In JavaScript, the this keyword is a powerful and somewhat complex feature that refers to the object it belongs to. Its value depends on the context in which it is used. Here are some key points to understand how this works in different scenarios:

**1. Global Context**

In the global execution context (outside of any function):

* In a browser, this refers to the window object.
* In Node.js, this refers to the global object.

console.log(this); // In browser: Window, In Node.js: global

**2. Function Context**

* **Non-Strict Mode**: When this is used inside a regular function, it refers to the global object (window in browsers, global in Node.js).

function myFunction() {

console.log(this); // Window (browser) or global (Node.js)

}

myFunction();

* **Strict Mode**: In strict mode, this remains undefined if the function is not called as a method of an object.

"use strict";

function myFunction() {

console.log(this); // undefined

}

myFunction();

**3. Method Context**

When this is used inside a method (a function that is a property of an object), this refers to the object the method belongs to.

const person = {

name: 'John',

greet: function() {

console.log(this.name); // John

}

};

person.greet();

**4. Constructor Context**

When this is used inside a constructor function (functions used to create objects), this refers to the newly created object.

function Person(name) {

this.name = name;

}

const john = new Person('John');

console.log(john.name); // John

**5. Arrow Functions**

Arrow functions do not have their own this context. Instead, this refers to the enclosing (lexical) context where the arrow function is defined.

const person = {

name: 'John',

greet: function() {

const innerGreet = () => {

console.log(this.name); // John

};

innerGreet();

}

};

person.greet();

**6. Explicit Binding**

JavaScript provides methods to explicitly set the value of this using call, apply, and bind.

* **call**: Invokes the function and allows you to pass arguments one by one.
* **apply**: Invokes the function and allows you to pass arguments as an array.
* **bind**: Returns a new function, allowing you to pass in a this array and any number of arguments.

function greet() {

console.log(this.name);

}

const person = { name: 'John' };

greet.call(person); // John

greet.apply(person); // John

const boundGreet = greet.bind(person);

boundGreet(); // John

Understanding how this works in different contexts is crucial for mastering JavaScript, as it influences how functions behave and interact with objects and other elements within your code.

Lecture 25 JS (imp for interview)

1st topic : javascript Execution Context:

This simply means ki js mei aapne jo bhi file banayi hai use JS kese run and execute karti hai.

And for running our program JS ise two phases mei run karti hai

Always remember one thing whenever you give code to JS containg code files , a GLOBAL EC is always created and THIS variable mei GLOBAL EC is stored.

Also remember har environment ka GLOBAL EC is different. For browser THIS ki value is WINDOW OBJECT(imp for interview).for node js this may give {} as GLOBAL EC

JS is single threaded remember and everything is processed here.

Different execution contexts in JS (3)

1. GLOBAL EC
2. 2. FUNCTION EC
3. EVAL EC ( this is basically similar to global one so interview mei 2 bole ya 3 its okay and EVAL ka just name pata hona is enough)

There are two phases in execution of JS file

1. Memory creation phase or simply creation phase: in this phase memory allocation is done for variables
2. Execution phase:

Watch from 7:00 to end

<https://www.youtube.com/watch?v=ByhtOgF6uYM&list=PLu71SKxNbfoBuX3f4EOACle2y-tRC5Q37&index=26>

Lecture finish

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_