⇒ Some( );

Syntax :—

```
array.some( function (element, index, array)
            { return Condition; });
```

or `array.some (element => Condition);`

Example :- check if any number is even

```
let numbers = [1, 3, 5, 8];
let hasEven = numbers.some(num => num
                                % 2 == 0);
document.write (num);
```

or.

```
console.log(hasEven); // true.
```

Methods :-

① # Filter () Method :-

⇒ what does filter() method do ?

- Returns a new array.
- Includes all elements that pass the test.
- Does Not modify the original array.
- If no elements matches → returns an empty array [].

Syntax :-

①
```
array.filter ( function(element, index, array){
    return Condition;
});
```

(2) using arrow function.

"array.filter(element ⇒ condition);"

filter even Number

Example :— let numbers = [1,2,3,4,5,6];

let evenNumbers = numbers.filter(num ⇒
num % 2 == 0);
document.write(evenNumbers); // [2,4,6]

Example :— # filtering objects

let users = [
{name: "Aman", age: 17},
{name: "Neha", age: 22},
{name: "Ravi", age: 19}
];
let adults = users.filter(user ⇒ user.age >= 18
document.write(adults);

Returns only users whose age is 18 or above.

	Method	Returns	Purpose
(1)	filter()	New array	Get all matching elements.
(2)	Some()	Boolean	check if any elements matches.

(2) map() method :-

⇒ map() method in JS is used to transform each element of an array and return a new array with transformed values.

⇒ what does map() do ?

(1) Returns a new array.
(2) Transforms every element in the array.
(3) Does Not modify the original array.
(4) Output array Length = input array length.

Syntax :-

① array.map(element ⇒ newValue);

② array.map(function(element, index, array){
    return new value;
});

Example :-

① Double each Number :-

```
let numbers = [1, 2, 3, 4];
let doubled = numbers.map(num ⇒ num * 2);
document.write(doubled); // [2, 4, 6, 8]
```

② working objects :-

```
let users = [ { name : "Aman", marks : 80 },
              { name : "Neha", marks : 90 } ];

let names = users.map( user => user.name );
document.write( names ); // [ "Aman", "Neha" ]
```

(3) reduce() method :-

This method is one of the most powerful array methods in JS. It is used to reduce an array to a single value by applying a function on each element.

⇒ what it does ?

- Returns one single value (number, object, array, string, etc)
- Executes a reducer function on each element.
- Uses an accumulator to store the result.
- Does Not modify the original array.

Accumulator :- , the accumulator is a variable that;

- Stores the result of previous iterations.
- gets updated on every loop.

- finally returns the single output value.

Think of it as a Container that keeps collecting values.

Syntax of reduce :-

```
array.reduce((accumulator, currentValue, index,
                array) => { return updatedAccumulator
}, initialValue);
```

Example :- Sum of all number.

```
let numbers = [10, 20, 30, 40];
let sum = numbers.reduce((acc, curr) => acc +
                curr, 0);
document.write(sum); // 100
```

function ↓ object

function in function

DATE PAGE

let Consumter = { } ← objet notation.

Difference :— (1) { id : 100 } ← JAVAScript obj

(2) { "id" : 100 } ← JSON

How to Represent Real world Entity :—

let customer = { "id":100, "name": "Neha", "age": 20 };

↑

JSON String

How to Access & Print the elements :— + customer

* document.write("
 name:" + ["Name"])
* document.write("
 name:" + customer.name);
* for(let key in customers){
  document.key("<br>" + key + " : " + customer[key]
  }

Array :—

let customer = { "id":100, "name": "Rahi", "age" : 23 };

Bactics :—

for(let customer OF customerlist){
d.w ('
 Id : ${ customer.id } Name : ${ customer.name });

⇒ Spread Operator ⇐

DATE
PAGE

How to Clone the array ?

One-D Array

```
let arr1 = [1,2,3,4,5,6];

let arr2 = [...arr1];

arr2.push(1000);
dow(arr2);
```

Spread operator [...]

use for cloning the array.

It will use & create Shallow Copy, if there is any, non-primitive or reference type data so it will take the reference.

reference wha ke copy karta hai

Two-D Array :-

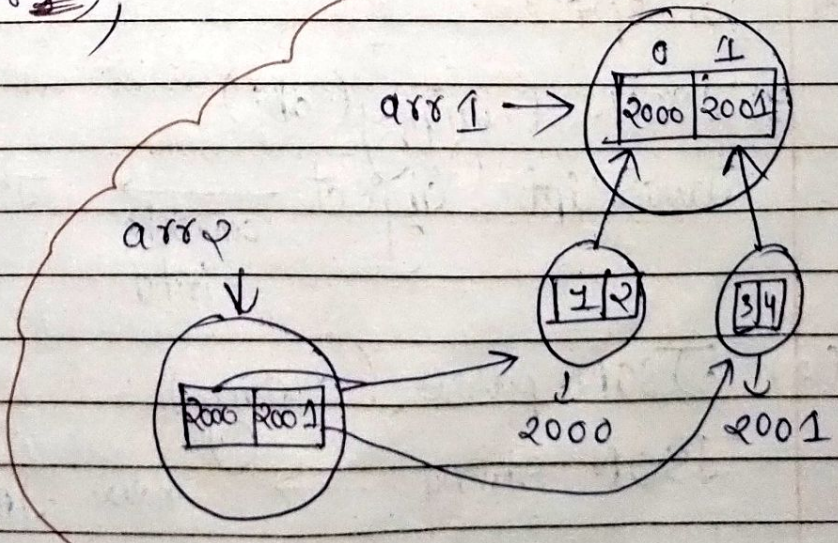
```
let arr1 = [[1,2],[3,4]];

let arr2 = [...arr1];

arr1.push(1000);
dow(arr2);
```

Shallow Copy in Memory.

```
let arr1 = [[1,2],[3,4]];
let arr2 = [...arr1];
```



object Notation Cloning :-

```
let c1 = {"id": 100, "name": "Neha"};
let c2 = {...c1, "age": 23};
d.w(c2.id + "  " + c2.name + "  " + c2.age);
```

Deep Copy creation :-

```
let arr1 = [[1,2], [3,4]];
let arr2 = JSON.stringify(arr2); // "[[1,2],[3,4]]"
arr2 = JSON.parse(arr2);
arr1[0].push(1000);
d.w("<br>" + arr1);
d.w("<br>" + arr2)
```

convert object → string : stringify

convert string → object : parse

When to use stringify And parse ?

```
let copy = JSON.parse(JSON.stringify(obj));
```

Concept :-

(1) JSON.stringify(obj)

JavaScript object —converts into→ JSON string

(2) JSON.parse(string)

JSON string → new JavaScript object

JAVASCRIPT FUNCTIONS

Syntax :-

```
function functionName (p1, p2) {
    code - - - - -
    return result;
}
```

each and every java script function return a value. By default it return undefined.

function call karte waqt, parameters pass karna mandatory nahi Hai

C=0 (default argument) we can also do this)
C=100(

Terminalogies :-

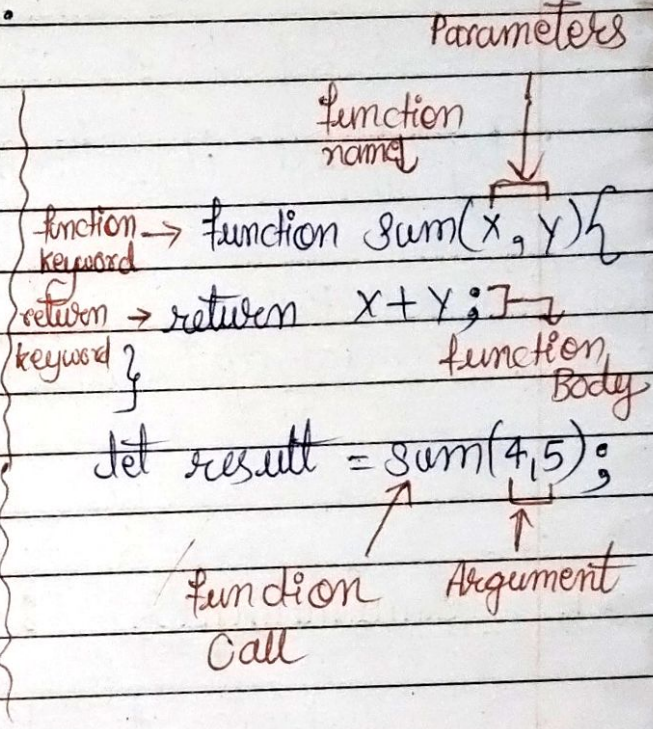
1. what is function
2. Function Statement
3. function Expressions
4. Arrow function
5. IIFEs (Immediately Invoked function Expressions)
6. Annonymous function.
7. Callback function (BackBone of JS)
8. error callback function
9. Higher order fundion
10. function Constructor
11. first - class function

Functions in JavaScript

- Functions in JavaScript are reusable blocks of code designed to perform specific tasks. They allow you to organize, reuse, and modularize code.
- It can take inputs, perform actions, and return outputs.

Why use functions :-

(1) Code Reusability,
(2) Modularity,
(3) Readability,
(4) Maintainability



Understanding functions :-

In functions, parameters are placeholders defined in the function, while arguments are the actual values you pass when calling the function.

Example :-

```
function greet(name){ // 'name' is parameter
document.write("Hello" + name);
}
greet("Alice"); // "Alice" is argument
```

- Argument : Actual value given at call time).

Concepts :-

① Default Parameters :-

- Default parameters are used when no argument is provided during the function call.
- If no value is passed, the function automatically uses the default value.

Example :-

```
// function greet (name = "Guest"){
    document.write ("Hello," + name);
}
greet(); // Hello, Guest (default used)
greet("Aman"); // Hello, Aman.
```

② Return Statement :-

- The return statement is used to send a result back from a function.
- When return executes, the function stops running at that point.
- The returned value can be stored in a variable or used directly.

Example:-
```
function add (a, b){
return a+b ; // returns the sum
}
let result = add (5,10);
document.write ( result); // 15
```

Types of functions :-

(1) Named functions :-

A function that has its own name when declared. It's easy to reuse and debug because the name shows up in error messages or stack traces.

Example:-
```
function greet(){
return "Hello";
}
d.w( greet()); // Hello
```

(2) Anonymous function :-

- A function that doesnot have a name.
- It is usually to a variable or used as a callback. Since, it has no name, it cannot be called directly.

Example:-
```
Const greet = function(){
return "Hi there";
};
d.w( greet()); // Hi there.
```

(3) Function Expression :-

when you assign a function (can be named or anonymous) to a variable. The function can then be used by calling that variable.

Example:- Const add = function(a, b){
    return a + b;
};
d.w(add(2, 3)); // 5

(4) Arrow function (ES6) :-

- A new ~~arr~~ way to write functions using the => syntax. They are shorter and do not have their own this binding, which makes them useful in some cases.

let square = n => n * n;
d.w (square(4)); // 16.

5) Immediately Invoked Function Expression (IIFE):-

IIFE executed immediately after their definition. They are often used to create isolated scopes.

Ex:- (function (){
    d.w("This runs immediately");
})();

Immediately Invoked functions Expressions (IIFE) are JS functions that are executed immediately after they are defined. They are typically used to create a local and scope for variables to prevent them from polluting the global space.

Syntax :-

```
(function(){
    // function Logic Here
})();
```

(6) Callback functions :-

A callback function is passed as an argument to another function and is executed after the completion of that function.

Example :-

```
function num(n, callback){
    return callback(n);
}

const double = (n) => n * 2;
d.w(num(5, double)); // 5, 10
```

(7) Constructor function :-

A special type of function used to create multiple objects with the same structure. It's called with new

Example :-

```
function Person(name, age){
    this.name = name;
    this.name = age;
}

let usere = new Person("Neha", 22);
document.write(user.name); // Neha
```

(8) Recursive Function :-

A function that calls itself until a condition is met. Very useful for problems like factorial, Fibondcci, or tree traversals.

Example :-

```
function factorial (n){
    if(n === 0) return 1;
    return n * Factorial (n-1);
}
dow( factorial(5)); // 120
```

(9) Higher-Order Function :-

⇒ A function that either takes another function as a parameter or returns another function. These are common in JS (e.g., map, filter, reduce).

Example :-

```
function multiplyBy ( factor){
    return function(num) {
        return num * factor;
    };
};
```

```
let double = multiplyBy (2);
dow( double(5));
output :- // 10
```

(10) Error-first Callback function :-

Used mostly in Node.js. (whose first argument represents the error).

Rule :-

callback(error, result).

- first parameter → error
- Second parameter → success data.

Example :-

```
function divide (a, b, callback){
if ( b === 0){
 callback(" Cannot divide by zero", null);
} else {
    callback( null, a/b);
  }
}
divide ( 10, 2, function (error, result){
  if ( error){
       d.w ( " Error :", error);
} else {
     d.w ( " Result :", result);
}
});
```

(11) First-class function :- (function behaves like a value)

Functions are treated like variables
A function can be :
- ✓ Assigned to a variable.
- ✓ Passed as argument.
- ✓ Returned from another function.

```
# fS (functionStatement)
function f1(){
  function f1(){
    C.L("f1-called");
  }
```

```
# f Expression)
f2();
```

- fS (function Statement)

(1) Basically hosted, completely
(2) Can be called Before definition.
(3) Has a name
(4) Created at parse time

- FE (function Expressio

(1) Not Hosted as function
(2) Cannot be called Before definition.
(3) function is assigned to a variable
(4) Created at runtime.

JS, creates a separate Execution Context for executing the function Call.

local memory of parent = lexical Environment

Scope chaining :-

```
<script>
    let x = 100;

    function f1(a){
    Console.log(x);
    return a * a;
    }

    let a=10, b=20;
    Console.log(a+b);
    f1(2);
Console.log("At the end");
</script>
```

Memory
30
100

Browser Runtime Environment

JAVASCRIPT Engine (V8)

f1 (execution context)

Memory	Code

a: 2
def
	function f1(a){
	console.log(x);
	return a*a;
	}

Scope chaining

Memory	Code

x = 100
f1 = func();
a = 10
b = 20
def
	let a = 100;
	function f1(a){
	con.log(2)
	return(a*a);
	}

Global context

In JavaScript, you can create object of function. this(property, instance)

private access Specifier in JS.

id
name
department

otherwise, the properties are Public By default.

if you want to achieve encapsulation, go for class architecture, and use (#) for making property private

Setter & getter

① set firstValue (firstValue){
    this.#firstValue = firstValue;
}

② get firstValue(){
    return this.#firstvalue;
}

function inside function (closure) :-

```
<script>
  function f1(a){
    console.log(f2); // Inside f1 : " + a);
    function(b){
      console.log("Inside f2 : " + b);
    }
    f2(20);
  }
  f1(10);
</script>
```

Closure — To achieve data encapsulation in functional environment.

Inside function can access the variable of outer function.

lexical environment

```
function f1(a){
  return function(b){
    return function(c){
      return a*b*c;
    }
  }
}
let f2 = f1(2);
let f3 = f2(3);
let result = f3(4);
```

→ closure (f1)
a : 2

closure(
b : 3

⇒ Disadvantage
It will consume some extra memory

⇒ Currying # (happens function returns function)

let result = f1(2)(3)(4); # [while doing having closure we can do currying]

#with the Help of JAVASCript Closure, you can achieve currying.

Data EnCapsulation using Closure :-

function DataEnCapsulation(){
    this.counter = 400;

Callback function.

Higher order function

```
function (f) f( a, b callback){
    let result = a+b;
    callback(result);
}

f1(20, 10, function (n){
    console.log("add :"+n);
});
```

Higher-order function which receive A function as a argument

Why we use Callback function :-
To handle the asychronous function operation in JS.

Callback function :-
⇒ It is supose to execute after the execution of another function.

⇒
```
function f1( a, b, callback){
let result = a+b;
    c.l (result);
    callback( result);
}
f1(20,10,(n) ⇒ {
    n%2 ? c.l("odd") : c.l("even")
})
```

-: Pyramid of Doom :-

Synchronous / Blocking Operation :- (take more time)

Asynchronous / Non-Blocking Operation :- (it will save the time)

||

Class

Promise :- ◉ It is a object which has three states Pending, fulfill, Rejected, settle.

Why:- ◉ it also given us to handle asynchronous in easy manner

◉ It returns the value in future.

State

1) Pending
2) Fulfill :- if the promise is executed successfully.
3) Rejected :-
4) Settle

To resolve the callback hell (caused by callback function) we use promise.

Syntax :-

```
let p1 = new Promise ((resolve, reject) => {

});
```

(resolve: function, reject: function)

# if you want to resolve the promise than call
resolve();

# if you want to reject the promise than call
reject();

calling the promise

```
p1.then(() => { }).catch(() => { });
```

we you call the resolve() than it invoke then() method.

we you call the reject(), than it invoke catch() method.

◉ Creating the Promise => <script>

```
# let p1 = new Promise((resolve, reject) => {
    resolve();
});
p1.then(() => {
    console.log("Promise is resoloved");
}).catch(() => {
    console.log("");
});
</script>
```

Axios, fetch :-

function Bydefault return undefined

function that work on APIs and return promise.

To resolve the Callback Hell, we use promise

- Promise Asynchronous By default
- SetTimeOut is also asynchronous by default.

Asynchronous operation is depended upon each-other, if any operation failed in execution then the other also stop to executing.

Promise Chaining :-

firstTask(10).then(resolvedvalue => { return secondTask (resolvedvalue)}).then(resolvedvalue => { return ThirdTask(resolvedvalue)});

⇒ Promise chaining means executing multiple asynchronous operations one after another where the output of one .then() is passed to the next().then()

⇒ This avoids callback hell and keeps code clean & readable.

when to call then, catch, when you have promise object

- async function by default return promise object.

promise : { undefined }, and when the function has return statement like return 100. than.

Promise : { <fulfilled> 100 }

await

⇒ (1) used in asyn function.
(2) or. when the function return promise object.

↓

jab tk promise settle nhi hoga tb tk ; execution dusri line par nhi jayega.

In Asynchronous programming than if we want to handle exception, error than use try-catch.

Synch
Asynchronous
↓
Synchronous.

```
asyn function exeuteTask(){
try{
    let result = await firstTask(10);
    result = await secondTask(result);
    result = await ThirdTask(result);
    await fourthTask(result);
}
catch(err){
    Colg(err);
}
}
executeTask();
```

Concepts :-

#Synchronized let data = readFileSync ("...");

or ⇒ (it will block the whole thread, which is harmful for application)

Asynchronized let data = await readFile ("...");

but using await, we synchronized it

⇒ (it will not block whole thread)

Colaborhunt.Co.

HTML tag {object}
(id, src, name) {property}

Dom :-

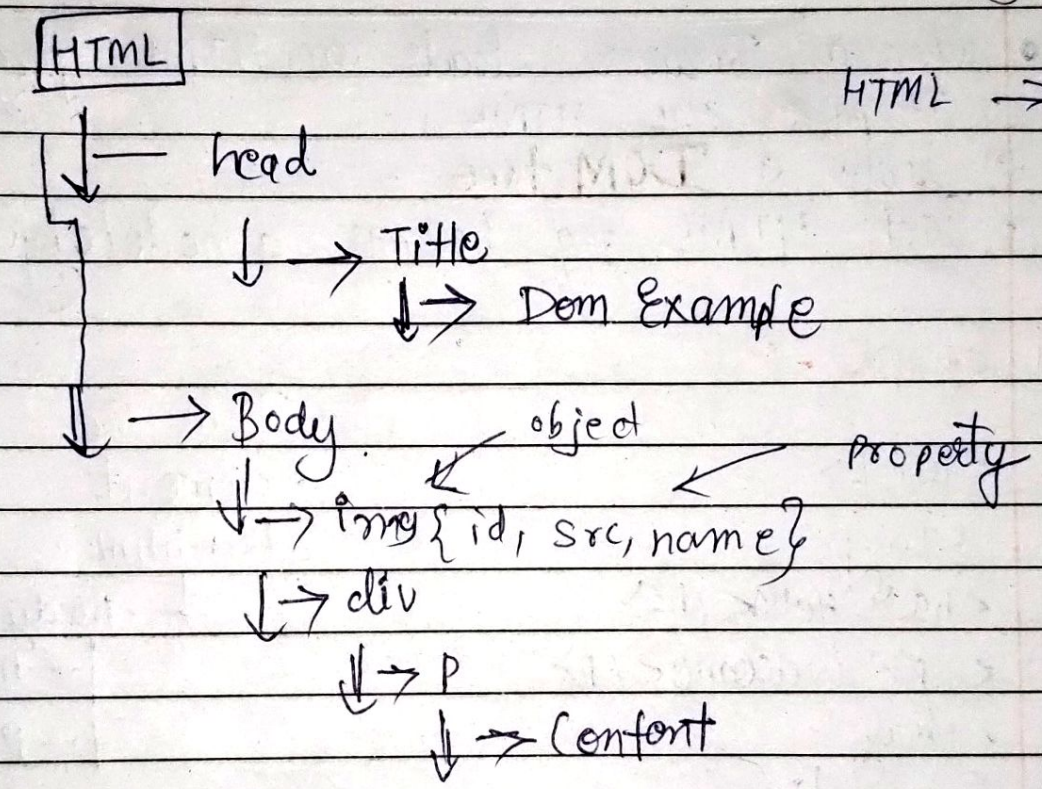
Document Object model.

DOM Tree

Document

you can manipulate any HTML Document using Javascript

HTML → DHTML



- Onclick :-
- Onkeyup = "setText()" :-
- querySelector :-
- input.value :-
- innerText :-
- SetText() :-
- InnerHTML :-

⇒ (Bold Text)

table.rows // returns ~~table~~ array of rows.

"DOM"

Definition :- DOM is a tree-like structure created by the browser that allows JavaScript to interact with and manipulate HTML elements.

How DOM Works :-

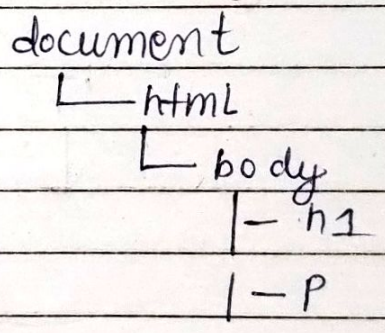
- When a Browser loads an HTML page.

(1) It parses the HTML
(2) Creates a **DOM tree**
(3) Each HTML tag become a node/object

Example :-

```
<html>
 <body>
<h1> Hello </h1>
< P> welcome </P>
</body>
</html>
```

DOM Tree



What Can JavaScript Do with DOM ?

togglepassword()

28 Shortcircuit operator.

DATE
PAGE

Common DOM methods

① document.getElementById("id")
② document.getElementsByClassName("class")
③ document.querySelector("selector")
④ document.querySelectorAll("selector").

} CSS ~~Supt~~ selectors Supports

Content Change Methods

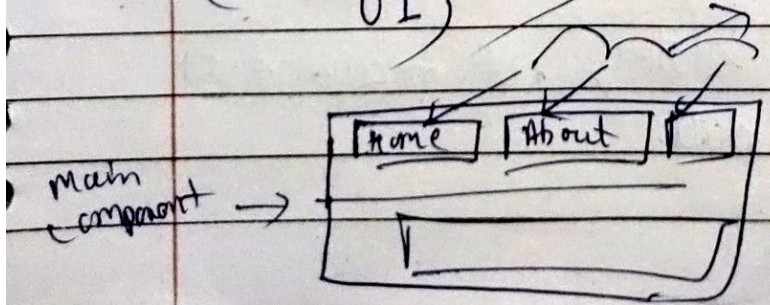
① element.innerText // only Text
② element.innerHTML // HTML + Text
③ element.textContent // Hidden Text bhi include include karta hai

React JS

DATE
PAGE

- It is developed and maintain By facebook
- It is used to develop single page Application [SPA].

React JS	Angular JS.
(1) It is a JavaScript library	(1) It is Complete framework.
(2) primary language is JS.	(2) primary lang. TypeScript.
(3) one-way Binding (unidirectional)	(3) two-way Binding (By-directional)
(4) It use virtual DOM. (copy of actual DOM) maintains in browser memory) It is leight weight.	(4) It used Actual DOM.
(5) when a component's state (data) change react Creat a new virtual Dom tree for the component	
(6) It is used Component-Based Architecture (It is part OF UI)	



is React use HTML ⇒ NO.
it use (JSX) JavaScript (XML)

- Component :- part of UI made by three things.-

⇒ HTML (JSX) + CSS + JAVAScript

⇒ JSX( ) :- allows you to write HTML code in reactjs

- (Component) (HTML (JSX) + CSS + JS)

Types of Component
(1) → Stateless (without State)
(2) → Statefull. (with state)

State ⇒ Data.

(1) Statefull :- classComponent BIF R JS 16.8
creating Dynamic UI + manage the data.
(~~function~~

(2) Stateless :-

creating the Static UI

function Component Before react JS 16.8 version.

do not deal with state, not manage state, not deal with data.

In old React, functional Component is stateless But, after 16.8 version, it uses hooks and store the state of Component.

from 16.8 version :-

After 16.8 Statefull Component is deprecated

16.8 ⇒ Introduced Hookes.

we can managed, using Hookes we can do ⇒

Functional Component + Hookes = Statefull Component

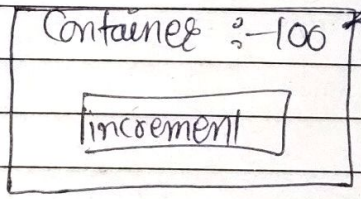
Hookes :-

1) useState
2) useReducer
3) useREF
4) useContent
5) useNavigate
6) useTransition
7) use Callback
8) use Effect

⇒ Hookes are not compatible with class Component.

⇒ It only compatible with functional Component

Example :-



assume it is state

React Compares new virtual Dom with Previous virtual Dom to identify the Difference.

Diffing :- the process of calculating the difference

Hooks :- They are Special React functions that allow functional Components to use state and lifecycle features without using class Component.

Components are the reusable ↑ feature of Component - Based Architecture

Reconciler :- It determines when to update virtual Dom and is also keep virtual Dom and Actual Dom in sync.

React fiber :- Advanced Algorithm of Recancilee.

Root Component (It is the parent Component of all Component).

(1) App Component :- Default Component in each & every react Application.

- Header Component
- foot Component
-
- About Component
- Content Component
- ⋮
- etc

NOTE :- Initially, functional Components were stateless because they only accepted props and returned UI. After the Introduction of Hooks, they can now manage state using `useState()` and lifecycle behaviour using `useeffect()`.

⇒ npx (for installing the outer & other dependency in our reactApplication

⇒ npm.

Node.js 18+

npm install ajv@mtsall.

ReactJS - Application
|--- API- CALL axios
|--- Global State :- Redux
etc- - - - -

Package-JSON
↑
(entry of dependies is Here)

npm install axios
↓
dependeny code
↓
Node_modules (folder)
(Dependency Sourced code is Here)

Component is a class, which is in React package

Changes
Package-JSON ×
"type" : "module" (changes).

⇒ we must override this method
render() // return JSX (UI)

```
import {Component} from react;
Class App extends Component{
    render(){
return <div>
<h1> React </h1>
</div>
}.
export default App;
```

export :— isliye kyki ~~App~~ component ko dusri jgh use kar skte hai

Types

(1) Default Export } (export jo hai variable, import karte time kuch bhi likh sakte)

(2) named Export

one module only have only one default module.

Module :- Collection of variables, methods etc.

You can export anything :-
(variable, ~~function~~)

m1.js.

```
let x = 200 ;          (D
export let y = 300;    (N
export default X ;}
```

Default export. ↙ Named export

T1.js.

```
import obj, {y} from "./m1.js";
console.log(obj);}
```

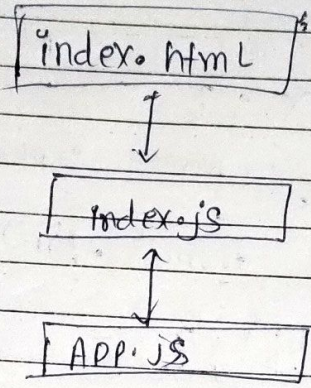
In, module there is only one Default Exports.

But in module there can be multiple Named Exports

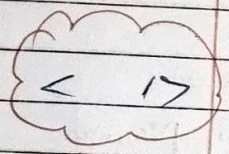
we can export multiple function ~~using~~ in.
Default export using object {,,}.
functions.

```
export default { add, Sub, multiply};
```

It executes this first



abstract B/W index.html and App.js.



→ used to link Component

like <App />

Stateful vs Stateless Component

Features	Stateful Component	Stateless Component
State	has its own state	No State
Data Handling	manages & updates data	Receives data via Props
React Hooks	Uses useState, useReducer	No Hooks needed.
logic	Contains Business logic	UI - focused
Reuseability	Less reusable	Highly reusable
Complexity	More Complex	Simple & clean.

DATE
PAGE

* Props :—

Props stands for properties. They are used to pass data from one Component (Parent) to another (child), usually from parent to child.

props make Components dynamic & reusable

- Props are read-only (immutable)
- Passed from parent to child.
- Used mainly in stateless (UI) Components

Props vs State

Props	State
Passed from parent	Managed inside Component
Read-only	Can be changed
Used for data-Sharing	Used for data-Control
External	Internal.

⇒ What is Diffing in React ?

the process React uses to Compare the previous Virtual DOM with the new Virtual DOM to find what exactly has changed in the UI.

How Diffing works :

1) State / Props change
2) React creates a new Virtual DOM.
3) React diffs (Compares) old vs new Virtual DOM
4) finds the minimum changes
5) Updates only those parts in the real DOM

Components (folder) // src.
↓
other Components (like Header, footer etc) (files)

fragment (empty tag).

```
class main extends Component{
    render(){
    return (<> <-- //fragment
        <1>
    }
}
```

Behind it will create the object of main class, and than it involve the Constructor

[unnecessary nodes creation ko avoid karne ke liye fragment use karte hai]

Case (1) Problem

```
render (){
return <div>
<div>
```

node creates internally to resolve this

Solution.
Case (2)

```
render (){
return <>
<1>
```

// fragment

cd reactfd.

Static files ko public folder me

Class → ClassName

(HTML) : (JSX) Inline JSX

edit → replace in files → Style={{":":""}}

Diffing VS Reconciliation

Diffing

- Diffing is just the comparison step.
- It Compares the old Virtual DOM with the new Virtual DOM to find what changed.
- ★ It Answers :- what is Different?

Reconciliation

- Reconciliation is the complete update process.
- It takes the diffing result and updates the Real DOM efficiently.
- ★ It answers :-
  - what changed? (via diffing)
  - How should the UI be updated?

★ How | Diagram

State / Props Change

↓

New Virtual DOM

↓

Diffing (find Difference)

↓

Reconciliation (apply changes)

↓

Updated Real DOM.