Explain reduce function

explain reduce function

Here is the **most beginner-friendly explanation** of `reduce()` — exactly in your style, Vipul.
I'll explain *what it is*, *how it works*, *why accumulator becomes undefined*, with **very simple examples**.

✅ What is `reduce()` in JavaScript?

`reduce()` is an array method that **reduces** an array to **a single value**.

That final value can be:

✔️ number
✔️ string
✔️ array
✔️ object
✔️ anything (depends on your logic)

✨ Reduce Syntax

```js
array.reduce((accumulator, currentValue, index, array) => {
  // logic
}, initialValue);
```

Meaning of arguments:

Name	Meaning
accumulator (acc)	The "result" you are building step-by-step
currentValue (cur)	Current item of array
index	Current index
array	Entire array
initialValue	Starting value of accumulator

🔥 Very Simple Example (Sum of numbers)

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

const sum = arr.reduce((acc, cur) => {
  return acc + cur;
}, 0);

console.log(sum);
```

Step-by-step:

Step	acc	cur
1	0	1
2	1	2
3	3	3
4	6	4
final	**10**	—

🍭 Example 2: Count how many times a word appears

```
js

const fruits = ["apple", "banana", "apple", "orange"];

const count = fruits.reduce((acc, cur) => {
  acc[cur] = (acc[cur] || 0) + 1;
  return acc;
}, {});

console.log(count);
```

Output:

```
js

{ apple: 2, banana: 1, orange: 1 }
```

🧠 Why you got the ERROR: `Cannot read properties of undefined (push)`

Your code:

```
js

acc.push(fullName);
```

This error comes because:

👉 You **did not give initialValue as `[]`**
So **acc starts as undefined** → you cannot do `.push()` on undefined.

Correct version:

```js
const users = [
  {firstName: "akshay", lastName: "saini", age: 26},
  {firstName: "donald", lastName: "trump", age: 75},
  {firstName: "elon", lastName: "musk", age: 50}
];

const result = users.reduce((acc, user) => {
  const fullName = user.firstName + " " + user.lastName;

  acc.push(fullName);   // now acc is []
  return acc;
}, []); // <--- MUST give empty array as initial value

console.log(result);
```

⭐ Example 3: Flatten array of arrays

```js
const arr = [[1,2], [3,4], [5,6]];

const flat = arr.reduce((acc, cur) => {
  return acc.concat(cur);
}, []);

console.log(flat);
```

Output:

```js
[1,2,3,4,5,6]
```

⭐ Example 4: Turn array into object

```js
const arr = ["a", "b", "c"];

const obj = arr.reduce((acc, cur, index) => {
  acc[index] = cur;
  return acc;
}, {});

console.log(obj);
```

Output:

```js
{0: "a", 1: "b", 2: "c"}
```

🥇 THE MAIN POINT

`reduce()` = You start with an **accumulator**, and you keep modifying it.

If accumulator should be:

- array → start with `[]`
- object → start with `{}`
- number → start with `0`
- string → start with `""`

If you forget initialValue → `acc` becomes the first item of array automatically.

If you want, I can also give:

✔️ 5 easy reduce questions
✔️ 5 medium
✔️ 10 hard
✔️ all with solutions + output

Just say **"give reduce questions"** 🔥

tell me what if when we don't passes initial value like here:
const arr = [10, 5, 30, 15];

const max = arr.reduce((acc, curr) => acc > curr ? acc : curr);
console.log(max);

Great question, Vipul — this is **exactly where reduce() behaves differently**.

Let me explain in **simple, beginner-friendly style**.

✅ What happens if we DON'T pass initial value?

When you do **NOT** pass an initial value, JavaScript does:

✔️ **`accumulator = first element of array`**

✔️ **`currentValue = second element`**

So reduce() starts from the **2nd item**, not from the 1st.

🔥 Your example

```js
const arr = [10, 5, 30, 15];
const max = arr.reduce((acc, curr) => acc > curr ? acc : curr);
console.log(max);
```

Step-by-step:

Step	acc	curr	result
Start	**10** (first element)	—	—
1st iteration	10	5	10
2nd iteration	10	30	30
3rd iteration	30	15	30

Final result → **30**

🧠 Why does it work here?

Because for **max/min**, it makes sense to start with the first element.

You don't need to give initial value because:

- the first element (10) already gives a starting comparison value.

⚠️ BUT be careful!

Not passing initial value can cause errors in some cases:

❌ Empty array

```
js

[].reduce(...)
```

👉 Error: **Reduce of empty array with no initial value**

❌ When accumulator must be array or object

Example:

```
js

arr.reduce((acc, item) => acc.push(item));
```

Without initial `[]`, `acc` becomes a **number**, **string**, or **object**, and `.push()` fails.

✔️ Final summary

Case	Without initial value	With initial value
max/min	Works	Works

Case	Without initial value	With initial value
sum	Works	Works
building array	❌ Fails (acc is not [])	✔️ Works
building object	❌ Fails	✔️ Works
empty array	❌ Fails	✔️ Works

If you want, I can also create a **table+diagram** for how reduce behaves with and without initial value.