Functional programming

Functional Programming is a programming paradigm where functions are first-class citizens, and the focus is on pure functions, immutability, and function composition rather than shared state and side-effects.

1. Pure Functions
2. Immutability
3. Declarative
4. Avoid Shared state
5. Avoid Side Effects
6. Reuse or Compose Logic
7. Don’t Iterate
8. Loose coupling
9. First-Class & Higher-Order Functions

1. Pure Functions

A function is pure if:

- ❑ It returns the same output for the same input.
- ❑ It doesn’t cause side effects (like modifying external variables or DOM).

```
// ❌Impure ****
let count = 0;
function increment() {
  count++; // modifies external state
}

// ✅Pure ****
function add(a, b) {
  return a + b;
}
```

2. Immutability

Do not modify existing data. Instead, return new copies.

```
// ❌Mutable *****
const user = { name: "Alice" };
user.age = 25; // directly modifies original object

// ✅Immutable *****
const updatedUser = { ...user, age: 25 };
```

3. Declarative Code

Describe what should be done, not how.

```
// ❌Imperative ****
let doubled = [];
for (let i = 0; i < numbers.length; i++) {
  doubled.push(numbers[i] * 2);
}

// ✅Declarative ****
const numbers = [1, 2, 3, 4];
const doubled = numbers.map(n => n * 2);
```

4. Avoid Shared State

Shared mutable state can lead to bugs, especially in async or parallel systems.

```
// ❌Shared State (Bad) ***
let total = 0;
function addToTotal(n) {
  total += n;
}

// ✅avoid shared state ***
function add(a, b) {
  return a + b;
}
```

5. Avoid Side Effect

Side effects are anything a function does outside its scope
(API call, DOM update, modifying global vars).

```
// ❌Side Effect ***
function logMessage(msg) {
  console.log(msg); // side effect: interacts with console
}

// ✅No Side Effect ***
function getGreeting(name) {
  return `Hello, ${name}`;
}
```

6. Reuse Or Compose Logic

Build small reusable functions and compose them together.

```
const toLower = str => str.toLowerCase();
const removeSpaces = str => str.replaceAll(' ', '');
const atTheRate = str => '@' + str;

let str = "Manas Kumar Lal";
let result = atTheRate(removeSpaces(toLower(str)))
console.log(result);
```

7. Don’t Iterate (Imperatively)

Avoid for, while, etc. Use map, filter, reduce.

```
// ❌Imperative Style ***
let evens = [];
for (let n of [1, 2, 3, 4]) {
  if (n % 2 === 0) evens.push(n);
}

// ✅FP (Declarative) Style ***
const evens = [1, 2, 3, 4].filter(n => n % 2 === 0);
```

8. Loose Coupling

Coupling refers to how dependent one piece of code is on another.

Loose coupling means less dependent, Keep functions and modules independent.

```
// ❌Tightly Coupled ***
function getUserData() {
  return fetch("https://api.example.com/user").then(res => res.json());
}

// ✅Loosely Coupled ***
function getData(api) {
  return fetch(api).then(res => res.json());
}
```

9. First-Class & Higher-Order Functions

- First-Class: Functions can be stored in variables, passed, and returned.
- Higher-Order: Functions that take other functions as arguments or return them.

```
const greet = () => "Hello";
const callWithName = fn => name => `${fn()} ${name}`;

const greetUser = callWithName(greet);
console.log(greetUser("School4U")); // Hello School4U
```

Note:

- All callbacks are first-class functions, but not all first-class functions are callbacks.
- All higher order functions are first class functions but not all first-class functions are higher order functions

1. What is a pure function, and why is it useful in UI rendering?

2. How would you use .map() to transform a list of products into a list of HTML elements?

3. How do you use .reduce() to calculate the total price in a shopping cart?

4. Explain immutability and how you would update an object in an array without mutating the original.

5. How would you compose multiple functions to transform data step-by-step (e.g., sanitize → trim → capitalize)? Scenario: You're preparing user input before storing it. (Expected knowledge: Function composition, chaining, pipe or compose logic.)

6. What is the difference between forEach and map, and when is it wrong to use map?

7. How do you implement your own version of .map() function on arrays?