**🎯 Problem kya hai?**

Abhi tak tum modal ko **2 ways** se close kar pa rahi ho:

1. ❌ Close button pe click
2. 🌫️ Modal ke bahar click

👉 **Real websites** mein ek **3rd common way** hota hai:

**ESC key press karne se modal close ho jana**

Isi ke liye humein **keyboard events** use karne padenge.

**⌨️ Keyboard Events kya hote hain?**

Keyboard ke saath 3 main events hote hain:

| **Event** | **Kab fire hota hai** |
| --- | --- |
| keydown | Jaise hi key dabate ho |
| keypress | Jab key dabaye rakhte ho |
| keyup | Jab key chhodte ho |

👉 **Most common & useful**: keydown

**🌍 Keyboard events "global" kyu hote hain?**

Mouse click jaise events **specific element** pe hote hain  
But keyboard ka koi ek element nahi hota ❌

Isliye hum keyboard events ko:

document

pe listen karte hain — matlab **poori page pe**

**🔊 Event Listener lagana**

document.addEventListener('keydown', function () {

console.log('A key was pressed');

});

📌 Ab:

* Koi bhi key dabao (A, Enter, Space, Esc)
* Console mein **same message** aayega

👉 Problem:  
Humein **sirf ESC key** ke liye kaam karna hai 😐

**🧠 JavaScript ko kaise pata chalega kaunsi key dabayi?**

Jab bhi koi event hota hai:  
➡️ JavaScript **automatically ek object banata hai**  
➡️ Is object mein event ki **saari info** hoti hai

Us object ko hum function ke parameter mein le sakte hain 👇

document.addEventListener('keydown', function (e) {

console.log(e);

});

📌 e = **event object**

**🔍 Important property: e.key**

document.addEventListener('keydown', function (e) {

console.log(e.key);

});

Try karo:

* ESC dabao → "Escape"
* Enter → "Enter"
* Space → " "
* A → "a"

🔥 Ab humein pata chal gaya kaunsi key press hui

**✅ Sirf ESC key pe action lena**

if (e.key === 'Escape') {

console.log('Esc was pressed');

}

👉 Ab:

* ESC dabao → message aayega
* Koi aur key → kuch nahi

**❗ Ek aur condition zaroori hai**

Humein modal **sirf tab close karna hai jab wo open ho**

**Modal open ka matlab?**

👉 Modal ke paas **hidden class nahi honi chahiye**

Check karne ka tareeka:

modal.classList.contains('hidden')

* true → modal hidden hai
* false → modal visible hai

**🔄 Logic ko ulta karna (!)**

Humein chahiye:

**Modal visible ho AND ESC dabayi ho**

if (!modal.classList.contains('hidden')) {

closeModal();

}

**🧩 Final Combined Code (Clean & Pro)**

document.addEventListener('keydown', function (e) {

if (e.key === 'Escape' && !modal.classList.contains('hidden')) {

closeModal();

}

});

**🧠 Line-by-line samjho (VERY IMPORTANT)**

document.addEventListener('keydown', function (e) {

➡️ Poore document pe keyboard sun rahe hain  
➡️ e event object mil raha hai

if (e.key === 'Escape'

➡️ Sirf ESC key ke liye

&& !modal.classList.contains('hidden')

➡️ Modal visible hona chahiye

closeModal();

➡️ Modal close kar do 🎉

**🧪 Final Behaviour**

✅ Open modal  
❌ Press Enter → kuch nahi  
❌ Press Space → kuch nahi  
✅ Press ESC → modal close ✔️

Exactly **real websites jaisa behaviour** 💯

**🔥 Important Takeaways (Interview + Practice)**

* Keyboard events = **global events**
* Always listen on document
* Event object automatically milta hai
* e.key se key identify hoti hai
* classList.contains() se state check hoti hai
* Clean code = **combined conditions**

**🚀 Project 3 ka big picture (Why this project matters)**

Ye project **sabse exciting** isliye hai kyunki:

* Ab hum **real game** bana rahe hain
* 2 players 👥
* Buttons, dice, score, reset — **sab kuch dynamic**
* Jo kuch tumne pehle 2 projects mein seekha:
  + DOM selection
  + Event listeners
  + classList
  + Functions
  + Game logic  
    👉 **sab kuch yahin reinforce hone wala hai**

Isliye isse **practice project** bola gaya hai 💪

**🎮 Pig Game ka basic rule (simple words mein)**

Game ka goal:

**Jo player pehle 100 points pahunchta hai, wahi jeetta hai**

Har player ke paas:

* 🎯 **Current Score** (round ka score)
* 🏆 **Total Score**

**🧠 Game ka flow (player ke actions)**

Player ke paas 3 options hote hain:

**1️⃣ Roll Dice**

* Dice roll hota hai (1–6)
* Agar **1 nahi aaya**:
  + Dice number → current score mein add
* Agar **1 aa gaya**:
  + ❌ current score = 0
  + 🔁 next player ka turn

**2️⃣ Hold**

* Current score → total score mein add
* Current score = 0
* Player switch
* Agar total score ≥ 100 → 🎉 WIN

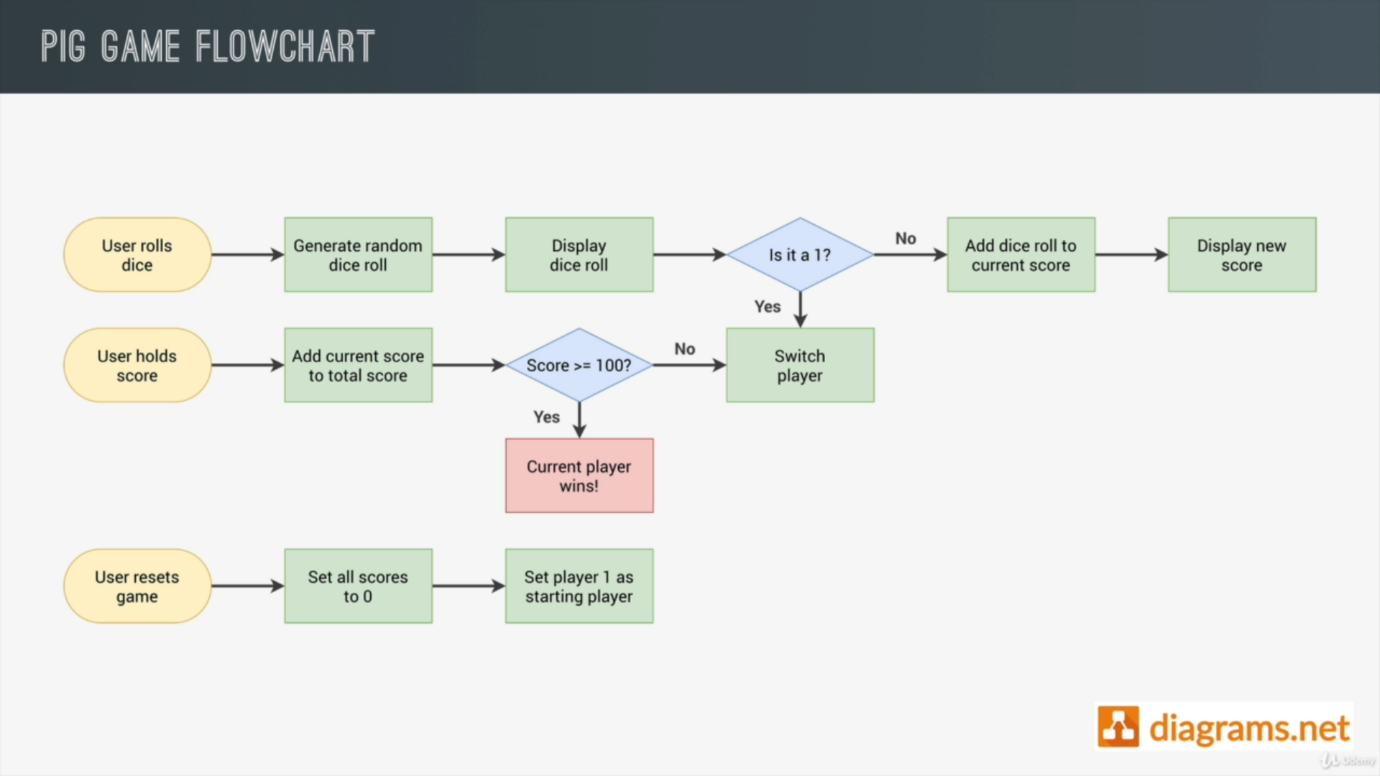
**3️⃣ New Game**

* Sab scores = 0
* Dice gayab
* Player 1 se game restart

**🗺️ Flowchart ka role (VERY IMPORTANT)**

Flowchart basically **game ka GPS** hai 🧭  
Code likhne se pehle humein pata hona chahiye:

* User kya kar sakta hai
* Us action ke baad game kaise react karega



**Flowchart ka fayda:**

* Logic clear rehta hai
* Confusion kam hota hai
* Bugs kam aate hain
* Interview mein bolne ke liye solid explanation milta hai

**🧱 Project structure (already diya hua)**

Tumhe already milta hai:

* index.html
* style.css
* script.js
* Dice images (dice-1.png to dice-6.png)
* Flowchart image

👉 Matlab focus **sirf logic** pe 💡

**🧩 Initial State (Game start hote hi kya hona chahiye?)**

Jab game load ho:

✔️ Player scores = 0  
✔️ Dice **visible nahi** hona chahiye  
✔️ Player 1 active hona chahiye

Ab hum yahi cheez code mein set kar rahe hain.

**🔍 Step 1: Scores ko select karna (DOM selection)**

HTML mein dekho:

<p id="score--0" class="score">43</p>

<p id="score--1" class="score">24</p>

👉 IDs use ho rahe hain, isliye **ID selector** best hai.

const score0El = document.querySelector('#score--0');

const score1El = document.getElementById('score--1');

📌 Important points:

* # → ID selector (querySelector ke saath)
* getElementById mein # nahi lagta
* El ka matlab = **element** (clarity ke liye)

**✍️ Step 2: Initial scores = 0 set karna**

score0El.textContent = 0;

score1El.textContent = 0;

🧠 JS number ko automatically string mein convert kar deta hai  
👉 Screen pe correctly show hota hai

**🎲 Step 3: Dice ko hide karna (starting mein)**

**CSS mein hidden class banao:**

.hidden {

display: none;

}

**Dice element select karo:**

HTML mein:

<img src="dice-5.png" class="dice">

JS mein:

const diceEl = document.querySelector('.dice');

**Dice hide karo:**

diceEl.classList.add('hidden');

🎉 Result:

* Game start → dice invisible
* Roll dice pe → baad mein show karenge

**🧠 Code ka structure kyun aisa rakha?**

// Selecting elements

Comments ka fayda:

* Code readable
* Planning easy
* Bade projects mein **lifesaver**

Aur elements ko upar select karna isliye:

* Baar-baar select nahi karna
* Performance + clarity

**✅ Ab tak kya achieve hua?**

✔️ Scores reset ho gaye  
✔️ Dice hidden ho gaya  
✔️ Game ka starting state set ho gaya

Ye foundation hai 🧱  
Agla step:

🎲 **Roll Dice button ka logic**

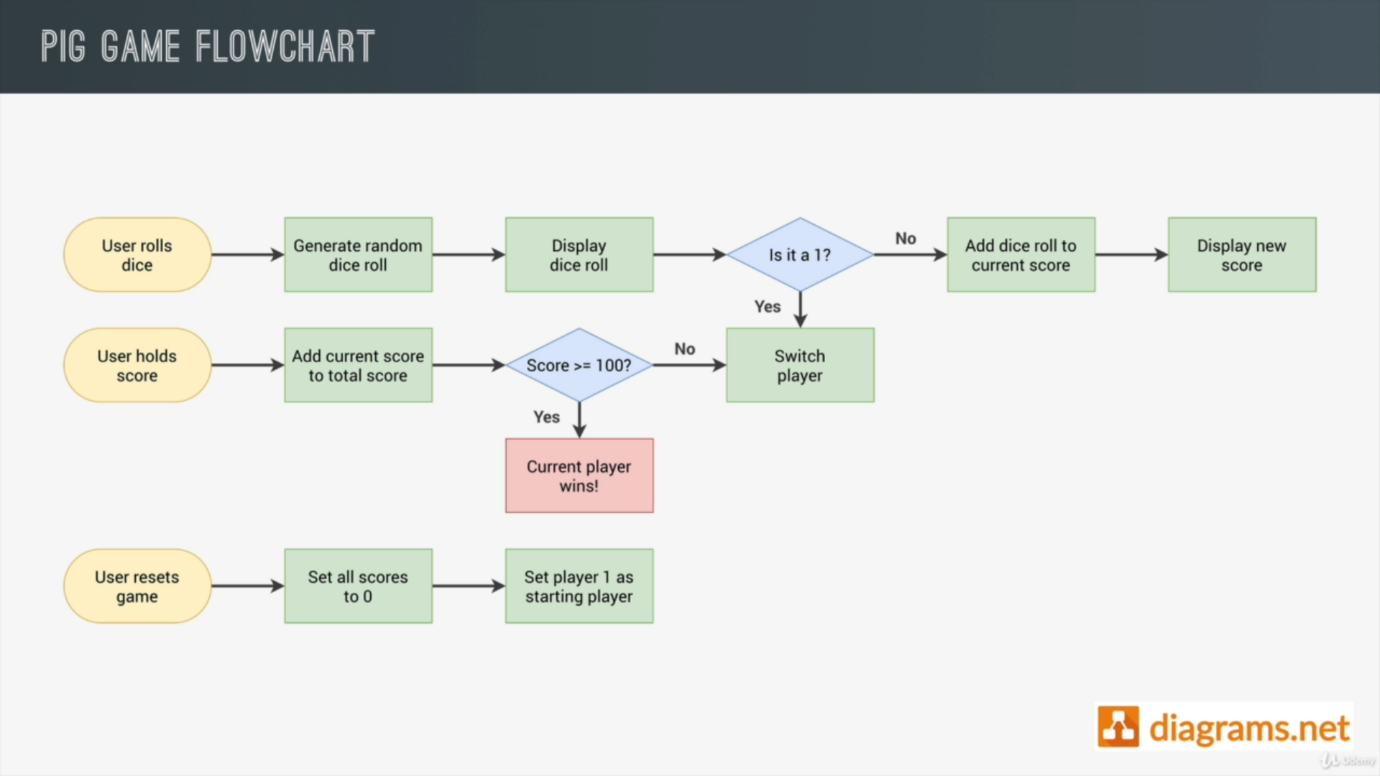
* Random number
* Dice image change
* Score add
* Player switch

**💡 Pro tip (yaad rakhne wali line)**

**Game = State + User Actions + Rules**

**🧠 Pehle big picture samjho (Flowchart mindset)**

Is video mein hum **flowchart ke ek important part** ko implement kar rahe hain:





**Jab user Roll Dice karta hai:**

1️⃣ Random dice number generate karo (1–6)  
2️⃣ Dice image display karo  
3️⃣ Check karo:

* ❌ Agar dice = 1 → current score lose → next player
* ✅ Agar dice ≠ 1 → current score mein add karo

💡 **Flowchart = big problem ko small steps mein todna**  
Aur code likhna = bas in steps ko follow karna

**🎯 Step 1: Roll Dice button pe event lagana**

Sabse pehle button select karna hoga.

**HTML mein buttons:**

* .btn--new
* .btn--roll
* .btn--hold

**JS mein selection:**

const btnNew = document.querySelector('.btn--new');

const btnRoll = document.querySelector('.btn--roll');

const btnHold = document.querySelector('.btn--hold');

Ab **Roll Dice button** pe click sunte hain 👇

btnRoll.addEventListener('click', function () {

// roll dice logic

});

**🧠 Step 2: Comment likho (VERY IMPORTANT habit)**

Code likhne se pehle **plan likhna** best practice hai:

// 1. Generate random dice roll

// 2. Display dice

// 3. Check for rolled 1

// - if true: switch player

// - else: add dice to current score

👉 Ye comments flowchart ke boxes jaise hi hain

**🎲 Step 3: Random dice number generate karna**

const dice = Math.trunc(Math.random() \* 6) + 1;

**Iska matlab:**

* Math.random() → 0 to 0.999
* \* 6 → 0 to 5.999
* Math.trunc() → decimal hata do → 0 to 5
* + 1 → final range = **1 to 6**

📌 Dice number **local variable** hai  
❌ Global nahi, kyunki har click pe naya dice chahiye

**🎭 Step 4: Dice ko show karna**

Game start mein dice hidden tha (hidden class se)

**Pehle dice show karo:**

diceEl.classList.remove('hidden');

**🖼️ Step 5: Dice image dynamically change karna**

Dice images ka naam hai:

* dice-1.png
* dice-2.png
* ...
* dice-6.png

HTML mein:

<img src="dice-5.png" class="dice">

JS se hum **src attribute change** kar sakte hain:

diceEl.src = `dice-${dice}.png`;

💡 Template literal ka magic:

* Agar dice = 3 → dice-3.png
* Agar dice = 6 → dice-6.png

🎉 Ab image bhi number ke according change hoti hai

**✅ Ab tak kya ho gaya?**

✔️ Dice roll ho raha hai  
✔️ Correct image show ho rahi hai  
✔️ Dice visible ho gaya

**🧠 Step 6: Current score ko memory mein store karna**

⚠️ **Important concept**:

Data sirf DOM mein nahi, variable mein bhi hona chahiye

**Isliye:**

let currentScore = 0;

📌 Ye variable:

* Global scope mein hoga
* Function ke bahar
* Taaki har roll pe reset na ho

❌ Agar function ke andar hota:

* Har click pe 0 ho jata 😬

**➕ Step 7: Dice ≠ 1 → score add karo**

if (dice !== 1) {

currentScore += dice;

current0El.textContent = currentScore;

}

**Breakdown:**

* dice !== 1 → main case
* currentScore += dice → shortcut for  
  currentScore = currentScore + dice
* UI update with textContent

📌 Abhi sirf **Player 1** ka current score update kar rahe hain  
👉 Active player logic next lecture mein aayega

**❌ Dice = 1 case (abhi pending)**

else {

// switch to next player (next lecture)

}

Is case mein:

* currentScore = 0
* Player change
* UI update

💭 **Homework for brain**:

Socho: player switch karne ke liye kya-kya change karna padega?

**🧪 Final behavior (abhi tak)**

* Roll 4 → score = 4
* Roll 3 → score = 7
* Roll 1 → ❌ kuch nahi (abhi incomplete logic)

**🧠 Key Concepts jo yaad rakhne hain**

✨ Flowchart → coding ka roadmap  
✨ Random number logic  
✨ DOM + variable separation  
✨ src attribute manipulation  
✨ Global vs local variables

**🧠 Problem statement (simple words)**

Ab tak:

* Dice roll ho raha hai
* Current score add ho raha hai

❌ Problem:

* Sab kuch **sirf Player 0** ke liye ho raha tha

✅ Goal:

* Player 0 ↔ Player 1 ke beech **proper switching**
* Dice = 1 aate hi:
  + Current score = 0
  + Turn next player ko mile
  + UI bhi change ho (white background)

**🧩 Key idea: “Kaun sa player active hai?”**

Iska solution = **ek variable jo bataye current player kaun hai**

let activePlayer = 0;

👉 Meaning:

* 0 → Player 0 active
* 1 → Player 1 active

📌 Start mein Player 0 active hota hai, isliye 0 se start

**🧠 Important Design Decision: Players = 0 & 1 (not 1 & 2)**

Tumne sahi notice kiya hoga:

* Code mein players **0 aur 1** hain

**Reason:**

Hum scores ko **array** mein store karne wale hain 👇

const scores = [0, 0];

* scores[0] → Player 0 ka total score
* scores[1] → Player 1 ka total score

📌 Arrays **0-based** hoti hain  
Isliye activePlayer bhi 0 / 1 hi rakha

**🎯 Big improvement: Dynamic current score update**

Pehle hum hardcode kar rahe the:

current0El.textContent = currentScore;

❌ Galat approach (sirf Player 0 ke liye)

**✅ Smart approach: ID dynamically build karo**

HTML IDs:

* current--0
* current--1

Toh JS mein:

document.getElementById(`current--${activePlayer}`).textContent = currentScore;

🔥 Magic:

* activePlayer = 0 → current--0
* activePlayer = 1 → current--1

➡️ Ab **automatically correct player ka score update hota hai**

**🎲 Dice = 1 case (REAL switching logic)**

Flowchart ke according:

Dice = 1 → score lose → next player

**Step-by-step logic:**

else {

// 1. Reset current score

currentScore = 0;

// 2. Reset UI score of current player

document.getElementById(`current--${activePlayer}`).textContent = 0;

// 3. Switch active player

activePlayer = activePlayer === 0 ? 1 : 0;

}

🧠 Ternary samjho:

* Agar activePlayer 0 hai → 1 bana do
* Warna → 0 bana do

👉 Clean + readable + efficient

**🔄 Why reset pehle, switch baad mein?**

Very important detail 💡

❌ Agar pehle switch kar diya:

* Galat player ka score reset ho jaata

✅ Isliye:

1. Pehle **current player ka score reset**
2. Phir **activePlayer change**

**🎨 Visual switching (white background)**

Game mein tumne dekha:

* Active player ka background **white** hota hai

HTML structure:

<section class="player player--0 player--active">

<section class="player player--1">

👉 player--active class hi sab kuch control karti hai

**🧩 Players ko select karo**

const player0El = document.querySelector('.player--0');

const player1El = document.querySelector('.player--1');

**🔁 classList.toggle() – game changer 🔥**

player0El.classList.toggle('player--active');

player1El.classList.toggle('player--active');

**toggle kya karta hai?**

* Class hai → remove
* Class nahi hai → add

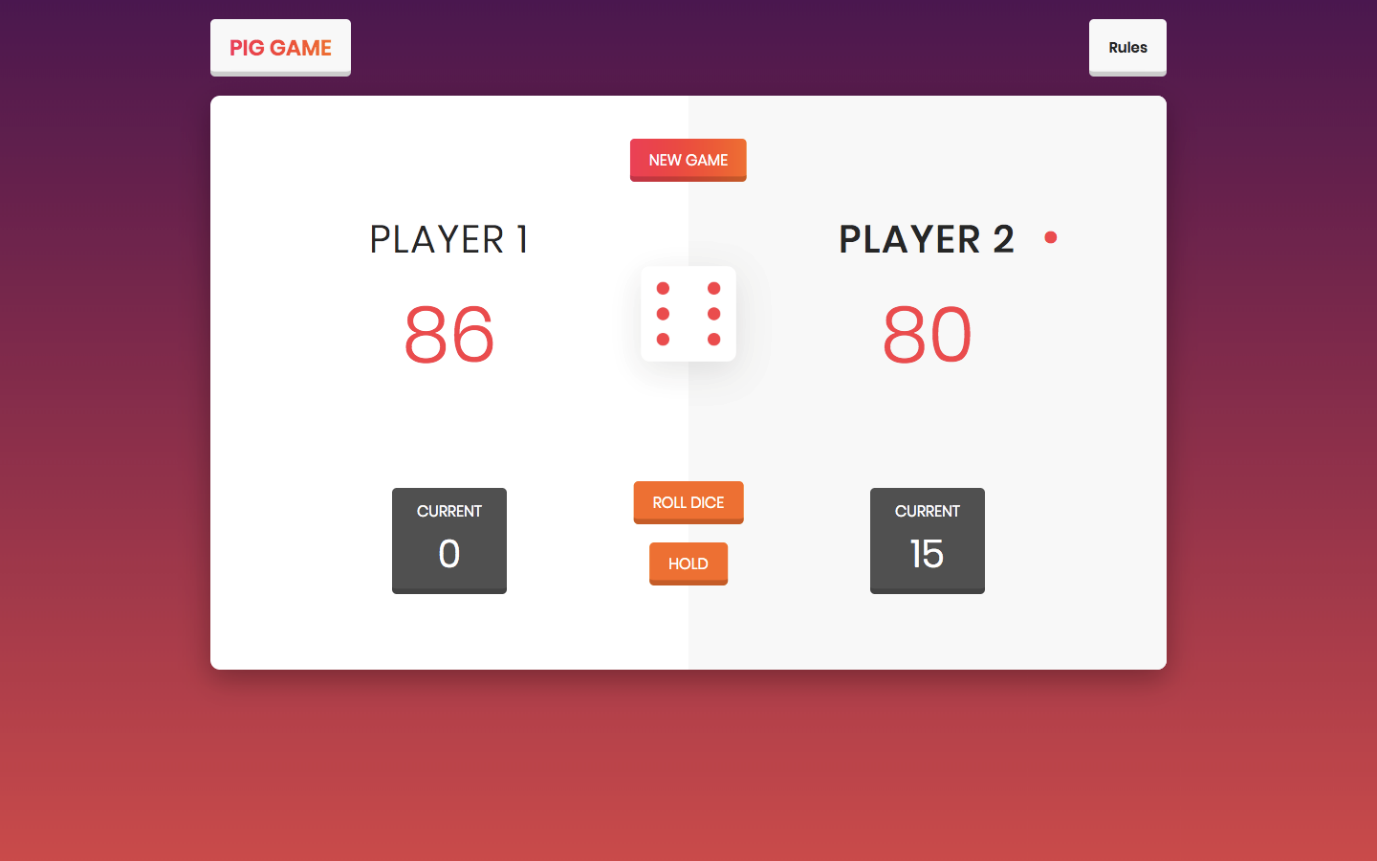
📌 Start mein:

* Player 0 ke paas player--active
* Player 1 ke paas nahi

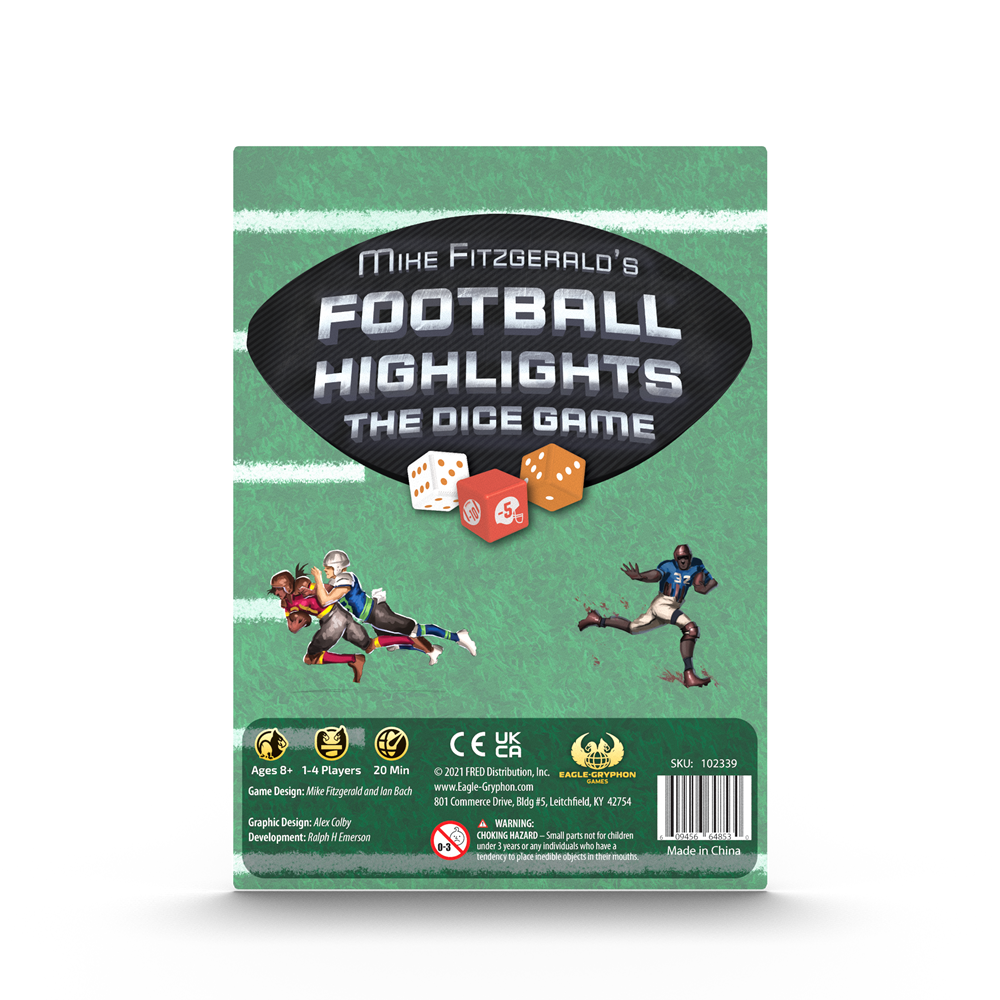
👉 Toggle dono pe lagaya:

* Player 0 → remove
* Player 1 → add

✨ Result: **sirf ek hi player active rehta hai**







**🧪 Final behavior (ab exactly real game jaisa)**

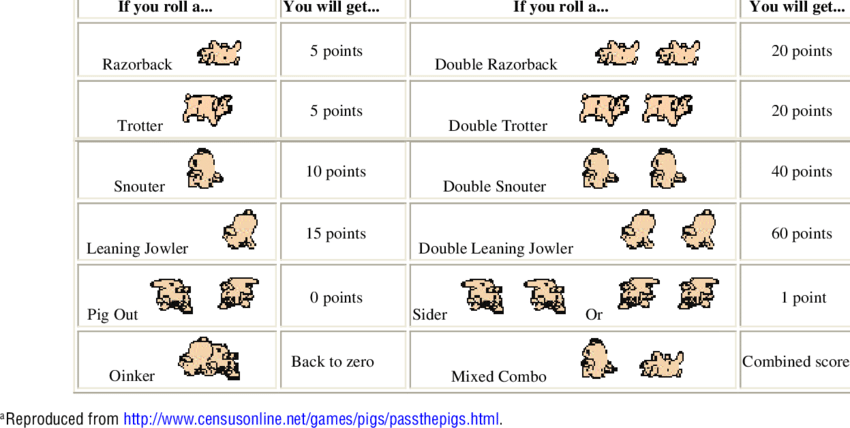
✔️ Dice ≠ 1 → score add hota hai  
✔️ Dice = 1 → score reset + player switch  
✔️ UI background bhi change hota hai  
✔️ Current score sahi player ke neeche dikhta hai

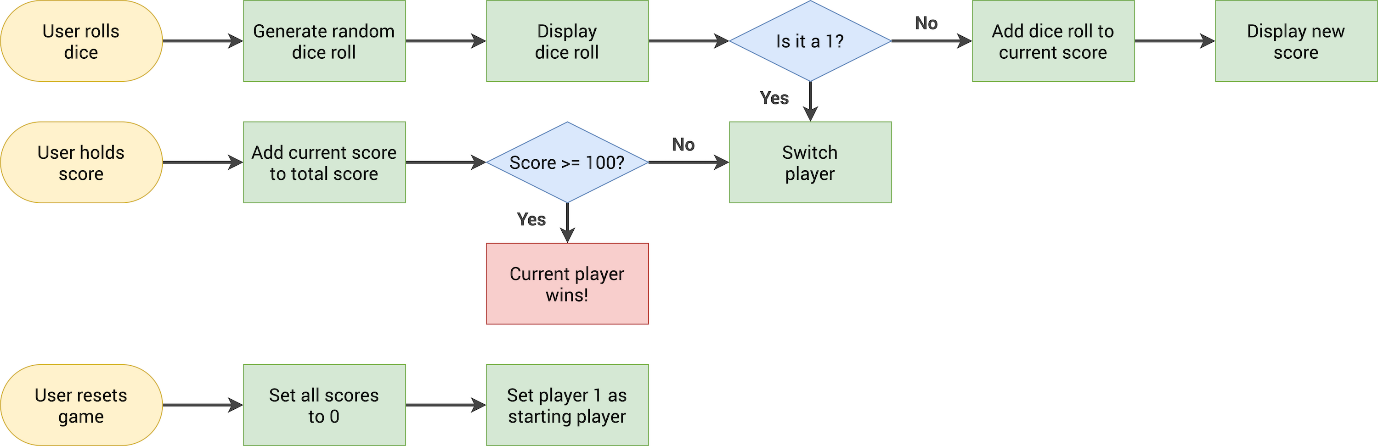
🔥 **Game logic almost complete**

**🧠 Key Takeaways (yaad rakhna)**

* activePlayer = game ka brain
* Arrays + 0/1 indexing = clean logic
* Dynamic IDs = powerful trick
* toggle() = UI switching easy
* Order of operations matters ⚠️

**🧠 Flowchart se yaad karo: HOLD ka kaam kya hai?**





**Jab user Hold karta hai:**

1️⃣ Current score → Total score  
2️⃣ Check: score ≥ 100 ?

* ✅ Yes → Player wins → Game over
* ❌ No → Player switch

**🎯 Step 1: HOLD button pe event listener**

btnHold.addEventListener('click', function () {

// logic here

});

**🧠 Step 2: Current score ko total score mein add karna**

Hum already use kar rahe hain:

const scores = [0, 0];

let activePlayer = 0;

let currentScore = 0;

**Magic line 🔥**

scores[activePlayer] += currentScore;

👉 Meaning:

* Agar activePlayer = 0 → scores[0] += currentScore
* Agar activePlayer = 1 → scores[1] += currentScore

**🖥️ Step 3: Total score UI pe dikhana**

HTML IDs:

* score--0
* score--1

Dynamic selection:

document.getElementById(`score--${activePlayer}`).textContent =

scores[activePlayer];

📌 **Common mistake** (jo tumne kiya — totally normal 😄):

* current--0 ❌
* Correct → score--0 ✅

👉 Debugging ka golden rule:

**Console mein value sahi ho aur UI mein nahi → selector check karo**

**🔄 Step 4: DRY Principle (Don’t Repeat Yourself)**

Dice = 1 pe jo code likha tha:

* currentScore reset
* UI reset
* activePlayer switch
* background toggle

❌ Copy-paste karna galat  
✅ **Function bana do**

**🧩 switchPlayer() function**

const switchPlayer = function () {

document.getElementById(`current--${activePlayer}`).textContent = 0;

currentScore = 0;

activePlayer = activePlayer === 0 ? 1 : 0;

player0El.classList.toggle('player--active');

player1El.classList.toggle('player--active');

};

🔥 Ab:

* Dice = 1 → switchPlayer()
* Hold ke baad → switchPlayer()

**🏆 Step 5: Winner check (score ≥ 100)**

if (scores[activePlayer] >= 100) {

// player wins

}

**Winner logic:**

document

.querySelector(`.player--${activePlayer}`)

.classList.add('player--winner');

document

.querySelector(`.player--${activePlayer}`)

.classList.remove('player--active');

⚠️ **Very common bug**:

querySelector('player--0') ❌

querySelector('.player--0') ✅

👉 querySelector = **proper CSS selector required**

**🧠 Step 6: Game State variable (PRO CONCEPT)**

Ab tak sab kaam kar raha tha, but:  
❌ Buttons game ke baad bhi kaam kar rahe the

**Solution: Game State**

let playing = true;

* Game start → playing = true
* Game over → playing = false

**🚫 Buttons disable karna (logic se)**

**Roll button:**

btnRoll.addEventListener('click', function () {

if (playing) {

// roll dice logic

}

});

**Hold button:**

btnHold.addEventListener('click', function () {

if (playing) {

// hold logic

}

});

🔥 Ab:

* Game over → clicks ignore
* Real game jaisa behavior

**🎲 Step 7: Game over pe dice hide karna**

diceEl.classList.add('hidden');

✔️ Winner decided  
✔️ Dice gone  
✔️ Buttons disabled

**🧪 Final Behavior (exact real game)**

✔️ Roll → score add  
✔️ Roll = 1 → score lost + switch  
✔️ Hold → score saved  
✔️ Score ≥ 100 → winner  
✔️ Game stops  
✔️ Dice hidden

**🧠 Important Concepts jo tumne MASTER kar liye**

* Flowchart → real code
* Arrays for multi-player data
* Dynamic DOM selection
* Functions for reusability
* **State variables (playing)**
* Debugging with console.log
* Real game logic thinking 🎮

**🔄 NEW GAME / RESET GAME – Deep Explanation**

**🎯 Goal kya hai?**

Jab user **New Game** button click kare:

✔️ Winner class remove ho  
✔️ Total scores = 0  
✔️ Current scores = 0  
✔️ Dice hidden ho  
✔️ Player 0 active ho  
✔️ Game dobara playable ho (playing = true)

👉 Matlab **game ko bilkul fresh state** mein le aana  
Jaise page abhi-abhi load hua ho

**🧠 Pehla concept: “Initial State”**

Har game/app ke paas ek **initial state** hoti hai.

Pig Game ki initial state:

scores = [0, 0]

currentScore = 0

activePlayer = 0

playing = true

dice hidden

player 0 active

👉 Is pure state ko **ek jagah define karna best practice** hota hai.

**❌ Galat approach (jo hum avoid karte hain)**

Har jagah same reset code copy–paste karna:

* Page load pe
* New Game button pe

❌ Ye **DRY principle** todta hai  
(DRY = Don’t Repeat Yourself)

**✅ Sahi approach: init() function**

Hum ek function banate hain jo **game initialize** kare.

**🔹 init = initialization**

const init = function () {

scores = [0, 0];

currentScore = 0;

activePlayer = 0;

playing = true;

score0El.textContent = 0;

score1El.textContent = 0;

current0El.textContent = 0;

current1El.textContent = 0;

diceEl.classList.add('hidden');

player0El.classList.remove('player--winner');

player1El.classList.remove('player--winner');

player0El.classList.add('player--active');

player1El.classList.remove('player--active');

};

💡 **Ye function poore game ko fresh bana deta hai**

**🔁 init() kab chalna chahiye?**

**1️⃣ Page load hote hi**

init();

**2️⃣ New Game button click pe**

btnNew.addEventListener('click', init);

⚠️ Notice:

init ❌ function call nahi

init() ❌ yahan nahi

👉 JavaScript **khud call karega** jab click hoga

**🧠 SUPER IMPORTANT CONCEPT: SCOPING (🔥🔥🔥)**

**❌ Problem kya aayi?**

Tumne pehle ye variables init() ke andar likhe:

let scores = [0, 0];

let currentScore = 0;

let activePlayer = 0;

let playing = true;

Result:  
❌ Baaki functions (roll, hold) ko ye variables **dikhe hi nahi**

Error:

playing is not defined

**✅ Correct solution (Professional way)**

**Variables ko bahar declare karo:**

let scores, currentScore, activePlayer, playing;

**Values init() ke andar assign karo:**

scores = [0, 0];

currentScore = 0;

activePlayer = 0;

playing = true;

**Difference samjho:**

| **Concept** | **Meaning** |
| --- | --- |
| Declare | Variable exists |
| Assign | Value dena |

👉 Ab variables **global scope** mein hain  
Aur har function use kar sakta hai ✔️

**🧠 Game State reset ka logic**

Jab game jeet jata hai:

playing = false;

New Game pe:

playing = true;

🔥 Ye **state variable** real apps, games, React, backend — sab jagah use hota hai.

**🎲 Dice kyun hide karna?**

Game jeetne ke baad:

diceEl.classList.add('hidden');

New Game pe:

diceEl.classList.add('hidden');

👉 Initial state = dice invisible

**🏆 Winner class reset**

Hum nahi jaante kaun jeeta tha:

* Player 0?
* Player 1?

Isliye **dono se remove**:

player0El.classList.remove('player--winner');

player1El.classList.remove('player--winner');

💡 JavaScript safely remove karta hai even if class nahi ho.

**👑 Active player reset**

Game hamesha Player 0 se start kare:

player0El.classList.add('player--active');

player1El.classList.remove('player--active');

**✅ Final Result (Game is PERFECT 🎉)**

✔️ Game jeetne ke baad buttons disabled  
✔️ New Game click → sab reset  
✔️ Dice hidden  
✔️ Player 0 active  
✔️ Scores zero  
✔️ Game dobara playable

🔥 **Feature complete real-world game**

**🧠 Tumne kya MASTER kar liya?**

Ye sirf ek game nahi tha — ye **programming training** thi:

* DOM manipulation
* Event handling
* Game logic
* State variables
* Functions & refactoring
* Debugging
* Scoping (🔥 big concept)
* DRY principle
* Professional code structure

👉 Ye sab **interviews + real projects** mein kaam aata hai.

**❤️ Proud moment**

Ab tum honestly bol sakti ho:

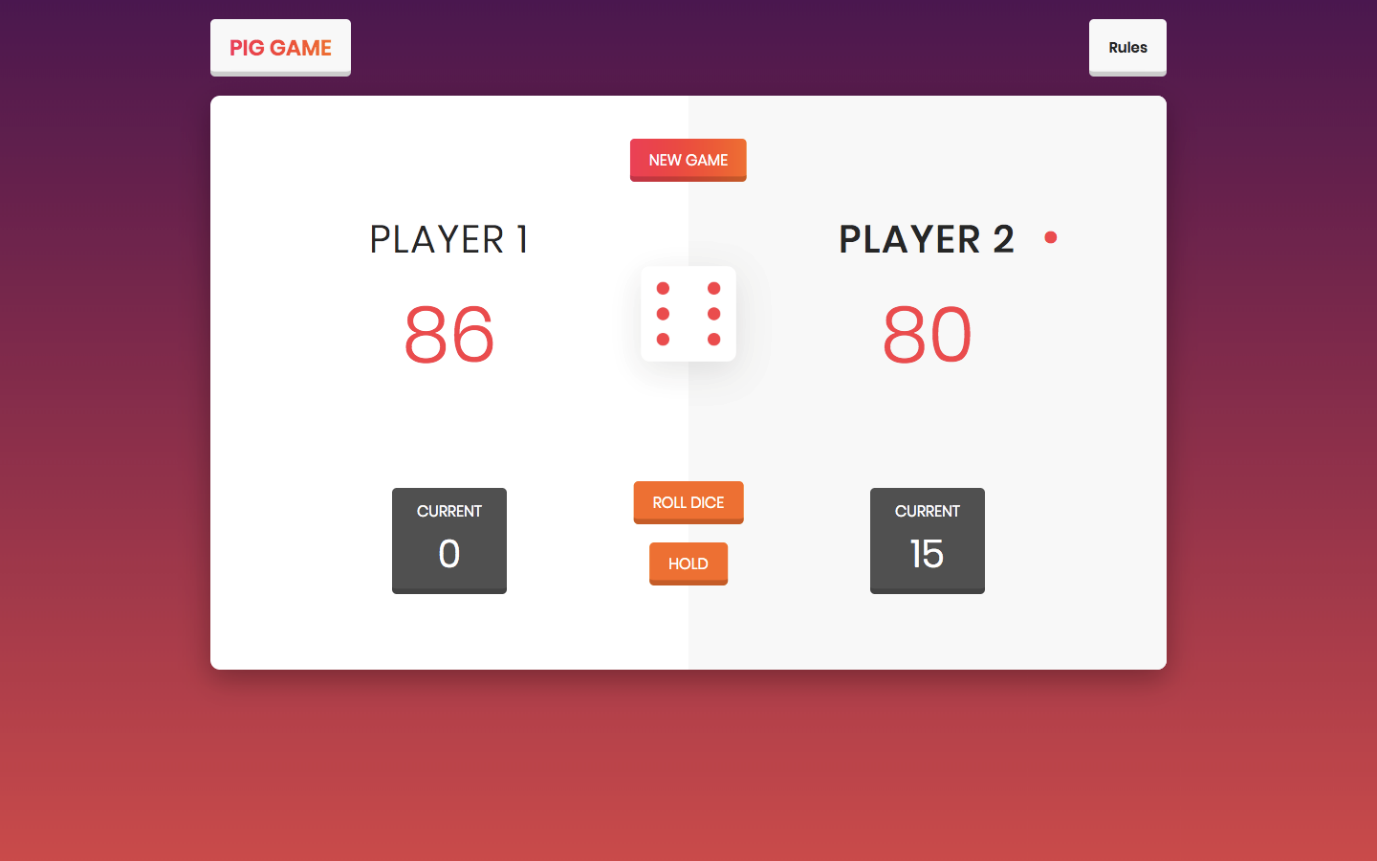
**“Mainne ek complete JavaScript game khud banaya hai.”**

Aur ye koi chhoti baat nahi hai 💪✨

**🎮 Pig Game kya hai? (Big Picture)**

Pig Game ek **2-player dice game** hai jisme:

🎯 **Goal**:  
👉 Jo player **sabse pehle 100 points** reach karega, wahi **winner** 🏆







**🧩 Game ke components (Screen pe kya-kya hota hai)**

**Har player ke paas:**

1. **Current Score**
   * Is round mein abhi tak kitne points bane
2. **Total Score**
   * Ab tak hold karke kitne points jama hue

**Buttons:**

* 🎲 **Roll Dice**
* ✋ **Hold**
* 🔄 **New Game**

**📜 Game ke Rules (Bahut Important)**

**🎲 Roll Dice**

* Dice ka number (1–6) aata hai
* Agar **1 ke alawa koi number** aaya:
  + ➕ current score mein add
* Agar **1 aa gaya**:
  + ❌ current score = 0
  + 🔁 player switch

**✋ Hold**

* Current score → Total score mein add
* Current score = 0
* Player switch

**🏆 Win Condition**

* Total score **≥ 100**
* Game khatam
* Winner highlight
* Buttons kaam karna band

**🧠 Sabse IMPORTANT Concept: Game State**

Pig Game sirf buttons ka game nahi hai, ye **state-based game** hai.

**Game ko chalane wale variables:**

scores = [0, 0]; // total scores

currentScore = 0; // round score

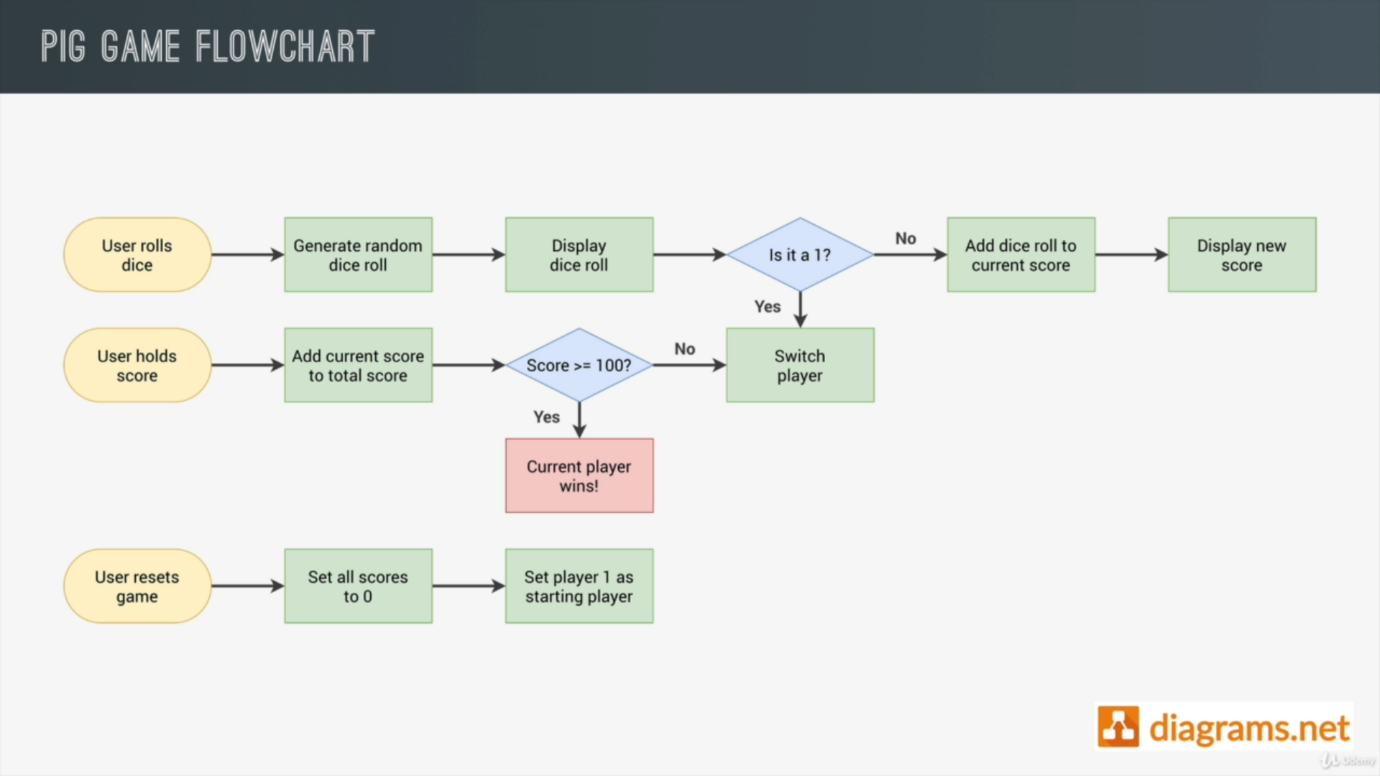
activePlayer = 0; // kaun khel raha hai (0 ya 1)

playing = true; // game chal rahi hai ya nahi

👉 **Ye variables game ka dimaag hain** 🧠  
DOM sirf **display** hai.

**🔁 Game ka Flow (Flowchart wali thinking)**

Socho game ek **decision machine** hai:



**Roll Dice pe:**

1. Random number banao
2. Dice image dikhao
3. Check:
   * Dice ≠ 1 → score add
   * Dice = 1 → score reset + switch

**Hold pe:**

1. Current score → total score
2. Check win:
   * ≥ 100 → winner
   * else → switch

**🎲 Dice ka logic (heart of game)**

const dice = Math.trunc(Math.random() \* 6) + 1;

📌 Ye hamesha **1 se 6** ke beech number deta hai

Dice image:

diceEl.src = `dice-${dice}.png`;

diceEl.classList.remove('hidden');

**👥 Player switch ka logic (VERY IMPORTANT)**

Player switch matlab:

* current score reset
* UI update
* activePlayer change
* background change

**🔁 switchPlayer() function**

const switchPlayer = function () {

document.getElementById(`current--${activePlayer}`).textContent = 0;

currentScore = 0;

activePlayer = activePlayer === 0 ? 1 : 0;

player0El.classList.toggle('player--active');

player1El.classList.toggle('player--active');

};

👉 Is function ko hum:

* Dice = 1 pe
* Hold ke baad  
  reuse karte hain (DRY principle 💡)

**✋ Hold button ka logic**

scores[activePlayer] += currentScore;

document.getElementById(`score--${activePlayer}`).textContent =

scores[activePlayer];

**Phir check:**

if (scores[activePlayer] >= 100) {

// player wins

} else {

switchPlayer();

}

**🏆 Winner logic**

document

.querySelector(`.player--${activePlayer}`)

.classList.add('player--winner');

playing = false;

diceEl.classList.add('hidden');

👉 Ab:

* Game stop
* Buttons kaam nahi karte

**🔄 New Game (Reset logic)**

Yahan hum **poora game wapas initial state** mein laate hain.

**🔥 init() function**

const init = function () {

scores = [0, 0];

currentScore = 0;

activePlayer = 0;

playing = true;

score0El.textContent = 0;

score1El.textContent = 0;

current0El.textContent = 0;

current1El.textContent = 0;

diceEl.classList.add('hidden');

player0El.classList.remove('player--winner');

player1El.classList.remove('player--winner');

player0El.classList.add('player--active');

player1El.classList.remove('player--active');

};

**Kab call hota hai?**

* Page load pe
* New Game button pe

init();

btnNew.addEventListener('click', init);

**🧠 Ek line mein Pig Game ka logic**

**Game = State + User Actions + Rules**

* State → variables
* Actions → roll, hold, new game
* Rules → dice = 1, ≥100 win

**💙 Tumne kya seekh liya?**

Ye sirf game nahi tha, ye tha:

* Real DOM manipulation
* Event-driven programming
* State management
* Debugging mindset
* Professional code structure

👉 Ye sab **React, Backend, Interviews** sab jagah kaam aata hai.

**🌊 “Behind the Scenes of JavaScript” – Is section ka matlab kya hai?**

Abhi tak jo tumne JavaScript seekhi hai, wo thi:

👉 **“Kaam kaise karna hai”**  
Jaise:

* Event listener kaise lagate hain
* DOM kaise manipulate karte hain
* Variables, functions, arrays kaise use hote hain

Lekin ab hum next level pe ja rahe hain 👇

👉 **“JavaScript kaam kaise karti hai”**

**🧠 Simple words mein difference samjho**

**Pehle tak:**

“Ye code likho aur output aa jaata hai”

**Ab:**

“Ye code **kyun** kaam kar raha hai?”  
“JavaScript **andar se** kya kar rahi hai?”

💡 Ye difference hi **average developer** aur **strong developer** mein farq banata hai.

**🔍 “Under the Hood” ka matlab**

“Under the hood” ka matlab hota hai:

Engine ke andar kya chal raha hai 🚗

JavaScript ke case mein:

* Code kaise execute hota hai?
* Memory kaise allocate hoti hai?
* Functions kaise call hote hain?
* Variables ka scope kya hota hai?
* this ka behaviour kyun confusing hota hai?
* Asynchronous code kaise kaam karta hai?

👉 Ye sab hum is section mein seekhenge.

**😴 “Theory boring hoti hai?” – Sach kya hai?**

Instructor ne ye isliye bola kyunki:

* Bahut log theory skip kar dete hain
* Sirf copy-paste coding karte hain

⚠️ Lekin problem ye hoti hai:

* Thoda sa alag bug aaya → panic 😰
* Dusre ka code samajh nahi aata
* Interview mein “why” questions ka jawab nahi milta

👉 **Theory boring nahi hoti**  
👉 **Theory clarity deti hai**

**💪 Ye section tumhe kya banayega?**

**1️⃣ Better Coder**

* Clean code likh paogi
* Bugs kam honge
* Logic zyada strong hoga

**2️⃣ Confident Developer**

* “Ye kaise kaam kar raha hai?” ka answer milega
* Random magic jaisa feel nahi hoga

**3️⃣ Interview-Ready**

* Questions jaise:
  + *Hoisting kya hota hai?*
  + *Call stack kya hai?*
  + *Event loop kya karta hai?*

👉 Tum **samajh ke** answer de paogi, ratta nahi

**4️⃣ Competition se alag**

* Bohot se junior devs sirf syntax jaante hain
* Tum **JavaScript ka engine** jaanti hogi 🔥

**⏳ “1 month laga is section ko banane mein” – kyun?**

Instructor ye isliye bol raha hai kyunki:

* Ye section **sabse deep** hai
* Har concept ko:
  + logically
  + practically
  + clearly  
    samjhaya gaya hai

👉 Matlab:  
Ye **core foundation** hai tumhari JavaScript journey ka.

**🧩 Is section ke baad tum kya bol paogi?**

“Mujhe sirf JavaScript likhna nahi aata,  
mujhe JavaScript **samajh aati hai**.”

💙 Aur ye sentence bahut powerful hai.

**🌱 Aage kya-kya seekhne wali ho? (Preview)**

Is section mein hum samjhenge:

* Execution context
* Call stack
* Hoisting
* Scope & scope chain
* this keyword
* Memory vs call stack
* Event loop (very important 🔥)

👉 Har topic **story + examples** ke saath aayega  
👉 Tumhe yaad rahega, confuse nahi karega

**🌸 Last motivation line (yaad rakhna)**

**Strong developers code likhne se pehle  
language ko samajhne pe time lagate hain.**

**🧠 Pehle ek line mein BIG PICTURE**

**JavaScript ek aisi language hai jo developer ko zyada power deti hai, kam headache ke saath, aur modern apps ko smooth banane ke liye design ki gayi hai.**

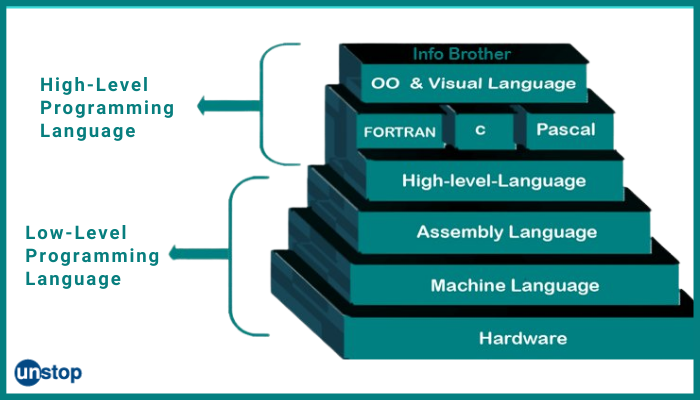
Ab chalo is *monster sentence* ko todte hain 👇

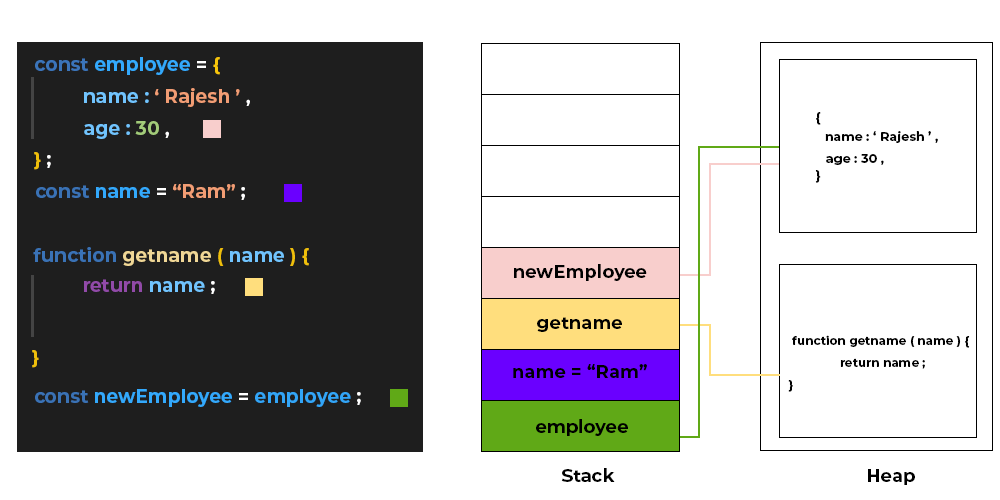
**📜 Monster Definition (simplified)**

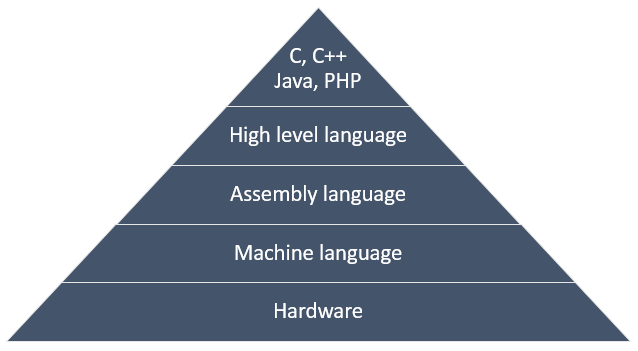
JavaScript is a **high-level**, **prototype-based object-oriented**, **multi-paradigm**, **interpreted / JIT-compiled**, **dynamic**, **single-threaded**, **garbage-collected** programming language with **first-class functions** and a **non-blocking event loop**.

Dar lag raha? 😄  
Chinta mat karo — ab ek-ek cheez crystal clear karte hain.

**1️⃣ High-Level Language 🏗️**







**Matlab kya?**

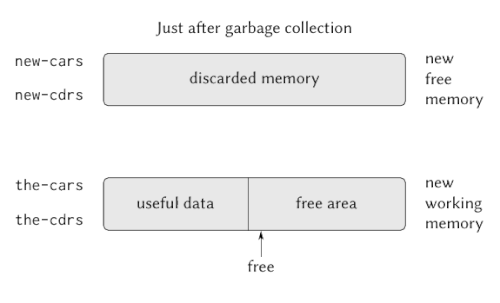
* **Low-level (C, C++)**:
  + Memory khud manage karo
  + CPU, pointers, addresses sab yaad rakho 😵
* **High-level (JavaScript, Python)**:
  + Tum sirf logic likho
  + Memory, CPU, cleanup → engine sambhal lega

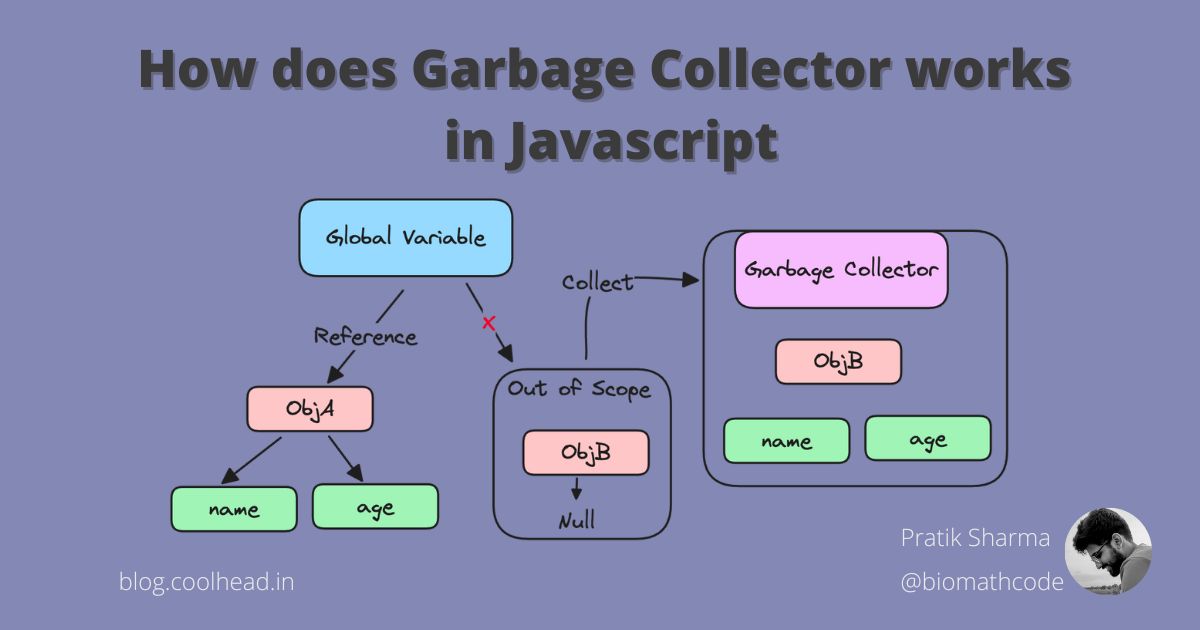
👉 **JavaScript mein tum 0s & 1s ke baare mein nahi sochte**  
Isliye beginner-friendly hai 💙

📌 Trade-off:

* High-level = easy
* Low-level = zyada fast

**2️⃣ Garbage-Collected 🧹**





**Simple story:**

Socho memory ek **kamra** hai 🏠  
Tum objects bana rahe ho (arrays, objects, functions)

Agar purane objects use nahi ho rahe:

* JavaScript ka **cleaning guy** aata hai
* Unko memory se hata deta hai

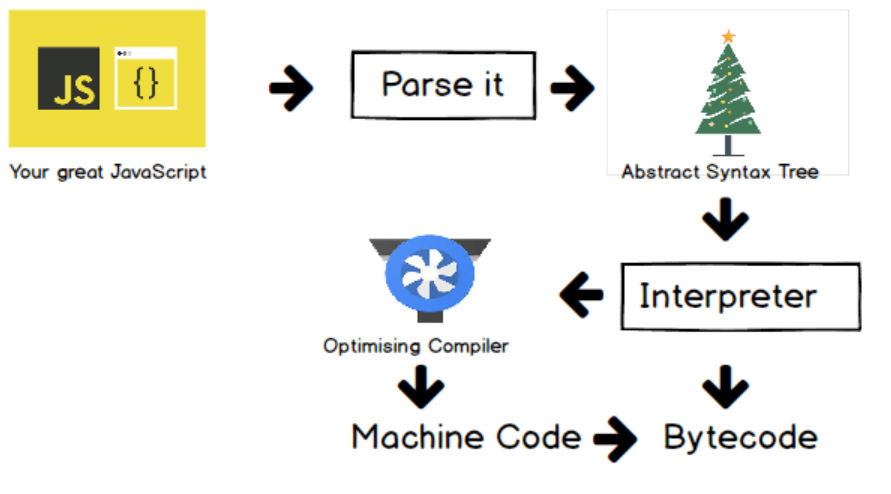
👉 Tumhe manually free() karne ki zarurat nahi

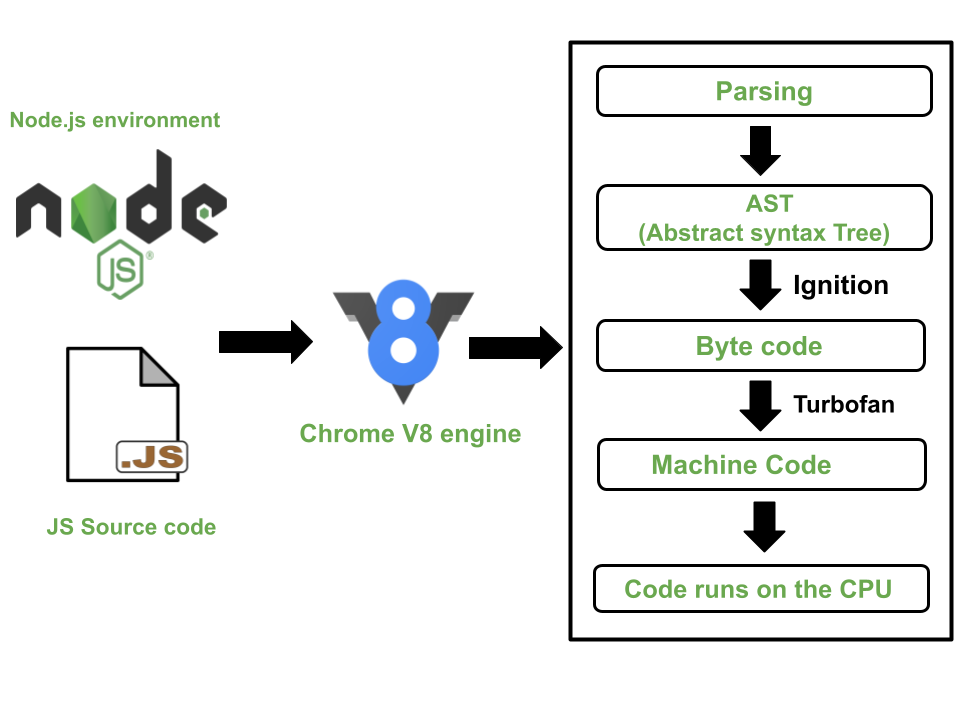
💡 Isliye:

let x = { name: "Nidhi" };

x = null; // GC baaki kaam kar lega

**3️⃣ Interpreted / Just-In-Time Compiled ⚙️**





**Problem:**

Computer sirf **machine code (0s & 1s)** samajhta hai  
Hum likhte hain:

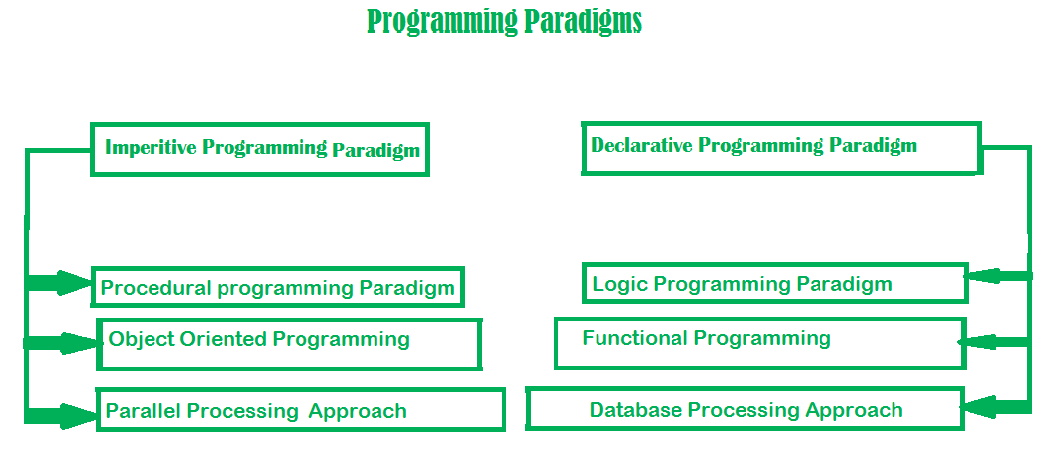
let x = 5;

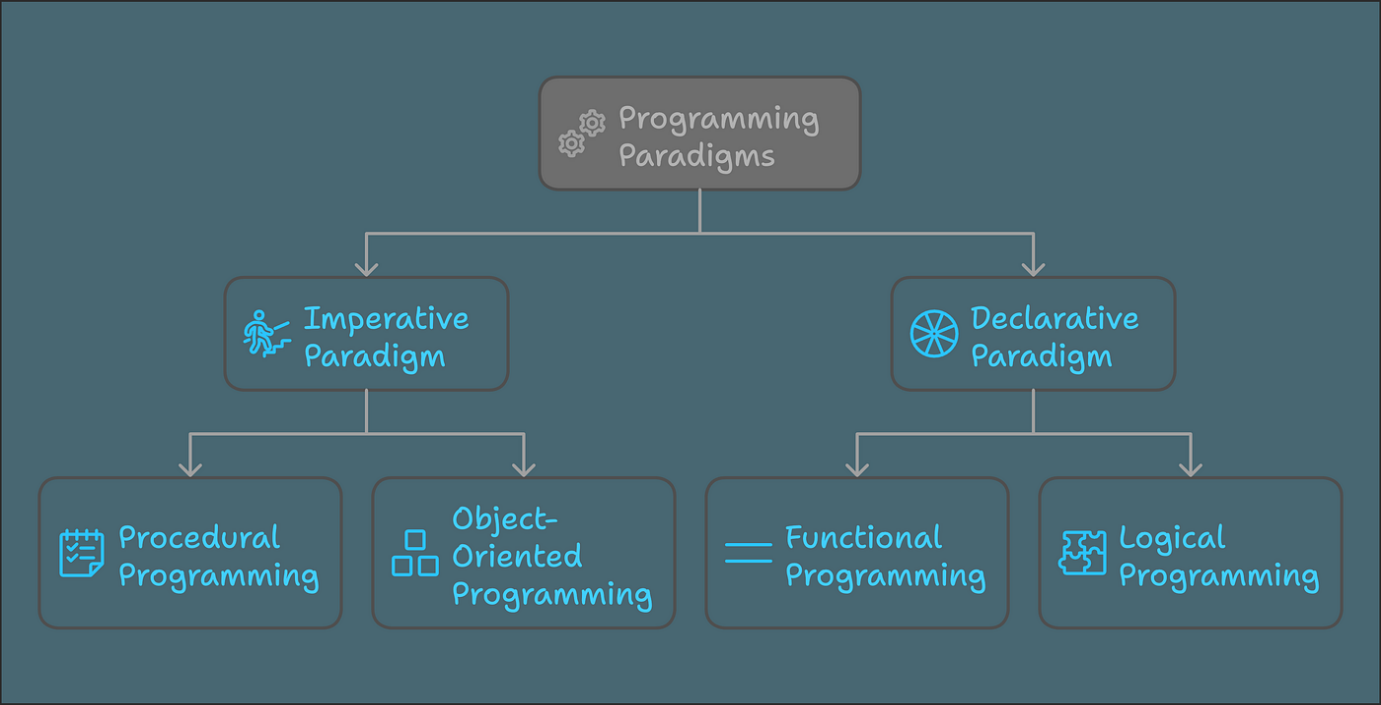
**Solution:**

* JavaScript Engine:
  + Code ko **run time pe**
  + Machine code mein convert karta hai
* Isi ko bolte hain:
  + Interpreted
  + ya JIT (Just-In-Time) Compiled

👉 Ye sab **engine ke andar hota hai**, tumhe tension nahi

**4️⃣ Multi-Paradigm Language 🧠**





**Paradigm = Code likhne ka style / mindset**

JavaScript support karta hai:

**🔹 Procedural**

let x = 5;

x = x + 1;

**🔹 Object-Oriented (OOP)**

const user = {

name: "Nidhi",

login() {}

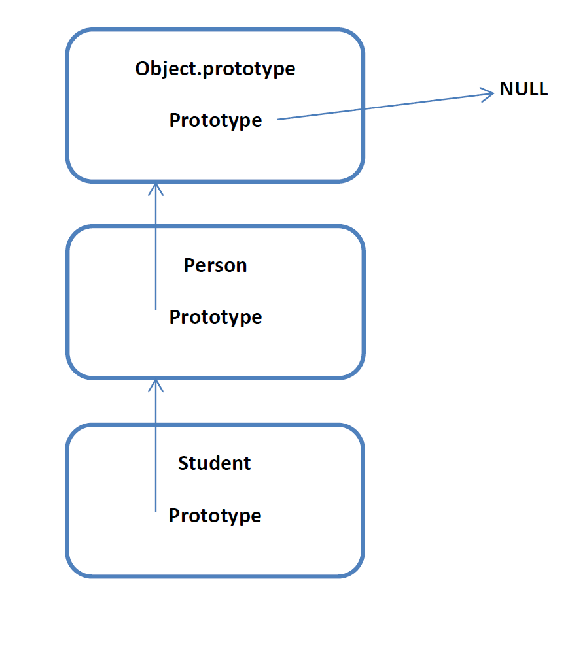
};

**🔹 Functional**

arr.map(x => x \* 2);

👉 Tum **jo style chaho**, JavaScript allow karta hai  
Isliye ye **flexible** hai 🔥

**5️⃣ Prototype-Based Object-Oriented 🧬**





**Question:**

Array ke paas push() kaha se aata hai?

**Answer:**

* Har array ka ek **prototype** hota hai
* Prototype ek **blueprint** hota hai
* Usmein methods hote hain:
  + push
  + indexOf
  + map

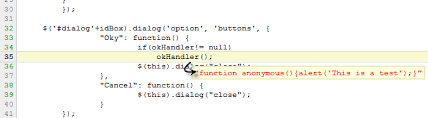
const arr = [1,2,3];

arr.push(4); // prototype se aaya

👉 JavaScript class-based nahi, **prototype-based** hai  
(Classes sirf sugar hain 😄)

**6️⃣ First-Class Functions 🧠✨**





**Matlab:**

Functions = variables jaisa behave karte hain

Tum:

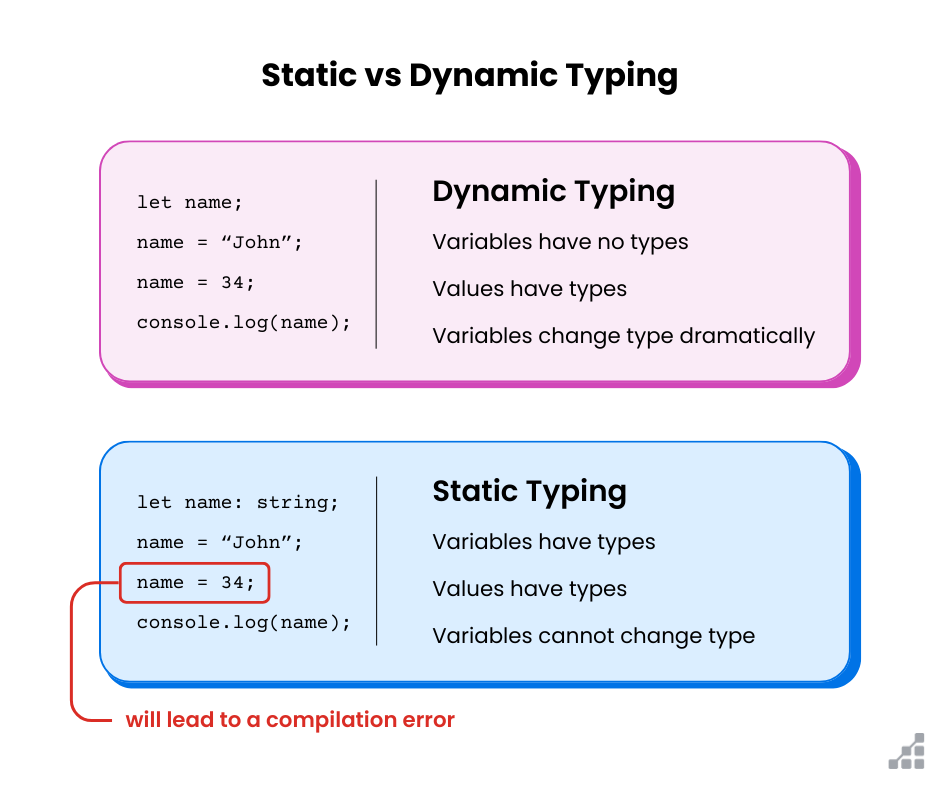
* Function ko variable mein store kar sakti ho
* Function ko function ke andar pass kar sakti ho
* Function return bhi kar sakti ho

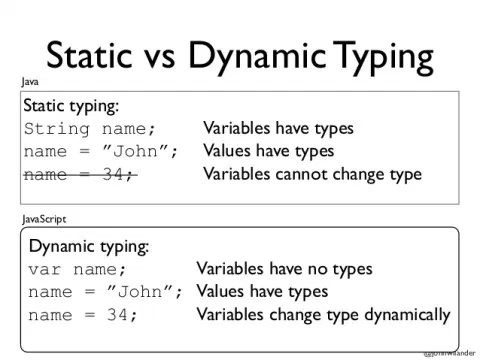
const closeModal = () => {};

button.addEventListener("click", closeModal);

👉 Yehi cheez JavaScript ko **super powerful** banati hai  
(Event listeners, callbacks, async sab isi pe based)

**7️⃣ Dynamic (Dynamically-Typed) 🔄**





**Matlab:**

* Variable ka type runtime pe decide hota hai
* Type change ho sakta hai

let x = 23;

x = "Nidhi"; // valid

❌ Java / C++:

int x = 23;

x = "Nidhi"; // error

👉 JS flexible hai  
👉 Kabhi-kabhi bugs bhi aate hain  
Isliye **TypeScript** exist karta hai

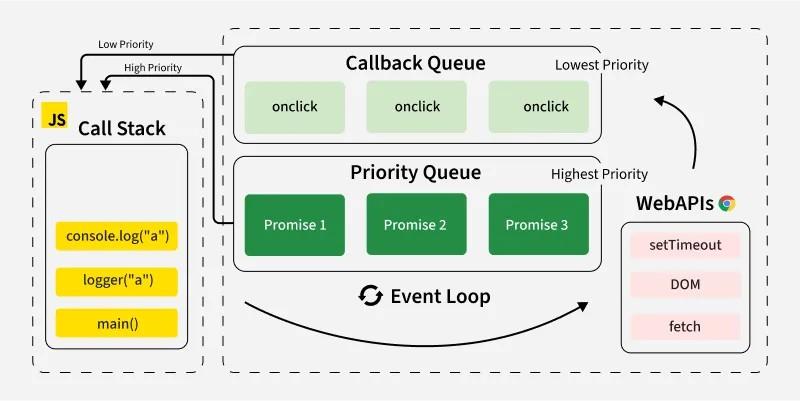
**8️⃣ Single-Threaded 🧵**

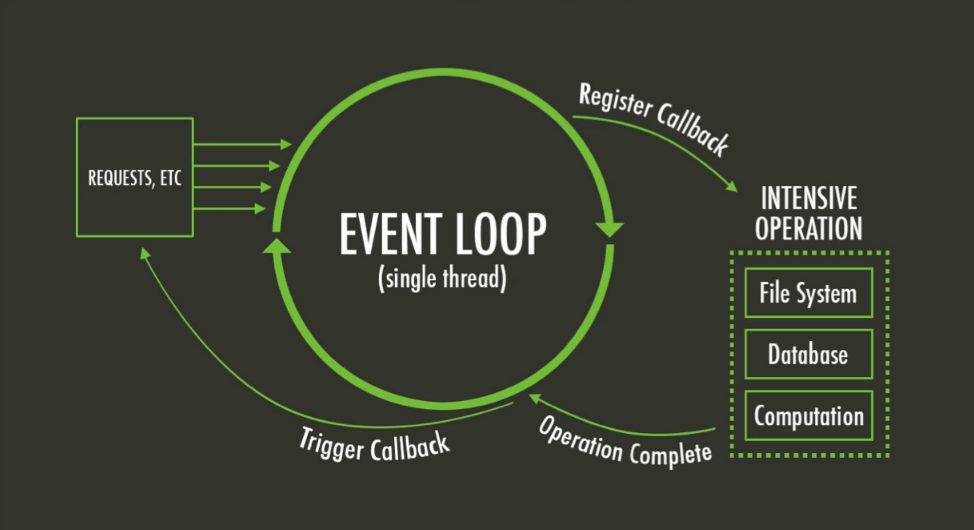
**Matlab:**

* JavaScript **ek time pe ek kaam** karta hai
* Ek hi **call stack**

❓ Problem:  
Agar long task ho (API call), to app freeze ho jayegi?

**9️⃣ Non-Blocking Event Loop 🔁🔥**





**Solution:**

**Event Loop**

Kaam ka flow:

1. Long task → background (Web APIs)
2. JS apna kaam karta rehta hai
3. Task complete → main thread mein wapas

setTimeout(() => {

console.log("done");

}, 3000);

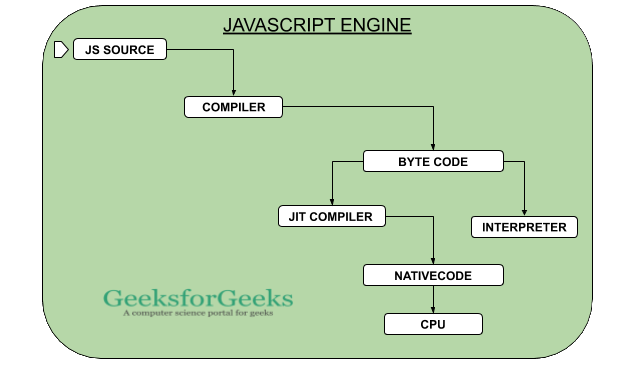
👉 Isliye:

* UI freeze nahi hoti
* Apps smooth lagte hain

**🧠 Final Summary (yaad rakhne wali)**

JavaScript tumhe **power deta hai bina low-level headache ke**,  
**flexible coding styles allow karta hai**,  
aur **event loop ke through smooth, non-blocking apps** banata hai.

**🧠 PART 1: JavaScript Engine kya hota hai?**



**🔹 Simple definition**

**JavaScript Engine ek program hota hai jo JavaScript code ko execute karta hai.**

Bas.  
Tumhara JS code **khud se nahi chalta**, usko chalane ke liye **engine chahiye**.

**🔹 Examples**

* Chrome → **V8 Engine**
* Node.js → **V8 Engine**
* Firefox → SpiderMonkey
* Safari → JavaScriptCore

👉 Matlab **browser ho ya server**, JS engine andar hota hi hota hai.

**🔧 Engine ke 2 main parts**

**1️⃣ Call Stack 📚**

**Yahin par tumhara JS code actually execute hota hai**

* Functions yahan **push / pop** hote hain
* Execution contexts yahin bante hain

Example:

function a() {

b();

}

function b() {

console.log("Hi");

}

a();

📌 Stack ka order:

a()

b()

console.log()

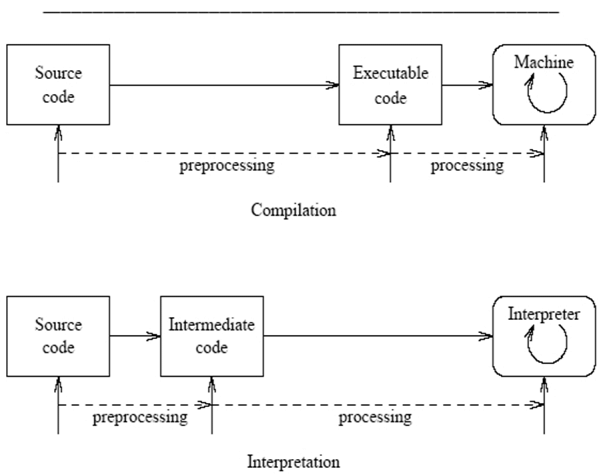
**2️⃣ Heap 🧺**

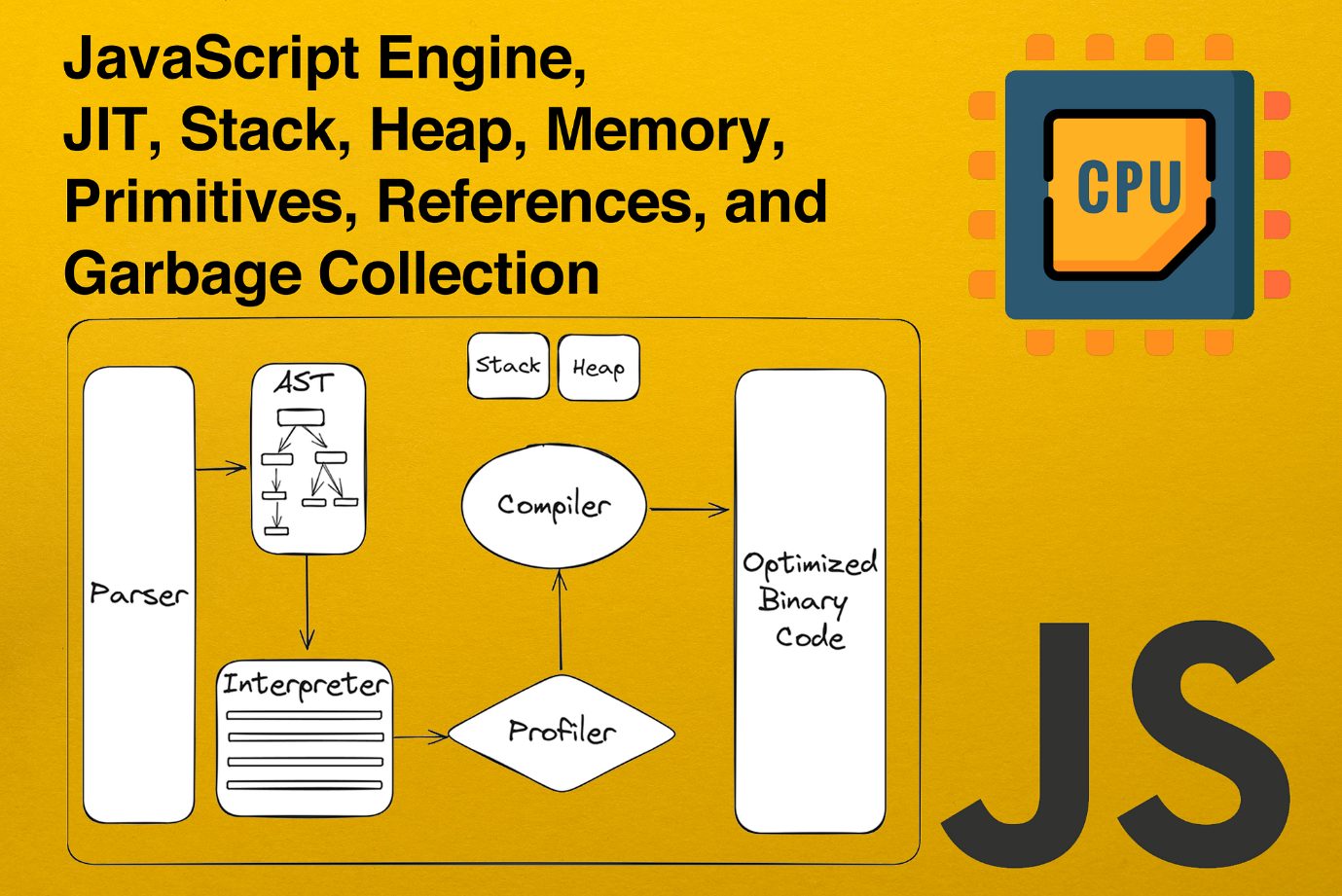
**Memory area jahan objects store hote hain**

const user = { name: "Nidhi" };

* user → stack mein reference
* { name: "Nidhi" } → heap mein actual object

**🧠 PART 2: Code machine code mein kaise badalta hai?**





Computer sirf **0s & 1s** samajhta hai 🤖  
Hum likhte hain:

let x = 10;

To beech mein **translation** hoti hai.

**🔁 3 Approaches (simple words mein)**

**1️⃣ Compilation**

* Pura code → machine code
* Phir run
* Example: C, C++

**2️⃣ Interpretation**

* Line by line read + run
* Slow hota hai

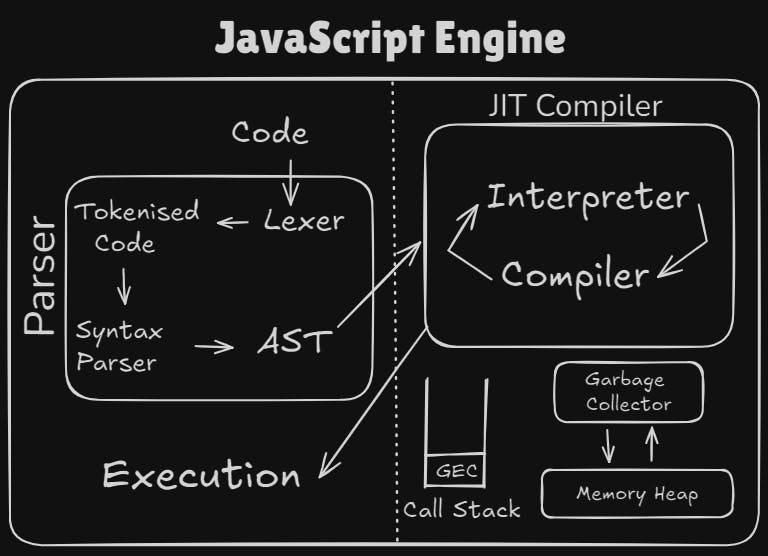
**3️⃣ ⭐ Just-In-Time (JIT) Compilation (JavaScript)**

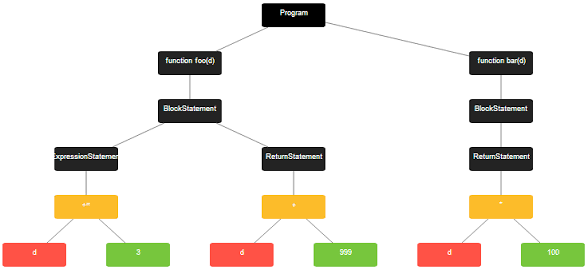
**Modern JavaScript engines ka secret sauce**

* Pura code → machine code
* Turant execute
* No separate executable file

👉 Fast + flexible = JS ❤️

**🧠 PART 3: JavaScript Engine ke andar kya hota hai?**





**Step-by-step journey 🧭**

**🔹 Step 1: Parsing**

* JS code read hota hai
* Syntax check hoti hai
* **AST (Abstract Syntax Tree)** banta hai

Example:

const x = 23;

AST = tree jaisa structure (engine ke liye)

❌ AST ≠ DOM  
(DOM HTML ka hota hai)

**🔹 Step 2: Compilation**

* AST → machine code
* Pehle **unoptimized** (fast start ke liye)

**🔹 Step 3: Execution**

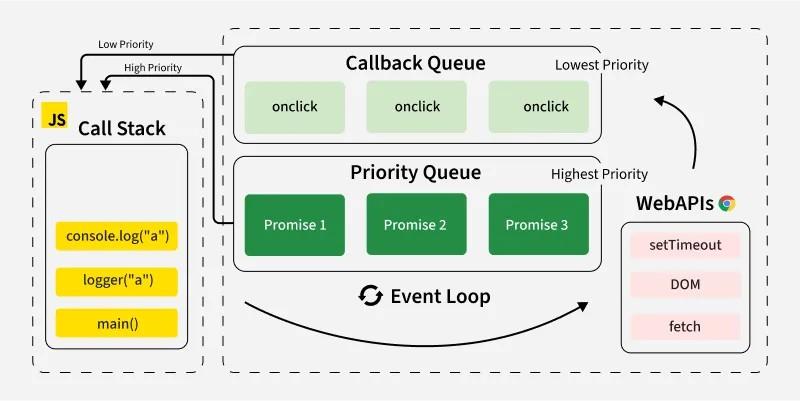
* Code **Call Stack** mein run hota hai

**🔹 Step 4: Optimization (🔥 powerful part)**

* Engine code observe karta hai
* Slow parts ko **background mein recompile**
* Better version se replace

👉 Isliye Chrome fast lagta hai 😎

**🧠 PART 4: JavaScript Runtime kya hota hai?**



**🔹 Runtime = poora environment**

Engine akela kaafi nahi hota

Runtime =  
✔ JS Engine  
✔ Web APIs  
✔ Callback Queue  
✔ Event Loop

**🌍 Browser Runtime (sabse important)**

**1️⃣ JS Engine**

* Call Stack
* Heap

**2️⃣ Web APIs 🧩**

Browser deta hai, JavaScript language ka part nahi

Examples:

* DOM
* setTimeout
* fetch
* console.log

Accessible via:

window

**3️⃣ Callback Queue 📥**

* Ready callbacks yahan wait karti hain

Example:

button.addEventListener("click", fn);

Click hua → fn queue mein aayi

**4️⃣ Event Loop 🔁 (heart ❤️)**

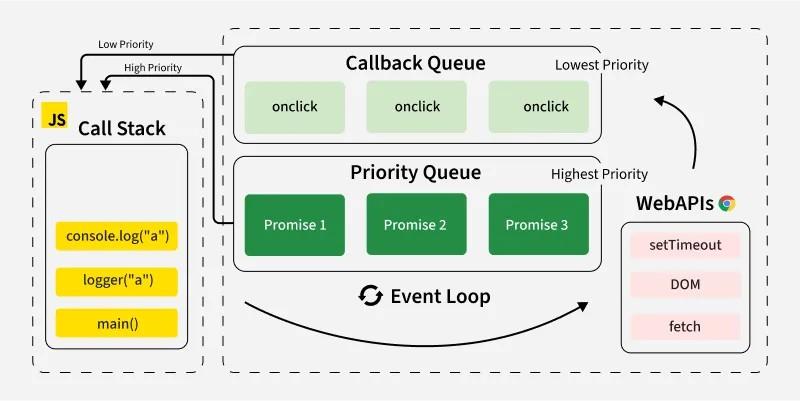
**Bridge between Queue & Stack**

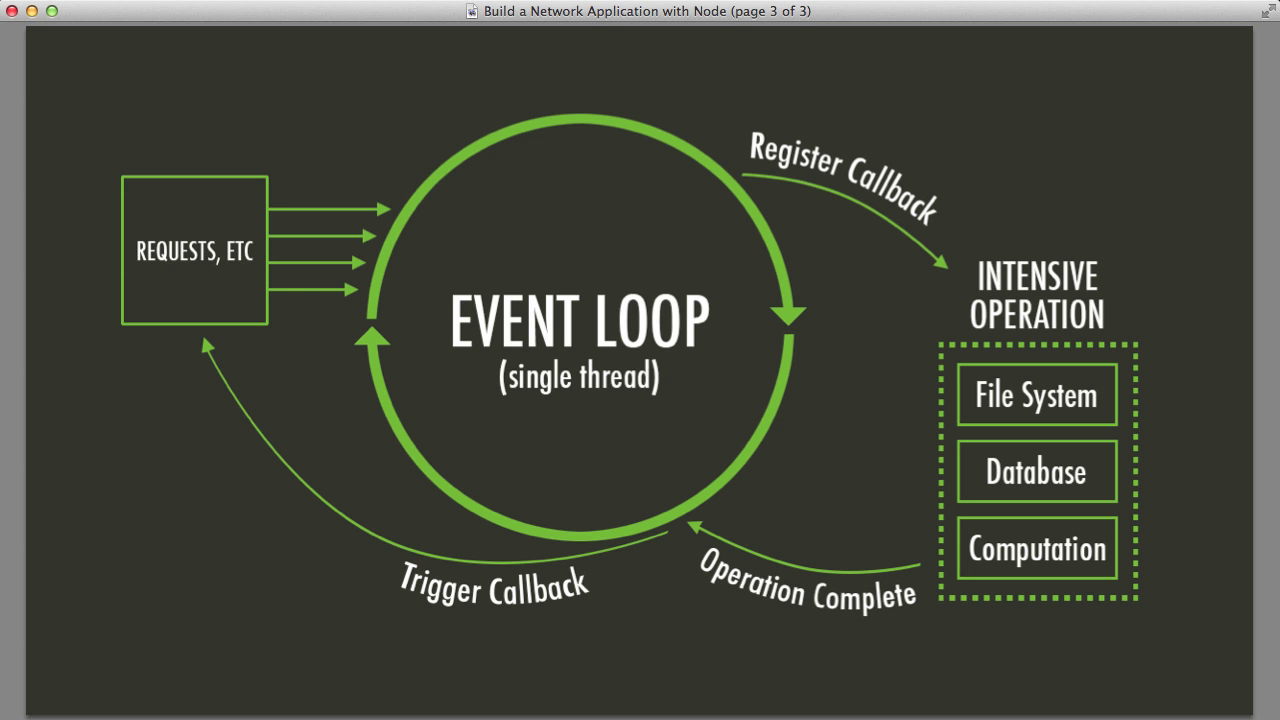
Rule:

* Jab Call Stack empty ho
* Tab Event Loop callback ko stack mein bhejta hai

👉 Isi wajah se JS **non-blocking** hoti hai

**🧠 PART 5: Node.js Runtime (short)**





Browser nahi hai ❌  
Isliye:

* ❌ DOM
* ❌ Web APIs

Instead:

* C++ bindings
* Thread pool
* Event loop same concept

**🧠 FINAL MENTAL MODEL (yaad rakhna 🔐)**

**JavaScript khud powerful nahi hoti,  
Engine usko chalata hai,  
Runtime usko duniya deta hai.**

**🔥 Interview-ready one-liners**

* JS Engine executes code
* Call Stack runs functions
* Heap stores objects
* JS uses JIT compilation
* Runtime ≠ Engine
* Event Loop makes JS non-blocking

**🌱 PART 1: JavaScript actually hai kya? (Big Picture)**

Sabse pehle ek **simple line yaad rakho**:

**JavaScript ek high-level, dynamic, single-threaded language hai jo engine ke through run hoti hai.**

Ab is line ko dheere-dheere todte hain.

**1️⃣ High-level language ka matlab kya?**

👉 **High-level = human friendly**

JavaScript me:

let x = 10;

Tumhe:

* memory allocate karna ❌
* RAM manage karna ❌
* CPU registers ka khayal ❌

👉 **ye sab JS khud karta hai**

**Contrast:**

* **C / C++ (low-level)** → developer ko memory manage karni padti hai
* **JS / Python (high-level)** → engine sab handle karta hai

Isliye JS easy hai, fast learning ke liye perfect 👍

**2️⃣ Garbage Collected – “Safai wala bhaiya” 🧹**

JS engine ke andar ek **garbage collector** hota hai.

👉 Agar koi object:

* use me nahi hai
* reference nahi hai

toh engine automatically memory se hata deta hai.

Tumhe kabhi:

free(x);

likhne ki zarurat nahi padti.

📌 **Tum sirf logic likho, safai engine karega**

**3️⃣ JavaScript Engine kya hota hai? 🤖**

**Simple definition:**

**JavaScript Engine = program jo JS code execute karta hai**

Examples:

* Chrome → **V8**
* Firefox → SpiderMonkey
* Node.js → V8

**Engine ke 2 main parts:**

| **Part** | **Kaam** |
| --- | --- |
| **Call Stack** | Code execute hota hai |
| **Heap** | Objects store hote hain |

**4️⃣ Compilation vs Interpretation vs JIT**

**Purana myth ❌**

“JavaScript interpreted language hai”

**Reality ✅**

**JavaScript uses Just-In-Time (JIT) Compilation**

**Flow:**

JS Code

↓

Parsing

↓

AST (Abstract Syntax Tree)

↓

Compilation → Machine Code

↓

Execution

👉 **Code pehle machine code me convert hota hai, phir turant run hota hai**

Isliye JS **fast bhi hai + flexible bhi**

**5️⃣ AST (Abstract Syntax Tree) 🌳**

Example:

const x = 23;

Engine isko ek **tree structure** me tod deta hai:

* variable declaration
* name = x
* value = 23

📌 AST ka **DOM se koi relation nahi**  
DOM = HTML ka tree  
AST = JS code ka tree

**6️⃣ JavaScript Runtime kya hota hai?**

Runtime = **Engine + Extra powers**

**Browser runtime me hota hai:**

* JS Engine
* Web APIs (DOM, fetch, setTimeout)
* Callback Queue
* Event Loop

**Node.js runtime me:**

* JS Engine
* C++ bindings
* Thread pool
* ❌ DOM

👉 **Runtime decides JS kya-kya kar sakta hai**

**🔄 PART 2: Execution Context (Sabse important)**

Ab asli game start hota hai 😄

**Execution Context kya hota hai?**

**Execution Context = box jisme JS code run hota hai**

🍕 Pizza analogy:

* Pizza = JS code
* Pizza box = Execution Context
* Box ke andar:
  + variables
  + arguments
  + scope
  + this keyword

**Types of Execution Context**

**1️⃣ Global Execution Context**

* Sirf **ek hi hota hai**
* Top-level code ke liye

let x = 10;

**2️⃣ Function Execution Context**

* **Har function call pe naya**

function add(a, b) {

return a + b;

}

**Execution Context ke andar kya hota hai?**

**🔹 1. Variable Environment**

* let, const, var
* function declarations
* arguments object (❌ arrow functions)

**🔹 2. Scope Chain**

* Bahar ke variables ka reference

**🔹 3. this keyword**

* Context ke hisaab se value
* ❌ arrow functions ka apna this nahi hota

📌 Ye sab **creation phase** me banta hai  
📌 Actual values **execution phase** me milti hain

**📚 PART 3: Call Stack – JS ka Google Maps 🗺️**

Call Stack = **Execution Contexts ka stack**

**Rule:**

**Jo upar hai, wahi execute ho raha hai**

**Example Code:**

function second() {

return 2;

}

function first() {

const b = second();

return b + 1;

}

const x = first();

**Stack movement:**

Global

↓

first()

↓

second()

↑

first()

↑

Global

📌 JS ek time pe **sirf ek kaam** karta hai  
📌 Isliye stack zaruri hai

**Important yaad rakhne wali baat 🧠**

* JS **single-threaded** hai
* Call stack ensure karta hai:
  + order lose na ho
  + function wahi se resume ho jahan ruka

**🔁 PART 4: Event Loop (Preview)**

JS slow kaam (API, timer, click) ko:

* background me bhej deta hai
* baad me callback queue se stack me laata hai

Isi wajah se JS **non-blocking** lagti hai ✨  
(isko hum next lectures me deep me padhenge)

**🧩 Final Mental Model (Exam / Interview ready)**

👉 Jab bhi JS code likho, socho:

1. Code → Engine ke paas gaya
2. Parsing → AST
3. Compilation → Machine code
4. Global Execution Context bana
5. Functions call hue → Stack me gaye
6. Return → Stack se nikle
7. Events → Event loop handle kare

**🌍 1. Scoping hota kya hai? (Basic Question)**

Sabse pehla sawal jo JavaScript puchti hai:

❓ **“Ye variable kaha milna chahiye?”**  
❓ **“Is jagah pe ye variable accessible hai ya nahi?”**

👉 **Isi ko bolte hain SCOPING**

📌 Short definition:

**Scoping decides where variables live & where we can access them**

**🧠 2. Lexical Scoping (Sabse important word)**

JavaScript **lexical scoped language** hai.

**Lexical ka matlab?**

👉 **Code me variable kaha likha gaya hai — wahi decide karta hai access**

❌ Ye matter nahi karta:

* function kisne call kiya
* call stack ka order

✅ Ye matter karta hai:

* function **code ke andar kaha likha hai**

🧩 Yaad rakhne wali line:

**“JS dekh ke decide karta hai, call dekh ke nahi”**

**📦 3. Scope kya hota hai?**

**Scope = wo area jaha variable declare hua**

Aur…

**Scope of a variable = wo area jaha variable accessible hai**

(Thoda subtle difference hai, par interviews me kaam aata hai)

**🧱 4. JavaScript me 3 types ke scope hote hain**

**1️⃣ Global Scope 🌍**

const name = "Jonas";

✔ Function ke bahar  
✔ Har jagah accessible  
✔ Sabse upar scope chain me

👉 Global variables = **global scope ke residents**

**2️⃣ Function Scope 🧠**

function calcAge() {

const age = 30;

}

✔ Sirf function ke andar accessible  
❌ Bahar access karoge → ReferenceError  
✔ Isko **local scope** bhi bolte hain

📌 Important:

* function declaration
* function expression
* arrow function  
  👉 **sab apna scope banate hain**

**3️⃣ Block Scope (ES6 ke baad) 🧩**

if (true) {

const x = 10;

let y = 20;

}

✔ {} ke andar ka area = block  
✔ Sirf let & const ke liye  
❌ var ko farak nahi padta

**⚠️ Golden rule:**

| **Keyword** | **Scope** |
| --- | --- |
| let | block scoped |
| const | block scoped |
| var | function scoped |

**🚨 var ka danger zone**

if (true) {

var test = "hello";

}

console.log(test); // 😱 works

👉 Isliye:

* modern JS me **var avoid**
* let & const use karo

**🔗 5. Scope Chain (Magic yahin hota hai ✨)**

Rule:

**Har scope ko apne parent scopes ke variables ka access hota hai**

Example:

const myName = "Jonas";

function first() {

const age = 30;

function second() {

const job = "teacher";

console.log(myName, age, job);

}

second();

}

first();

**Access ka flow:**

second scope

↑

first scope

↑

global scope

✔ second() → age mil gaya  
✔ second() → myName mil gaya

📌 **Variables copy nahi hote**, sirf lookup hota hai

**🔍 6. Variable Lookup kaise hota hai?**

Agar JS ko variable nahi mila:

1. current scope me dekhega
2. parent scope me dekhega
3. parent ka parent…
4. global tak
5. fir bhi nahi mila → ❌ ReferenceError

👉 Isko bolte hain **variable lookup**

**🚫 7. Scope chain one-way hoti hai**

❌ Parent scope **child ka variable** nahi dekh sakta  
❌ Sibling scopes ek dusre ke variables nahi dekh sakte

function first() {

const a = 10;

function second() {

const b = 20;

}

console.log(b); // ❌ error

}

📌 **Scope chain upar jaati hai, neeche nahi**

**👯‍♂️ Sibling scopes example (slides ka purple & yellow box)**

function parent() {

if (true) {

const x = 10;

}

function child() {

const y = 20;

}

}

❌ if block ko child() ka access nahi  
❌ child() ko if block ka access nahi

👉 Kyunki:

* dono siblings hain
* ek dusre ke andar likhe hi nahi gaye

**📞 8. Scope Chain vs Call Stack (MOST CONFUSING PART)**

**🔹 Call Stack**

👉 **Functions kis order me call hue**

**🔹 Scope Chain**

👉 **Functions code me kis order me likhe gaye**

📌 Ye dono **independent cheezein hain**

**Important line (interview ready):**

❌ Scope chain ka **call order se koi lena dena nahi**

**Example jo galat lagta hai but sahi hai**

function first() {

second();

}

function second() {

third();

}

function third() {

console.log(a);

}

const a = 10;

first();

✔ third() ko a mil jaata hai  
❌ Chahe second ne call kiya ho

👉 Kyunki third() **global me likha hai**

**🧠 FINAL MENTAL MODEL (Yaad rakhne ke liye)**

Socho JS ek **lift** hai 🛗

* Lift **sirf upar ja sakti hai**
* Sideways ❌
* Neeche ❌

Aur lift ka map = **scope chain**

**📌 Ultra-Short Summary (Notes ke liye)**

* Scoping = variables kaha milenge
* JS uses **lexical scoping**
* 3 scopes:
  + global
  + function
  + block (let/const only)
* var = function scoped
* Scope chain:
  + child → parent → global
  + one way only
* Scope chain ≠ call stack

**🔹 STEP 1: Global Scope se shuru karte hain**

function calcAge(birthYear) {

const age = 2037 - birthYear;

console.log(firstName);

}

**Yahan kya hua?**

* calcAge **global scope** me likha gaya
* Isliye:
  + calcAge → global scope ka member
  + Function ke andar → **function scope**

**Ab ek global variable banaya:**

const firstName = 'Jonas';

📌 **Global scope** me firstName aa gaya

**Function call:**

calcAge(1991);

**Output:**

Jonas

**WHY ❓ (Most important)**

* JS console.log(firstName) pe:
  1. calcAge scope me dekhta hai ❌
  2. parent (global) scope me jata hai ✅
* **Scope chain ka use hua**

👉 **Scope chain = current → parent → global**

**❌ Agar variable exist hi na kare?**

console.log(lastName);

❌ Error:

lastName is not defined

👉 JS poori scope chain dekh ke thak gaya 😄

**🔹 STEP 2: Nested Function (Scope chain deep hoti jaati hai)**

function calcAge(birthYear) {

const age = 2037 - birthYear;

function printAge() {

console.log(`You are ${age}, born in ${birthYear}`);

}

printAge();

}

**Output:**

You are 46, born in 1991

**Yahan kya ho raha hai?**

* printAge ke paas:
  + age ❌ local nahi
  + parent (calcAge) se mila ✅
  + birthYear bhi parent se mila ✅

📌 **Function parameters bhi variables jaise behave karte hain**

**❌ Outer scope child ko access nahi kar sakta**

console.log(age); // ❌

printAge(); // ❌

👉 **Scope chain one-way hoti hai**

* child → parent ✅
* parent → child ❌

**🔹 STEP 3: Global → Function → Function (long lookup)**

function printAge() {

console.log(firstName);

}

Lookup order:

printAge scope ❌

calcAge scope ❌

global scope ✅

📌 JS jab tak mil na jaye tab tak upar jaata rehta hai

**🔹 STEP 4: Block Scope (if block)**

if (birthYear >= 1981 && birthYear <= 1996) {

const str = `Oh, you're a millennial, ${firstName}`;

console.log(str);

}

**Works fine ✅**

**Bahar access karne pe?**

console.log(str);

❌ Error:

str is not defined

👉 **const & let = block scoped**

**⚠️ STEP 5: var ka drama 😈**

if (true) {

var millennial = true;

}

console.log(millennial);

✅ Works 😱

**WHY?**

* var = **function scoped**
* block ignore kar deta hai

📌 Isi wajah se:

❌ var use mat karo  
✅ let / const use karo

**🔹 STEP 6: Functions bhi block scoped hote hain (strict mode)**

if (true) {

function add(a, b) {

return a + b;

}

}

add(2, 3);

❌ Error (strict mode me)

👉 **ES6 ke baad functions bhi block scoped hote hain**

**🔹 STEP 7: Variable Shadowing (same name, different scope)**

if (true) {

const firstName = 'Steven';

console.log(firstName);

}

console.log(firstName);

**Output:**

Steven

Jonas

**Explanation:**

* Block ke andar → Steven
* Bahar → global Jonas

📌 **Inner scope ka variable outer ko shadow karta hai**

**🔹 STEP 8: Reassign vs Redeclare (VERY IMPORTANT)**

**Case 1️⃣: Reassign outer variable**

let output = 'original';

if (true) {

output = 'new output';

}

console.log(output);

✅ Output:

new output

👉 **Same variable modify hua**

**Case 2️⃣: Redeclare (new variable)**

let output = 'original';

if (true) {

const output = 'new output';

}

console.log(output);

✅ Output:

original

👉 **Naya variable bana, parent untouched**

**🧠 FINAL RULES (EXAM + INTERVIEW GOLD)**

**🔑 Scoping Rules:**

1. JS **lexical scoped** hai
2. Scope chain = current → parent → global
3. Block scope:
   * let, const ✅
   * var ❌
4. Inner scope:
   * parent variables read kar sakta hai
   * overwrite bhi kar sakta hai
5. Same name allowed (shadowing)

**🧩 One-line Memory Trick**

**“JS pehle apne ghar me dekhta hai,  
phir maa-baap ke ghar,  
par kabhi bachche ke ghar nahi.”** 😄