

WEB APP DEVELOPMENT WITH REACTJS (INT252)

Lecture 3: Functional Programming Concepts for React

Unit I – JavaScript Refresher & React Foundations

Syllabus Mapping:

- Functional programming basics
- Pure functions
- State change & re-render logic

Target Learner: Beginner (HTML, CSS, JavaScript knowledge)

Why Functional Programming Matters in React

React is built on **functions**, not classes (modern React).

React expects:

- Predictable output
- No hidden side effects
- Same input → same UI

What is Functional Programming?

Functional Programming (FP) means:

- Programs built using functions
- Functions treated as values
- Avoid changing data directly

React follows **FP principles**

Functions as Building Blocks

In React:

- Components are functions
- Hooks are functions
- UI = function(state)

React Mental Model

$UI = f(state, props)$

Change state → React recalculates UI

Pure Functions (Very Important)

A pure function:

- Has no side effects
- Does not modify external data
- Same input → same output

Pure Function Example

```
function add(a, b) {  
  return a + b;  
}
```

Predictable 

Impure Function Example

```
let total = 0;  
function addToTotal(x) {  
  total += x;  
}
```

Unpredictable ✖

Why React Loves Pure Functions

- Easy to debug
- Easy to test
- Easy to re-render

React components should behave like pure functions

Component as a Pure Function

```
function Welcome({ name }) {  
  return <h1>Hello {name}</h1>;  
}
```

Same name → same UI

Side Effects (Preview)

Side effects include:

- API calls
- Timers
- DOM updates

React separates side effects using `useEffect`

Immutability & Pure Functions

Pure functions:

- Do not modify input
- Return new values

```
const newArr = arr.map(x => x * 2);
```

Referential Transparency (Simple)

```
getTotal(2,3)
```

Always returns 5

React relies on this predictability

Why Mutation Breaks React Logic

Same object reference



React thinks nothing changed

Correct React Data Flow

```
graph TD; A[User Action] --> B[State Update (New Object)]; B --> C[Re-render UI];
```

User Action
↓
State Update (New Object)
↓
Re-render UI

State Change Triggers Re-render

React checks:

```
oldState !== newState
```

Only then UI updates

Common Beginner Mistakes

- ✗ Writing impure components
- ✗ Mutating state directly
- ✗ Expecting React to auto-detect changes

Practice Exercises

1. Identify pure vs impure functions
2. Rewrite impure function as pure
3. Explain why mutation is bad in React

Answers – Practice Exercises

1. Pure Function

```
const multiply = (a, b) => a * b;
```

✓ Answers – Continued

2. Pure Rewrite

```
function addPure(total, x) {  
  return total + x;  
}
```

Answers – Continued

3. Explanation

Mutation keeps same reference.

React cannot detect changes.

Key Takeaways

- React follows functional programming
- Components should be pure
- State change = new data
- Predictability = reliable UI

Next Lecture

Lecture 4: SPA vs MPA & React Philosophy

Unit I – JavaScript Refresher & React Foundations