INTERNSHIP ASSEGINMENT QUESTIONS

1) Difference between HTTP and HTTPS?

Ans:- **HTTP:-**

1. HTTP sends data in plain text, so anyone can read it.
2. It does not protect passwords or personal data.
3. Hackers can easily intercept information.
4. It does not use encryption.
5. HTTP works on port 80.
6. It does not show a lock symbol in the browser.
7. Browsers mark HTTP sites as "Not Secure."
8. It is used only for non-sensitive information.

HTTPS:-

1. HTTPS sends data in encrypted form, so it stays private.
2. It protects passwords and personal data.
3. Hackers cannot read intercepted data.
4. It uses SSL/TLS encryption for security.
5. HTTPS works on port 443.
6. It shows a lock 🔒 symbol in the browser.
7. Browsers trust HTTPS websites.
8. It is used for secure websites like banking and shopping.

---

2) About Web Architecture?

Ans:- Web architecture is the structure of a website that shows how the client, server, and database interact to deliver web pages over the internet.

1. Client: The client is the user's device (computer or mobile) and browser that sends a request, like opening a website.
2. Web Server: The web server receives the request from the client and processes it.
3. Application Server: The application server runs the logic of the website, such as checking login details or processing data.
4. Database: The database stores website data like user details, passwords, and content.
5. Request & Response: The client sends a request, and the server sends back a response (webpage or data).
6. Internet / Network: The internet connects the client and server and allows data to travel between them.
7. Protocols: Protocols like HTTP/HTTPS are used to communicate between client and server.

In simple words:

User → Browser → Server → Database → Server → Browser

---

3) About DNS (Domain Name System)?

Ans:- DNS (Domain Name System) is a system that translates human-readable domain names (like www.google.com) into IP addresses (like 142.250.190.14) so that computers can identify and communicate with each other on the internet.

How DNS works (step by step)

When you type a website name into your browser:

1. **Browser cache check:**
   Your browser checks if it already knows the IP address.
2. **Operating system cache:**
   If not found, your OS checks its DNS cache.
3. **DNS Resolver (ISP):**
   The request goes to a DNS resolver (usually provided by your ISP or services like Google DNS).
4. **Root DNS Server:**
   Directs the query to the correct Top-Level Domain server.

5. **TLD Server** (e.g., .com, .org):
   Points to the domain's authoritative DNS server.
6. **Authoritative DNS Server:**
   Returns the correct IP address.
7. **Website loads:**
   Your browser connects to the IP address and loads the site

Types of DNS servers

- **Recursive Resolver –** Finds the answer for you
- **Root Server –** Starting point of DNS
- **TLD Server –** Handles .com, .net, etc.
- **Authoritative Server –** Final source of truth

Common DNS records
DNS stores different types of records:

Record	**Purpose**
A	Maps domain → IPv4 address
AAAA	Maps domain → IPv6 address
CNAME	Alias of another domain
MX	Mail server information
TXT	Verification, security info
NS	Nameservers for a domain

---

4) Difference between ARIA and WIA ARIA?
Ans:- Here are some key differences between ARIA and WIA-ARIA:

Aspect	**ARIA**	**WAI-ARIA**
Full form	Accessible Rich Internet Applications	Web Accessibility Initiative – Accessible Rich Internet Applications
What it refers to	Informal or shortened name	Official, formal name
Who defines it	—	W3C (via the Web Accessibility Initiative)
Purpose	Make web apps accessible to assistive technologies	Same purpose
Usage in practice	Common in developer conversations	Used in specifications, standards, documentation
Technical features	Roles, states, properties	Same roles, states, and properties
Example usage	"Use ARIA roles"	"According to WAI-ARIA 1.2 spec"

---

5) How Arithmetic operators and Concatenation operators works in browser?
Ans: Arithmetic Operators in the Browser
Arithmetic operators in JavaScript include:

- + → addition
- - → subtraction
- * → multiplication
- / → division
- % → modulus (remainder)
- ** → exponentiation

How they work:

1. **Numbers**:
   If both operands are numbers, the operation works normally:
   console.log(5 + 3); // 8

```
console.log(5 * 3); // 15
console.log(5 % 3); // 2
```

2. **String + Number / String**:
   Most arithmetic operators (-, *, /, %) **convert strings to numbers automatically**.
   ```
   console.log("10" - 2);  // 8
   console.log("10" * 2);  // 20
   console.log("10" / 2);  // 5
   console.log("10" % 3);  // 1
   ```
3. **If conversion fails**:
   If the string cannot be converted to a number, the result is NaN (Not a Number):
   ```
   console.log("abc" - 2); // NaN
   ```

Concatenation Operator (+):

- The **+ operator is special** because it can do **addition** or **string concatenation**, depending on the operands.
- **Rules:**
  1. If **both operands are numbers**, it performs **addition**.
  2. If **either operand is a string**, it converts the other operand to a string and **concatenates**.

Examples:
```
console.log(5 + 10);      // 15→ numbers added
console.log("5" + 10);    // "510" → number converted to string, concatenated
console.log(5 + "10");    // "510" → same as above
console.log("Hello " + "World"); // "Hello World"
```

Behind the Scenes in the Browser

When you run JS in a browser:
1. The browser uses a **JavaScript engine** (like V8 in Chrome) to execute the code.
2. For each operator:
   - The engine checks the **data type** of the operands.
   - **If + is used:**
     - If either operand is a string → perform **concatenation**.
     - Otherwise → perform **numeric addition**.
   - **If other arithmetic operators are used (-, *, /, %)** → convert operands to numbers and perform the operation.

---

6) Programs on if, if else, else if, switch, looping conditions and execute 7 programs on each condition in javasrcipt(ex: if condition 7 programs).

Ans:- **1. if Statement:**

Program 1: Check if a number is positive
```
let num = 10;
if (num > 0) {
  console.log(num + " is positive");
}
```

Program 2: Check if a number is even
```
let number = 8;
if (number % 2 === 0) {
  console.log(number + " is even");
}
```

Program 3: Check if age is eligible to vote
```
let age = 20;
```

```
if (age >= 18) {
   console.log("You are eligible to vote");
}
```
Program 4: Check if a string is empty
```
let text = "";
if (text === "") {
   console.log("String is empty");
}
```

Program 5: Check if a number is divisible by 5
```
let n = 25;
if (n % 5 === 0) {
   console.log(n + " is divisible by 5");
}
```

Program 6: Check if a student passed (marks > 40)
```
let marks = 45;
if (marks > 40) {
   console.log("Student passed");
}
```

Program 7: Check if a character is a vowel
```
let char = 'a';
if ('aeiou'.includes(char.toLowerCase())) {
   console.log(char + " is a vowel");
}
```

2. <u>IF-ELSE Statement:</u>

Program 1: Check if number is positive or negative
```
let num = -5;
if (num > 0) {
   console.log("Positive");
} else {
   console.log("Negative");
}
```
Program 2: Check if number is even or odd
```
let n = 7;
if (n % 2 === 0) {
   console.log("Even");
} else {
   console.log("Odd");
}
```
Program 3: Check if a student passed or failed
```
let marks = 35;
if (marks >= 40) {
   console.log("Pass");
} else {
   console.log("Fail");
}
```
Program 4: Check if a number is divisible by 3
```
let number = 9;
if (number % 3 === 0) {
   console.log(number + " is divisible by 3");
} else {
   console.log(number + " is not divisible by 3");
}
```

Program 5: Check if a string is empty or not

```
let text = "Hello";
if (text === "") {
   console.log("Empty string");
} else {
   console.log("Not empty");
}
```

Program 6: Check if age is adult or minor

```
let age = 16;
if (age >= 18) {
   console.log("Adult");
} else {
   console.log("Minor");
}
```

Program 7: Check if number is divisible by 2 and 5

```
let x = 20;
if (x % 2 === 0 && x % 5 === 0) {
   console.log(x + " is divisible by 2 and 5");
} else {
   console.log(x + " is not divisible by both 2 and 5");
}
```

<u>3. ELSE IF Statement:</u>

Program 1: Grade based on marks

```
let marks = 85;

if (marks >= 90) {

   console.log("Grade A");

} else if (marks >= 75) {

   console.log("Grade B");

} else if (marks >= 50) {

   console.log("Grade C");

} else {

   console.log("Fail");

}
```

Program 2: Find largest of 3 numbers

```
let a = 10, b = 20, c = 15;

if (a > b && a > c) {

   console.log(a + " is largest");

} else if (b > a && b > c) {

   console.log(b + " is largest");

} else {
    console.log(c + " is largest");
}
```

Program 3: Determine season by month

```
let month = 4;
if (month === 12 || month <= 2) {
    console.log("Winter");
} else if (month >= 3 && month <= 5) {
    console.log("Spring");
} else if (month >= 6 && month <= 8) {
    console.log("Summer");
} else {
    console.log("Autumn");
}
```

Program 4: Check temperature

```
let temp = 30;
if (temp > 40) {
    console.log("Very hot");
} else if (temp > 30) {
    console.log("Hot");
} else if (temp > 20) {
    console.log("Warm");
} else {
    console.log("Cold");
}
```

Program 5: Day type based on day number

```
let day = 6;
if (day >= 1 && day <= 5) {
    console.log("Weekday");
} else if (day === 6 || day === 7) {
    console.log("Weekend");
} else {
    console.log("Invalid day");
}
```

Program 6: Check sign of number

```
let num = 0;
```

```
if (num > 0) {
    console.log("Positive");
} else if (num < 0) {
    console.log("Negative");
} else {
    console.log("Zero");
}
```

Program 7: Check marks grade

```
let marks = 72;
if (marks >= 90) {
    console.log("Excellent");
} else if (marks >= 75) {
    console.log("Very Good");
} else if (marks >= 60) {
    console.log("Good");
} else if (marks >= 50) {
    console.log("Average");
} else {
    console.log("Poor");
}
```

4. <u>SWITCH Statement:</u>

Program 1: Day of the week

```
let day = 3;
switch(day) {
    case 1: console.log("Monday"); break;
    case 2: console.log("Tuesday"); break;
    case 3: console.log("Wednesday"); break;
    case 4: console.log("Thursday"); break;
    case 5: console.log("Friday"); break;
    case 6: console.log("Saturday"); break;
    case 7: console.log("Sunday"); break;
    default: console.log("Invalid day");
}
```

Program 2: Month name

```
let month = 5;
```

```
switch(month) {
    case 1: console.log("January"); break;
    case 2: console.log("February"); break;
    case 3: console.log("March"); break;
    case 4: console.log("April"); break;
    case 5: console.log("May"); break;
    default: console.log("Other month");
}
```

Program 3: Simple calculator

```
let a = 10, b = 5, operator = '+';
switch(operator) {
    case '+': console.log(a + b); break;
    case '-': console.log(a - b); break;
    case '*': console.log(a * b); break;
    case '/': console.log(a / b); break;
    default: console.log("Invalid operator");
}
```

Program 4: Traffic light

```
let color = "red";
switch(color) {
    case "red": console.log("Stop"); break;
    case "yellow": console.log("Get Ready"); break;
    case "green": console.log("Go"); break;
    default: console.log("Invalid color");
}
```

Program 5: Weekend check

```
let dayNumber = 7;
switch(dayNumber) {
    case 6:
    case 7: console.log("Weekend"); break;
    default: console.log("Weekday");
}
```

Program 6: Fruit price

```
let fruit = "apple";
```

```
switch(fruit) {
    case "apple": console.log("100 per kg"); break;
    case "banana": console.log("50 per kg"); break;
    case "mango": console.log("150 per kg"); break;
    default: console.log("Not available");
}
```

Program 7: Grade system

```
let grade = 'B';
switch(grade) {
    case 'A': console.log("Excellent"); break;
    case 'B': console.log("Very Good"); break;
    case 'C': console.log("Good"); break;
    case 'D': console.log("Average"); break;
    case 'F': console.log("Fail"); break;
    default: console.log("Invalid grade");
}
```

<u>5. Looping:</u>

Program 1: Print numbers 1 to 5 (for loop)

```
for(let i = 1; i <= 5; i++) {
    console.log(i);
}
```

Program 2: Print even numbers up to 10

```
for(let i = 2; i <= 10; i += 2) {
    console.log(i);
}
```

Program 3: Sum of first 5 numbers

```
let sum = 0;
for(let i = 1; i <= 5; i++) {
    sum += i;
}
console.log("Sum =", sum);
```

Program 4: Print array elements

```
let arr = [10, 20, 30];
for(let i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

Program 5: Print numbers 1 to 5 using while loop

```
let i = 1;
while(i <= 5) {
   console.log(i);
   i++;
}
```

Program 6: Do-while loop example

```
let j = 1;
do {
   console.log(j);
   j++;
} while(j <= 5);
```

Program 7: Print multiplication table of 5

```
for(let i = 1; i <= 10; i++) {
   console.log("5 x " + i + " = " + (5*i));
}
```

---

7) Explain How web works behind the screen?

Ans:- **<u>1. You type a web address (URL):</u>**
Example: www.example.com
Your browser goes:
"Cool, but… where *is* that?"

<u>2. DNS: the internet's phone book</u>:
Computers don't understand names like example.com. They use IP addresses (numbers like 93.184.216.34).
So your browser asks a DNS server:
"Hey, what number belongs to example.com?"
DNS replies with the IP address.

<u>3. Your browser knocks on the server's door:-</u>
Now that it knows the address, your browser sends a request over the internet using HTTP or HTTPS:
"Hi server, can I get the homepage please?"
This request travels through:

- your Wi-Fi / mobile network
- your ISP
- multiple routers (internet traffic cops)
- until it reaches the server hosting the site

All of this usually takes milliseconds.

<u>4. The server responds:</u>
The server says:
"Sure!"
And sends back files like:

- HTML – the structure (headings, paragraphs, buttons)
- CSS – the styling (colors, fonts, layout)
- JavaScript – the behavior (clicks, animations, logic)
- images, videos, etc.

5. Your browser builds the page:
Your browser:
1. Reads the HTML top to bottom
2. Applies CSS to make it look good
3. Runs JavaScript to make it interactive

This process is called rendering.
That’s when you *see* the page.

6. JavaScript keeps talking in the background :
Even after the page loads, JavaScript can:
- fetch new data (scrolling, live updates)
- send form data
- talk to APIs
- update the page without reloading

That’s how apps like Instagram, Gmail, and YouTube feel “alive”.

7. Security happens the whole time :
If it’s HTTPS:
- Data is encrypted
- Your browser verifies the server’s identity
- Hackers see scrambled nonsense instead of your info

---

8)Explain CDN?

Ans: A CDN is a network of servers spread across the world that deliver website content from the location closest to the user.
Goal:
→ Faster load times
→ Less load on the main server
→ More reliability

What does a CDN actually serve?
Mostly **static stuff** (things that don’t change per user):
- images
- videos
- CSS files
- JavaScript files
- fonts
- sometimes whole HTML pages

These are perfect for copying everywhere.

What happens behind the scenes?
1. You request a page
“I want example.com”

2. DNS sends you to the nearest CDN location
Instead of pointing directly to the main server, DNS says:
“Go to the closest branch kitchen.”

This is called an **edge server**.

3. Cache check :
The edge server asks:
“Do I already have this file?”

- **Yes** → instant response ⚡
- **No** → fetch from main server, save it, then serve you

This is called **caching**.

4. You get the content FAST
Less distance =

- lower latency
- faster load
- smoother scrolling

Your browser doesn’t know or care — it just feels faster.

CDN ≠ backend replacement
Important distinction:

- **CDN** → fast delivery of files
- **Backend server** → logic, databases, user data

Think:

- CDN = reheating & plating
- Backend = actually cooking

Extra superpowers CDNs have:
Most CDNs also:

- block DDoS attacks
- rate-limit bots
- compress files
- handle HTTPS certificates
- cache API responses (sometimes)

---

9) Explain what is the smallest js file which you can execute in web?
Ans: The smallest JavaScript file you can execute is an *empty file* (0 bytes). This is valid because JavaScript engines (like those in browsers or Node.js) treat an empty script as a no-op—it runs without errors or output, effectively doing nothing. It's executable in any JavaScript runtime environment.

- In a browser: Include it via <script src="empty.js"></script>; the page loads without issues.
- In Node.js: Run node empty.js from the command line; it exits successfully with no output.
- Why it works: JavaScript parsers accept empty input as valid code, per the ECMAScript specification (e.g., a script can be an empty sequence of statements).

-If you meant the smallest file with actual code (e.g., producing output), the minimal example is a single character: ; (a semicolon, 1 byte). This is an empty statement that executes without error.

-For something functional, like printing "Hello, World!", the smallest is: console.log("Hello, World!") (about 27 bytes). But the truly smallest executable is the empty file.

---

10) Explore map(),for each(),filter(),sort(),render?
Ans: **1) map() — transform every item:**
What it does

- Runs a function on each element

- Returns a new array
- Same length as original

Example

```
const names = users.map(user => user.name);
```

Result:

```
["Alice", "Bob", "Charlie"]
```

Use when

- You want to change shape of data
- You want a new array
- Don’t use just to loop

2) forEach() — do something, return nothing

What it does

- Loops over array
- Returns undefined
- Used for side effects (logging, DOM updates)

Example

```
users.forEach(user => {
  console.log(user.name);
});
```

Key limitation:

```
const result = users.forEach(u => u.name);
```

Result === undefined

- No break
- No return of values

3) filter() — keep some items, drop others

What it does

- Runs a condition
- Returns a new array
- Length ≤ original

Example

```
const adults = users.filter(user => user.age >= 18);
```

Result:

```
[
  { name: "Alice", age: 28, active: true },
  { name: "Charlie", age: 35, active: true }
]
```

Use when

- Removing items
- Applying rules

4) Sort() — reorders the array (MUTATES!)

What it does

- Sorts elements in place
- Converts values to strings by default

Example (WRONG for numbers)

```
[10, 2, 5].sort();
// ["10", "2", "5"]
```

Correct numeric sort

```
const ages = [...users].sort((a, b) => a.age - b.age);
```

Why spread?

Because:sort() mutates the original array

5)Render() — not JS built-in, but a pattern

What “render” means:
Take data → turn it into UI
Usually involves map().

Example: render list in HTML

```
const ul = document.querySelector("ul");

function renderUsers(list) {
  ul.innerHTML = list
    .map(user => `<li>${user.name} (${user.age})</li>`)
    .join("");
}

renderUsers(users);
```

Flow:
DATA → map → HTML → DOM

---

11) Complete DOM manipulation?

ANS:
DOM manipulation is using **JavaScript to read, change, add, or remove parts of a web page after it has loaded.**
The browser turns your HTML into a structure called the **DOM (Document Object Model)**.
JavaScript talks to that structure like normal objects.

<u>What is the DOM (in simple terms)</u>
HTML:

```
<h1>Hello</h1>
<button>Click</button>
```

DOM (conceptually):

```
document
 └── body
     ├── h1
     └── button
```

Each tag becomes a **node (object)** you can access in JS.

<u>Why DOM manipulation exists</u>
Without DOM manipulation:

- Web pages would be static
- No interaction
- No updates without reload

With DOM manipulation:

- Buttons react
- Content updates
- Forms validate
- Data renders dynamically

Core things you can do with DOM manipulation
<u>1) Select elements:</u>

```
document.getElementById("title");
document.querySelector(".box");
document.querySelectorAll("li");
```

<u>2) Read or change content</u>

```
el.textContent = "New text";
el.innerHTML = "<strong>Bold</strong>";
```

3) Change styles or classes

```
el.style.color = "red";
el.classList.add("active");
el.classList.remove("hidden");
```

4) Create and remove elements

```
const div = document.createElement("div");
div.textContent = "Hello";
parent.append(div);
div.remove();
```

5) Handle user interaction (events)

```
button.addEventListener("click", () => {
  console.log("Clicked!");
});
```

Example: simple DOM manipulation

HTML:

```
<p id="msg">Hello</p>
<button id="btn">Change</button>
```

JS:

```
const p = document.getElementById("msg");
const btn = document.getElementById("btn");
btn.addEventListener("click", () => {
  p.textContent = "Hello DOM!";
  p.classList.toggle("highlight");
});
```

Result:

- Click button → text changes
- Style updates instantly
- No page reload

How DOM manipulation works internally (high level)

1. Browser loads HTML
2. Builds the DOM tree
3. JavaScript accesses nodes
4. Changes trigger **reflow & repaint**
5. Browser updates the screen

12) Explore npm and npx?

Ans:- 1. npm (Node Package Manager)

npm is the default package manager for **Node.js**, used to install, manage, and share JavaScript libraries (packages).

Key Points:

- **Install packages** (locally or globally)
- **Manage dependencies** in your package.json
- **Run scripts** defined in package.json
- Comes installed with Node.js

Common Commands:

a) Initialize a project

```
npm init
```

```
# or
npm init -y  # skips questions, uses defaults
```

b) Install a package locally

```
npm install lodash
```

- Installs the package in node_modules/
- Adds it to dependencies in package.json (by default)

c) **Install a package globally**

```
npm install -g create-react-app
```

- Makes the package available system-wide (CLI tools are a common example)

d) Update packages

```
npm update
```

e) Uninstall packages

```
npm uninstall lodash
```

f) Run scripts

```
// package.json
{
  "scripts": {
    "start": "node index.js",
    "test": "jest"
  }
}
```

```
npm run start
npm run test
```

2. npx (Node Package Executor)

npx comes bundled with npm (v5.2+). Its main job is **to execute Node packages without needing to install them globally**. It's perfect for running one-off commands.

Key Points:

- Runs CLI tools **temporarily** (without permanent installation)
- Can run **local packages** in node_modules/.bin
- Useful for trying packages quickly

Common Uses:

a) Run a package without installing

```
npx cowsay "Hello World!"
```

- Installs cowsay temporarily, runs it, then discards it.

b) Run a specific version of a package

```
npx create-react-app@5.0.0 my-app
```

- Ensures you’re using that version without globally changing anything

c) Run local binaries

If a package is installed locally:

```
npm install eslint
```

```
npx eslint .
```

- You don’t need a global install; npx finds it in node_modules/.bin.

d) Scaffolding projects

```
npx create-next-app my-next-app
```

- Automatically downloads and runs the CLI to scaffold a new Next.js project.

---

13) **Execute do while 7 program, math object, function structure execution?**

Ans:- Program 1: do...while loop – Print numbers 1 to 5

```
let i = 1;
do {
   console.log(i);
   i++;
} while (i <= 5);
```

Output:

```
1
2
3
4
5
```

Program 2: do...while loop – Sum of first 5 numbers

```
let sum = 0;
let j = 1;
do {
   sum += j;
   j++;
} while (j <= 5);
console.log("Sum of first 5 numbers is:", sum);
```

Output:

```
Sum of first 5 numbers is: 15
```

Program 3: Math object – Generate random number between 1 and 100

```
let randomNum = Math.floor(Math.random() * 100) + 1;
console.log("Random number between 1 and 100:", randomNum);
```

Output:

```
Random number between 1 and 100: 42  // (example)
```

Program 4: Math object – Find square root

```
let num = 49;
let sqrt = Math.sqrt(num);
console.log("Square root of", num, "is", sqrt);
```

Output:

```
Square root of 49 is 7
```

Program 5: Function declaration – Add two numbers

```
function add(a, b) {
   return a + b;
}
console.log("Sum of 8 and 12 is:", add(8, 12));
```
Output:
```
Sum of 8 and 12 is: 20
```

Program 6: Function expression – Check if number is even
```
const isEven = function(n) {
   return n % 2 === 0;
};
console.log("Is 7 even?", isEven(7));
console.log("Is 10 even?", isEven(10));
```
Output:
```
Is 7 even? false
Is 10 even? true
```

Program 7: Arrow function – Factorial of a number
```
const factorial = (n) => {
   let result = 1;
   for (let i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
};
console.log("Factorial of 5 is:", factorial(5));
```
Output:
```
Factorial of 5 is: 120
```

---

14) Create an array in all the ways, work them throw all the loops.

Ans:

- **Ways to create an array**

1. Array literal (MOST COMMON)

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

2. Array constructor

```
const arr2 = new Array(1, 2, 3, 4, 5);
```

Be careful:

```
const arrWeird = new Array(5); // creates empty array with length 5
```

3. Empty array, then push

```
const arr3 = [];

arr3.push(1);

arr3.push(2);

arr3.push(3);
```

4. Array.from()

```
const arr4 = Array.from([1, 2, 3, 4, 5]);
```

Or generate values:

```
const arr4b = Array.from({ length: 5 }, (_, i) => i + 1);
// [1, 2, 3, 4, 5]
```

5. Spread operator

```
const base = [1, 2, 3];
const arr5 = [...base, 4, 5];
```

6. Using split (from string)

```
const arr6 = "1,2,3,4,5".split(",");
```

- **<u>Looping through arrays (ALL common loops)</u>**

Let's use:

```
const arr = [1, 2, 3, 4, 5];
```

1. Classic for loop

Best when you need index control

```
for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

2. for...of

Clean, readable, value-focused

```
for (const value of arr) {
  console.log(value);
}
```

3. for...in

Loops over **indexes**, not values

```
for (const index in arr) {
  console.log(arr[index]);
}
```

4. forEach()

Functional, clean, no break allowed

```
arr.forEach((value, index) => {
  console.log(index, value);
});
```

5. map() (transforms array)

```
const doubled = arr.map(value => value * 2);
console.log(doubled);
```

6. filter() (keeps matching items)

```
const evens = arr.filter(value => value % 2 === 0);
console.log(evens);
```

7. reduce() (boils array into one value)

```
const sum = arr.reduce((total, value) => total + value, 0);
console.log(sum);
```

8. while loop

```
let i = 0;
while (i < arr.length) {
  console.log(arr[i]);
  i++;
}
```

9. do...while

```
let i = 0;
do {
  console.log(arr[i]);
  i++;
} while (i < arr.length);
```
