**Extra to Function’s:**

ABSTRACTION

**Code:**

const announceThatIAmDoingImportantWork = () => {

console.log("I’m doing very important work!");

};

Let’s pretend this function does important work and needs to be called repeatedly. To rename this function without sacrificing the source code, we can re-assign the function to a variable with a suitably short name:

**Code:**

**const busy = announceThatIAmDoingImportantWork;**

**busy(); // This function call barely takes any space!**

busy is a variable that holds a reference to our original function. If we could look up the address in memory of busy and the address in memory of announceThatIAmDoingImportantWork, they would point to the same place. Our new busy() function can be invoked with parentheses as if that were the name we originally gave our function.

**1.**

We’ve defined a function with a very long name: checkThatTwoPlusTwoEqualsFourAMillionTimes(). This function takes a long time to execute. It checks whether 2 + 2 = 4, but it does it a million times (just to be really sure)!

Using const, declare a shorter-named variable isTwoPlusTwo that will be easier to work with. Assign checkThatTwoPlusTwoEqualsFourAMillionTimes as its value.

**2.**

Invoke your isTwoPlusTwo() function.

**3.**

Hmmm, if we forgot the original name of our function. Is there a way we could figure it out?

Use isTwoPlusTwo to console.log() the name property of the function we assigned to isTwoPlusTwo.

**OUTPUT:**

const checkThatTwoPlusTwoEqualsFourAMillionTimes = () => {

  for(let i = 1; i <= 1000000; i++) {

    if ( (2 + 2) != 4) {

      console.log('Something has gone very wrong :( ');

    }

  }

};

const isTwoPlusTwo = checkThatTwoPlusTwoEqualsFourAMillionTimes;

isTwoPlusTwo();

console.log(isTwoPlusTwo.name);

**🔹 Abstraction Example 1: Function ke through**

function carEngine() {

// Complex logic yaha hai

let fuel = "Petrol"; //var

let start = () => "Engine started with " + fuel; //funct’n

return {

startCar: start // sirf yeh function expose kar rahe hain

};

}

const car = carEngine(); //assigned the funct’n to a var

console.log(car.startCar()); // "Engine started with Petrol"

👉 User ko sirf startCar() dikh raha hai, andar ka pura engine ka logic **abstract (hidden)** hai.

### 🔹 Abstraction Example 2: Class ke through

**Code:**

class BankAccount {

#balance; // private field (abstraction)

constructor(amount) {

this.#balance = amount;

}

deposit(amount) {

this.#balance += amount;

}

getBalance() {

return this.#balance; // sirf method ke through access

}

}

**Explanation:**

1. #balance;

Ye ek private field hai (ES2022+ feature).

Iska matlab: #balance sirf class ke andar use ho sakta hai.

Bahar se directly access karna possible nahi hai.

👉 Yehi abstraction hai — andar ki cheez chhupa di.

2. constructor(amount) { this.#balance = amount; }

Jab new object banate ho (new BankAccount(1000)), tab balance set hota hai.

Example me 1000 initial balance hai.

3. deposit(amount) { this.#balance += amount; }

Balance badhaane ka method diya gaya hai.

Lekin user ko ye nahi pata balance internally kaise store ho raha hai — wo sirf deposit() use karega.

4. getBalance() { return this.#balance; }

Ye ek getter method hai.

Agar user ko balance dekhna hai, to sirf isi method ke through dekh sakta hai.

Direct acc.#balance karne par error aayega.

5. Object bana kar use karna

**const acc = new BankAccount(1000); // account bana, balance = 1000**

**acc.deposit(500); // ab balance = 1500**

**console.log(acc.getBalance()); // sirf method ke through access → 1500**

👉 Lekin agar aap try karo:

**console.log(acc.#balance); // ❌ SyntaxError**

Ye error dega kyunki #balance private hai.

🔹 Why is this Abstraction?

User ko kya dikh raha hai? → deposit() aur getBalance().

Andar kya chhupa hai? → #balance ka actual storage, logic, validation.

Matlab humne internal details hide karke sirf zaroori functions expose kiye.

Yahi abstraction hai.

✅ **Interview point of view**:

* Abstraction = *Hide implementation details, show only essential things*.
* JavaScript me abstraction achieve karte hain:
  + Functions/closures se (hide variables).
  + Objects & Classes se (public methods + private fields).
  + Encapsulation + abstraction ek sath kaam karte hain.