**Day 1 – Ultimate JavaScript Revision Pack**

**1.] Variables**

 var → function scope hota hai, re-declare ho sakta hai.

 let → block scope hota hai, re-declare nahi ho sakta.

 const → block scope + reassign nahi ho sakta (but object ke andar values change ho sakti hain). [DETAILED EXPLANATION BELOW]

**2.] Primitives vs Reference Types**

| **Type** | **Example** | **Copy Behavior** |
| --- | --- | --- |
| Primitive: | number, string, boolean, null, undefined | Copy by value → independent |
| Reference | object, array, function | Copy by reference → same memory |

// Primitive

let a = 10;

let b = a;

b = 20;

console.log(a,b); // 10 20

// Reference

let arr1 = [1,2];

let arr2 = arr1;

arr2.push(3);

console.log(arr1,arr2); // [1,2,3] [1,2,3]

**3.] Null vs Undefined**

let a;

let b = null;

console.log(a); // undefined

console.log(b); // null

* undefined → declared but no value (value just abhi nahi di hai, but it can have a value)
* null → explicitly empty (ham janbujh k chahte hai ki voh bina value k rahe).

**⚡4.] Operators in JavaScript**

👉 Operator = **symbol jo values pe operation karta hai**.  
Example: +, -, \*, ===, &&, etc.

**🔹 Types of Operators**

**1. Arithmetic Operators (maths wale)**

* + (addition) → 5 + 3 = 8
* - (subtraction) → 5 - 3 = 2
* \* (multiplication) → 5 \* 3 = 15
* / (division) → 6 / 2 = 3
* % (modulus → remainder) → 7 % 2 = 1
* \*\* (power) → 2 \*\* 3 = 8

**2. Assignment Operators**

👉 Variable ko value dene ke liye.

* = → let x = 5;
* += → x += 2; (x = x + 2)
* -= → x -= 2;
* \*= → x \*= 2;
* /= → x /= 2;

**3. Comparison Operators (condition check karte hain → true/false return karte hain)**

👉 Ye mostly **control flow ke andar use hote hain** (if/else, while loops, etc.).

* == → value equal hai kya? (DATAtype ignore karta hai)  
  5 == '5' → true
* === → value + DATAtype dono equal hai kya?  
  5 === '5' → false
* != → not equal
* !== → value ya type match nahi hona chahiye
* > (greater than) → 10 > 5 → true
* < (less than) → 10 < 5 → false
* >= (greater or equal)
* <= (less or equal)

👉 Example in control flow:

let age = 18;

if (age >= 18) {

console.log("Adult");

}

**4. Logical Operators (multiple conditions check karne ke liye)**

* && (AND) → dono true hone chahiye  
  (age >= 18 && citizen === true)
* || (OR) → ek bhi true ho to chalega  
  (marks > 90 || sportsCertificate === true)
* ! (NOT) → ulta kar deta hai  
  !(true) → false

👉 Example:

let age = 20;

let hasID = true;

if (age >= 18 && hasID) {

console.log("Entry allowed");

}

**5. Increment / Decrement Operators**

* ++ (increment by 1)
* -- (decrement by 1)

👉 Example:

let count = 5;

count++; // 6

count--; // 5

**6. Ternary Operator (short if-else)**

👉 Shortcut of if-else.

let age = 18;

let result = (age >= 18) ? "Adult" : "Minor";

console.log(result);

**7. Type Operators**

* typeof → data type check karna  
  typeof 123 → "number"
* instanceof → object kis class ka hai check karna

### 5.] Control Flow:

**🚦 Control Flow in JavaScript**

👉 Matlab: “Program ka **traffic signal**” – kis line ka code chalna hai, kab chalna hai, aur kitni baar chalna hai, yeh decide karna.

By default JS **upar se neeche** chalti hai, lekin hum **control flow statements** se uska rasta badal sakte hain.

**🔹 1. Conditional Statements (Decision making)**

Socho tu bazar gaya hai – agar barish ho rahi hai to chhatri leke jaayega, warna bina chhatri ke.  
👉 Yehi kaam if-else karte hain.

**(a) if**

let age = 20;

if (age >= 18) {

console.log("Vote de sakte ho ✅");

}

➡ Agar condition true hai → andar ka code chalega.

**(b) if...else**

let age = 15;

if (age >= 18) {

console.log("Vote de sakte ho ✅");

} else {

console.log("Abhi chhote ho ❌");

}

➡ Agar condition galat ho → else wala chalega.

**(c) if...else if...else**

let marks = 75;

if (marks >= 90) {

console.log("Grade A 🏆");

} else if (marks >= 60) {

console.log("Grade B 👍");

} else {

console.log("Grade C 👌");

}

➡ Jaise school me marks ke hisaab se alag-alag grade.

**(d) switch**

Socho tu restaurant gaya hai aur menu card me choice deni hai –  
👉 switch ek menu card ki tarah hai.

let day = "Monday";

switch (day) {

case "Monday":

console.log("Kaam shuru ho gaya 😓");

break;

case "Friday":

console.log("Weekend aane wala hai 😍");

break;

default:

console.log("Beech ka din 😅");

}

**🔹 2. Loops (Repetition)**

Socho tu push-ups kar raha hai – ek hi kaam baar-baar repeat ho raha hai.  
👉 Yeh kaam loops karte hain.

**(a) for loop → fix baar repeat karna**

for (let i = 1; i <= 5; i++) {

console.log("Push-up:", i);

}

➡ 1 se 5 tak push-ups print honge.

**(b) while loop → jab tak condition true hai, tab tak repeat**

let i = 1;

while (i <= 5) {

console.log("Push-up:", i);

i++;

}

➡ Jab tak condition sahi hai, loop chalta rahega.

**(c) do...while loop → ek baar zaroor chalega**

let i = 1;

do {

console.log("Push-up:", i);

i++;

} while (i <= 5);

➡ Pehle ek baar chalega, fir condition check karega.

**(d) for...of → array ke liye**

let fruits = ["apple", "banana", "mango"];

for (let fruit of fruits) {

console.log(fruit);

}

**(e) for...in → object ke liye**

let person = { name: "Amit", age: 22 };

for (let key in person) {

console.log(key, ":", person[key]);

}

**🔹 3. Control Flow Modifiers**

Jaise game khelte waqt kabhi "pause" karte ho ya "skip" kar dete ho.

* **break** → loop tod deta hai.

for (let i = 1; i <= 5; i++) {

if (i === 3) break;

console.log(i);

}

// Output: 1 2

* **continue** → current step skip kar deta hai.

for (let i = 1; i <= 5; i++) {

if (i === 3) continue;

console.log(i);

}

// Output: 1 2 4 5

* **return** → function ke andar use hota hai, wahan hi ruk jaata hai.

function greet(name) {

if (!name) {

return "No name provided";

}

return "Hello " + name;

}

console.log(greet("Trupti"));

**⚡ Final Summary (Aam Bhasa)**

👉 JavaScript ke control flow me yeh sab hota hai:

1. **Decision making** → if, if-else, switch
2. **Repetition (loops)** → for, while, do-while, for...of, for...in
3. **Flow modifiers** → break, continue, return

Bas inhi ke through program apna “rasta” badalta hai.

### 6.] Shallow Copy (1st level new)

Methods: spread ..., slice(), concat()

let obj1 = { name:"Aman", address:{city:"Pune"} };

let obj2 = {...obj1};

obj2.address.city = "Delhi";

console.log(obj1.address.city); // Delhi → nested shared!

* Arrays example:

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

let arr2 = [...arr1];

arr2[0].push(99);

console.log(arr1,arr2); // [[1,2,99],[3,4]] both changed

Spread operator (...) sirf **shallow copy** banata hai → outer array naya hota hai, par andar ke objects/arrays ke references same rehte hain.

Isliye ek ko change karoge to doosre me bhi reflect hoga.

**7.] Deep Copy (Full independent)**

**Methods:** JSON.parse(JSON.stringify(obj)), structuredClone()

### 1. **JSON.parse(JSON.stringify(obj))**

* Pehle object/array ko **JSON string** me convert karta hai.
* Fir usko dobara **parse karke naya object/array** banata hai.
* Is wajah se **har nested object/array ki nayi copy** ban jaati hai (reference break ho jata hai).

⚠️ Limitation:

Date, Map, Set, undefined, functions, circular references handle nahi hota.

EXAMPLE:

let obj1 = { name: "Raj", address: { city: "Delhi" } };

let obj2 = JSON.parse(JSON.stringify(obj1)); // deep copy

obj2.address.city = "Mumbai";

console.log(obj1.address.city); // Delhi (original unaffected)

console.log(obj2.address.city); // Mumbai (independent copy)

*👉 Yaha obj2.address ek naya object hai, isliye changes obj1 ko affect nahi karte.*

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

let arr3 = JSON.parse(JSON.stringify(arr1)); // deep copy

arr3[0].push(500);

console.log(arr1); // [ [1,2], [3,4] ] (original safe)

console.log(arr3); // [ [1,2,500], [3,4] ] (independent copy)

*👉 Yaha inner arrays bhi alag ban gaye, isliye arr1 unaffected hai.*

### 2. **StructuredClone(obj)** (Modern & Better ✅)

*Ye modern JS method hai (Node.js 17+ / Browser support).*

*Deep copy banata hai* ***without JSON limitations****.*

*Date, Map, Set, RegExp, TypedArray sab handle kar leta hai.*

Example:

let obj1 = { name: "Raj", date: new Date(), nested: { x: 10 } };

let obj2 = structuredClone(obj1);

obj2.nested.x = 99;

console.log(obj1.nested.x); // 10

console.log(obj2.nested.x); // 99

console.log(obj1.date === obj2.date); // false (new Date object cloned)

**8.] Brain Shortcut / Memory Trick**

* Shallow → “Roof new, rooms same” (first level new, nested shared) [ [], [] ]
* Deep → “Brand new house” (everything new, original unaffected)

One-line Interview Explanation

"Shallow copy creates a new container but nested objects/arrays share reference; deep copy duplicates every level so changes in the copy do not affect the original."

**9.] ASCII Memory Diagrams**

**Shallow Copy – Object**

obj1

+---------+ +----------------+

| name | -------> | "Aman" |

| address | -------> | { city:"Pune"} | (same reference)

+---------+

obj2

+---------+

| name | -------> "Aman" (new copy)

| address | -------> SAME reference as obj1.address

**Deep Copy – Object**

obj1

+---------+

| name | -> "Aman"

| address | -> { city: "Pune" }

obj2

+---------+

| name | -> "Aman" (new)

| address | -> { city: "Delhi" } (new object)

**Shallow Copy – Array**

arr1 arr2 (shallow copy)

+---------+ +---------+

| [1,2] | --------> | [1,2] | <-- SAME reference!

| [3,4] | | [3,4] | <-- SAME reference!

+---------+ +---------+

**Deep Copy – Array**

arr1 (original)

+---------+

| [1,2,99]|

| [3,4] |

+---------+

arr2 (deep copy)

+---------+

| [1,2,99,500] |

| [3,4] |

+--------

💡 **Tips for Revision:**

* Type and run every code snippet in console for memory retention
* Remember ASCII diagrams and Brain shortcuts for interview explanation
* Practice extra questions to impress interviewer with fundamentals

## PRACTICE:

⚡ Ab tu try kar ek chhoti practice:

* Ek variable x banao bina value ke (undefined).
* Ek variable y banao jo null ho.
* Ek object {city:"Pune"} banao aur uska ek dusra reference lo, phir property badal ke dekho.

**ANS:**

let a;

let b = null;

let obj1 = { city: "Pune" };

let obj2 = obj1;

obj2.city = "Goa";

console.log(a); // undefined

console.log(b); // null

console.log(obj1); // { city: "Goa" }

console.log(obj2); // { city: "Goa" } //dono chnge hogye cause of same reference.

**⚡ Output Explanation:**

1. a → ❌ koi value assign hi nahi, isliye (undefined).
2. b → ✅ explicitly khud ne null assign kiya, matlab "empty".
3. obj1 aur obj2 → dono ek hi memory ko point karte hain → jab obj2.city = "Goa" kiya, toh obj1.city bhi "Goa" ban gaya.

👉 Isiliye interview mein interviewer puchta hai:  
"**Primitive aur Reference type me kya difference hai?**"  
Aur tu confidently bol:

* Primitive (string, number, boolean...) → copy by value
* Reference (objects, arrays, functions) → copy by reference.

**Bhai, ab ek chhota challenge:**let num1 = 10; let num2 = num1; num2 = 20;  
Ab console.log(num1, num2); ka output kya hoga? 🔥

ANS:

let num1 = 10;

let num2 = num1; // yaha num2 me num1 ka value copy hua (10)

num2 = 20; // sirf num2 ka value change hua

**QUE:**

let arr1 = [1, 2, 3]; // arr1 ek array ban gaya

let arr2 = arr1; // arr2 same memory ko point kar raha hai (copy nahi bana!)

arr2.push(4); // isne arr2 me 4 add kiya, but kyunki dono same memory share kar rahe hain → arr1 bhi update ho jayega

### ✅ Output:

arr1 = [1, 2, 3, 4]

arr2 = [1, 2, 3, 4]

### ✅ Ways to make a new copy of an array

1. **Spread Operator (...)**

let arr1 = [1, 2, 3];

let arr2 = [...arr1]; // new copy bana

arr2.push(4);

console.log(arr1); // [1, 2, 3]

console.log(arr2); // [1, 2, 3, 4]

1. **slice()**

let arr1 = [1, 2, 3];

let arr2 = arr1.slice(); // new copy

arr2.push(5);

console.log(arr1); // [1, 2, 3]

console.log(arr2); // [1, 2, 3, 5]

1. **concat()**

let arr1 = [1, 2, 3];

let arr2 = [].concat(arr1);

arr2.push(6);

console.log(arr1); // [1, 2, 3]

console.log(arr2); // [1, 2, 3, 6]

⚡ Challenge time bhai:

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

let arr2 = [...arr1];

arr2[0].push(99);

console.log(arr1);

console.log(arr2);

### ✅ Output:

arr1 = [[1, 2, 99], [3, 4]]

arr2 = [[1, 2, 99], [3, 4]]

JSON.parse(JSON.stringify(arr1)) ek **deep copy** banata hai.

* Deep copy me har ek level ka data **naya ban jata hai**, koi bhi reference share nahi hota.
* Matlab arr1[0] aur arr2[0] ab totally alag arrays hain.

### Example:

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

let arr2 = JSON.parse(JSON.stringify(arr1));

arr2[0].push(500);

console.log(arr1); // [[1,2], [3,4]]

console.log(arr2); // [[1,2,500], [3,4]]

👉 So ab rule clear hai:

Shallow copy (spread, slice, concat) → sirf ek level tak naya banega, nested objects/arrays abhi bhi same reference hote hain.

Deep copy (JSON method, ya structuredClone) → har level pe naya data banega.

⚡ Challenge for you bhai:

let obj1 = { name: "Aman", address: { city: "Pune" } };

let obj2 = { ...obj1 };

obj2.address.city = "Delhi";

console.log(obj1);

console.log(obj2);

### ✅ Output:

obj1 = { name: "Aman", address: { city: "Delhi" } }

obj2 = { name: "Aman", address: { city: "Delhi" } }

**Explanation:**

* Spread operator { ...obj1 } ek **shallow copy** banata hai.
* name: "Aman" primitive hai → alag copy ban gayi.
* address ek object hai (reference type) → dono (obj1 aur obj2) usi **same memory** ko point karte hain.
* Jab obj2.address.city = "Delhi" kiya, to obj1.address.city bhi change ho gaya.

👉 Agar hume alag-alag copy chahiye hota (deep copy), toh use karte:

let obj2 = JSON.parse(JSON.stringify(obj1));

Phir obj1 aur obj2 independent hote.

⚡ Ab tu try karke bata:  
Agar mai deep copy karu:

let obj1 = { name: "Aman", address: { city: "Pune" } };

let obj2 = JSON.parse(JSON.stringify(obj1));

obj2.address.city = "Delhi";

console.log(obj1);

console.log(obj2);

### ✅ Output:

obj1 = { name: "Aman", address: { city: "Pune" } }

obj2 = { name: "Aman", address: { city: "Delhi" } }

### Explanation:

* JSON.parse(JSON.stringify(...)) ne obj1 ka ek **completely independent deep copy** banaya.
* Ab obj1.address aur obj2.address alag memory blocks hain.
* Isliye obj2.address.city = "Delhi" karne se sirf obj2 change hua, obj1 unaffected raha.

RULES:

**1.] LET:**

### 1. Block Scope hota hai

* Jo variable let se declare hota hai, uski validity sirf us block {} ke andar hoti hai.
* Example:

{

let x = 10;

console.log(x); // 10

}

console.log(x); // ❌ Error: x is not defined

Yaha x sirf {} ke andar valid hai. Bahar use nahi kar sakte.

### 2. Re-declare nahi ho sakta (same scope me)

* Agar aap ek hi scope me ek hi naam ka variable let se do baar declare karne ki koshish karoge, to error aayega.

let y = 20;

let y = 30; // ❌ Error: Identifier 'y' has already been declared

Lekin update karna allowed hai:

let y = 20;

y = 30; // ✅ valid

console.log(y); // 30

**2.] CONST:**

### 1. ****Objects & Arrays ke andar changes allowed hote hain****

* const sirf **reference ko lock karta hai**, **value ke andar ki cheez change ho sakti hai**.

const obj = { name: "Amit", age: 21 };

obj.age = 22; // ✅ allowed

console.log(obj); // { name: "Amit", age: 22 }

obj = { name: "Raj" }; // ❌ Error: Assignment to constant variable

Same with arrays:

const arr = [1, 2, 3];

arr.push(4); // ✅ allowed

console.log(arr); // [1, 2, 3, 4]

arr = [5, 6]; // ❌ Error

(BLOCK SCOPED ND NON-CHANGEBLE SAME H LET K JESE)

### ⚡ WOW(Imp):

1. **var ka ek hidden issue** → hoisting ka example (declare ho jata hai par undefined hota hai).
2. console.log(a); // undefined
3. var a = 10;

(Ye interviewer puchh lete hain)

1. **Operator precedence** ka ek line mention → kaunsa operator pehle chalega (e.g. \* before +).
2. **Strict equality (===)** ko highlight karna zaruri hai interview ke liye, kyunki JS ka sabse common confusion hai == vs ===.  
   (tune already add kiya hai, bas thoda emphasize kar dena "Always prefer === in real projects").

### ✅ FIT-CHECK:

1. **Variables (var, let, const)** → clear explanation + examples + block scope vs function scope ✅
2. **Primitives vs Reference** → copy by value vs copy by reference, example with arrays + objects ✅
3. **Null vs Undefined** → mast example diya, interviewer style explanation bhi sahi ✅
4. **Operators** → categories wise (Arithmetic, Assignment, Comparison, Logical, Increment/Decrement, Ternary, Type) + examples ✅
5. **Control Flow** →
   * if, if-else, else if, switch → real life analogies (barish, menu card, school grade) → 👌
   * Loops → for, while, do...while, for...of, for...in → push-up analogy → mast ✅
   * Flow modifiers (break, continue, return) → simple aur sahi ✅
6. **Shallow Copy vs Deep Copy** → spread, slice, concat vs JSON.parse, structuredClone → diagrams + memory trick → 👑
7. **ASCII diagrams** → interview me visual samjhane ke liye 10/10 ✅
8. **Practice challenges** → har ek topic ke baad chhoti challenge → mast revision hack ✅
9. **LET / CONST extra details** → block scope, redeclare, reassignment, object/array modification → sahi pakda ✅

**Day 1 Complete ✅**