WAD END

Here is the **exam answer rewritten in short and crisp NOTES format**, perfect for quick revision or writing in an exam:

Q1 a) Simple Application in TypeScript using Modules

📝 **Notes Format**

➡️ **Definition:**

A **module** in TypeScript is a file containing functions, classes, variables, etc., that can be exported and imported into other files for reuse.

📁 **Step 1: mathUtils.ts** – Module File

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


export function subtract(a: number, b: number): number {
  return a - b;
}
```

📁 **Step 2: main.ts** – Main Application

```
import { add, subtract } from './mathUtils';


let x = 10, y = 5;
console.log("Addition:", add(x, y));
console.log("Subtraction:", subtract(x, y));
```

⚙️ **Compilation Steps:**

1. Install TypeScript: npm install -g typescript
2. Compile: tsc main.ts mathUtils.ts

3. Run: node main.js

✅ **Output:**

```
Addition: 15

Subtraction: 5
```

Q1 b) Pipe in Angular

📝 **Notes Format**

➡️ **Definition:**

A **pipe** in Angular is used to transform data in templates.

✅ Angular has **built-in pipes** like date, uppercase, currency, etc.

✅ You can also create **custom pipes**.

---

1) Built-in Pipe Example:

app.component.ts

```
export class AppComponent {
  today: Date = new Date();
}
```

app.component.html

```
<p>Today's Date: {{ today | date:'fullDate' }}</p>
```

---

2) Custom Pipe Example:

🔹 **exponential.pipe.ts**

```
import { Pipe, PipeTransform } from '@angular/core';


@Pipe({ name: 'exponential' })
export class ExponentialPipe implements PipeTransform {
  transform(value: number, exponent: number = 1): number {
    return Math.pow(value, exponent);
  }
}
```

🔹 **app.component.html**

```
<p>2 raised to 3 is: {{ 2 | exponential:3 }}</p>
```

🔹 **app.module.ts**

```
@NgModule({
 declarations: [AppComponent, ExponentialPipe],
 imports: [BrowserModule],
 bootstrap: [AppComponent]
})
export class AppModule {}
```

✅ **Summary:**

- Pipes are used for data transformation in Angular templates.
- Built-in pipes are ready to use.
- Custom pipes allow advanced formatting logic.

Let me know if you want this in **PDF**, **handwritten style**, or **translated to Marathi**!

Sure, here's the answer rewritten in **notes format** for quick and effective exam revision:

Q2 a) Angular Application with Multiple Components

1. Basic Layout Structure

- src/app/ – Main application folder.
- Example folders:
  - header/ → HeaderComponent
  - footer/ → FooterComponent
  - dashboard/ → DashboardComponent
- Main files:
  - app.component.ts → Root component logic

    - app.module.ts → Module to register all components

2. Creating a Component

- **Command**:
  ng generate component <name> or ng g c <name>
- **Generates**:
    - .ts → Logic
    - .html → Template
    - .css → Styles
    - .spec.ts → Test (optional)

3. Component Structure (Example)

```
@Component({
 selector: 'app-header',
 templateUrl: './header.component.html',
 styleUrls: ['./header.component.css']
})
export class HeaderComponent {
 title = 'My App';
}
```

4. Registering Component

- Inside app.module.ts:

```
@NgModule({
 declarations: [AppComponent, HeaderComponent, FooterComponent],
 imports: [BrowserModule],
 bootstrap: [AppComponent]
})
```

5. Using Components

- In app.component.html:

```
<app-header></app-header>
```

```
<router-outlet></router-outlet>

<app-footer></app-footer>
```

Q2 b) React JS – Basic Hooks

1. Common Built-in Hooks

- useState – Adds state in functional components
- useEffect – Handles side effects
- useContext, useReducer, useCallback, useMemo, useRef

---

2. useState Hook

- **Purpose**: Add state in functional components
- **Syntax**:

```
const [count, setCount] = useState(0);
```

- count → current value
- setCount → function to update state
- Re-renders component when state updates

---

3. useEffect Hook

- **Purpose**: Perform side effects (e.g., fetch, DOM updates)
- **Syntax**:

```
useEffect(() => {
  document.title = `Clicked ${count} times`;
  return () => {
    // Cleanup code
  };
}, [count]);
```

- Runs **after render**
- Dependency array [count] → runs when count changes

- Empty array [] → runs only once (like componentDidMount)

Let me know if you'd like diagram illustrations or code flow explanations too.

Here is a **well-organized NOTES FORMAT** answer for **Q3 (a & b)** — suitable for **writing in exams**:

Q3 a) What is CRUD? Explain CRUD using Node.js.

[9 Marks]

✅ Definition of CRUD:

CRUD stands for:

- **C** – Create
- **R** – Read
- **U** – Update
- **D** – Delete

It refers to the basic operations used in databases and web applications to manage data.

✅ CRUD in Node.js (using Express + JSON data)

📁 1. Setup

Install Express:

```
npm install express
```

📄 2. Sample Code (server.js)

```
const express = require('express');

const app = express();

app.use(express.json());


let users = [

 { id: 1, name: "Tanmay" },
 { id: 2, name: "Ravi" }
];

// Create (POST)
app.post('/users', (req, res) => {
 users.push(req.body);
 res.send("User added");
});

// Read (GET)
app.get('/users', (req, res) => {
 res.json(users);
});

// Update (PUT)
app.put('/users/:id', (req, res) => {
 const id = parseInt(req.params.id);
 const index = users.findIndex(u => u.id === id);
 if (index !== -1) {
  users[index] = req.body;
  res.send("User updated");
 } else {
  res.send("User not found");
 }
});

// Delete (DELETE)
app.delete('/users/:id', (req, res) => {
  users = users.filter(u => u.id !== parseInt(req.params.id));
  res.send("User deleted");
});

app.listen(3000, () => console.log("Server running on port 3000"));
```

✅ **Output Operations:**

- POST /users → Add user
- GET /users → Show all users
- PUT /users/1 → Update user with ID 1
- DELETE /users/1 → Delete user with ID 1

Q3 b) What is Node.js? Explain file handling in Node.js.

[8 Marks]

✅ **Definition of Node.js:**

- Node.js is an **open-source**, **cross-platform** JavaScript runtime.
- It runs JavaScript code **outside the browser**.
- Built on **Chrome's V8 engine**.
- Commonly used for **server-side programming**.

✅ **Features:**

- Non-blocking, asynchronous I/O
- Fast and scalable
- Supports modules
- Event-driven architecture

✅ **File Handling in Node.js (Using fs module)**

📌 Node.js provides a built-in module called **fs** (File System) to perform file operations.

📁 **Common File Operations:**

1. Create/Write File:

const fs = require('fs');

fs.writeFileSync('example.txt', 'Hello World!');

2. Read File:

const data = fs.readFileSync('example.txt', 'utf8');

console.log(data); // Output: Hello World!

3. Append to File:

fs.appendFileSync('example.txt', '\nWelcome to Node.js!');

4. Delete File:

fs.unlinkSync('example.txt');

✅ **Summary:**

- **CRUD** is used for data management operations.
- In **Node.js**, we use Express to build CRUD APIs.
- **File handling** in Node.js is done via the built-in fs module.

Let me know if you'd like this as a **PDF**, or translated to **Marathi/Hindi**, or formatted for your **college assignment/report**.

Q4 a) Express.js Routes Example

- **Setup**
- npm init -y
- npm install express
- **app.js**
- const express = require('express');

```
const app = express();
const port = 3000;

// Parse JSON bodies
app.use(express.json());

// 1. Root route
app.get('/', (req, res) => {
  res.send('Welcome to the Home Page');
});

// 2. Static route
app.get('/about', (req, res) => {
  res.send('About Us');
});

// 3. Route with URL parameter
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User Profile for ID: ${userId}`);
});

// 4. Route handling POST
app.post('/users', (req, res) => {
  const newUser = req.body;          // expects JSON { name, email }
  // (would normally save to DB here)
  res.status(201).json({
    message: 'User created',
    user: newUser
  });
});

// 5. Chained route handlers
app.route('/products')
  .get((req, res) => {
    res.send('List of Products');
  })
  .post((req, res) => {
    const product = req.body;
    res.status(201).json({ message: 'Product added', product });
  });

// 6. 404 handler (fallback)
app.use((req, res) => {
  res.status(404).send('Page Not Found');
});

// Start server
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});
```

- **Key Points**
    - app.get(), app.post(), etc. define routes for HTTP methods.
    - URL parameters via :paramName accessed in req.params.
    - app.use(express.json()) parses incoming JSON payloads.
    - Route chaining with app.route(path) groups handlers.

    - Middleware order matters; place 404 handler last.

Q4 b) Database Replication

Definition

- **Replication**: Copying and maintaining database objects (e.g., tables) in multiple database servers to ensure consistency and availability.

Types of Replication

- **Master–Slave** (Primary–Replica)
- **Multi-Master**
- **Snapshot**
- **Transactional**

Advantages

1. **High Availability**
    - Failover capability if primary node fails
2. **Load Balancing**
    - Distribute read queries across replicas to reduce latency
3. **Fault Tolerance & Durability**
    - Multiple copies protect against data loss
4. **Geographic Distribution**
    - Place replicas close to users globally for faster access
5. **Backup & Reporting**
    - Use replicas for backups and running analytic/reporting jobs without impacting primary
6. **Scalability**
    - Scale out read operations by adding replicas

Q5 a) Navigation in jQuery Mobile

- **Definition of Navigation**

    - Moving between “pages” (DOM sections) within a single HTML file or across files
    - Handled via Ajax loading and URL hash changes (#pageID)
    - Enhances mobile UX with smooth transitions

- **Key Concepts**

1. **Multi-page Templates**
    - Multiple <div data-role="page" id="..."> in one HTML
2. **Links & Buttons**
    - <a href="#page2">Go to Page 2</a> for internal navigation
    - <a href="other.html">External page</a> loads via Ajax
3. **JavaScript API**
    - $.mobile.changePage(target, options) to programmatically navigate
    - target can be a selector ("#page2") or URL

- **Example Code**
- <!DOCTYPE html>
- <html>
- <head>
- <title>jQuery Mobile Navigation</title>
- <link rel="stylesheet" href="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
- <script src="https://code.jquery.com/jquery-1.11.3.min.js"></script>
- <script src="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
- </head>
- <body>
-
- <!-- Page 1 -->
- <div data-role="page" id="page1">

```
  <div data-role="header"><h1>Home</h1></div>
  <div data-role="content">
    <p>Welcome to Page 1.</p>
    <!-- Link-based navigation -->
    <a href="#page2" data-role="button">Go to Page 2</a>
    <!-- JS-based navigation -->
    <button id="btnNav" data-role="button">JS Navigate to Page 2</button>
  </div>
</div>

<!-- Page 2 -->
<div data-role="page" id="page2">
  <div data-role="header"><h1>Page 2</h1></div>
  <div data-role="content">
    <p>You are on Page 2.</p>
    <a href="#page1" data-role="button">Back to Home</a>
  </div>
</div>

<script>
  // Programmatic navigation
  $('#btnNav').on('click', function() {
    $.mobile.changePage('#page2', {
      transition: 'slide',
      changeHash: true
    });
  });
</script>
```

-
- </body>
- </html>

Q5 b) jQuery Mobile & Layout Types

- **What is jQuery Mobile?**
  - A touch-optimized web framework for smartphones/tablets
  - Built on jQuery core; provides UI widgets, theming, and Ajax navigation
  - Uses HTML5 data-attributes for declarative UI
- **Layout in jQuery Mobile**
  - Defines how header, footer, and content regions are arranged
  - Uses <div> roles and CSS classes for responsive, grid-based layouts
- **Primary Layout Types**

1. **Header / Footer Bars**
    - <div data-role="header"> and <div data-role="footer">
    - Fixed or inline; can contain nav buttons, titles
2. **Content Block**
    - <div data-role="content"> holds main page content
3. **Grid Layouts**
    - <div class="ui-grid-a"> to <div class="ui-grid-d"> for 2–5 columns
    - Child blocks:
        - ui-block-a, ui-block-b, etc.
    - Example:
    - <div class="ui-grid-a">
    -   <div class="ui-block-a">Block A</div>
    -   <div class="ui-block-b">Block B</div>
    - </div>
4. **Panel Layouts**

        - Off-canvas panels with <div data-role="panel">
        - Slides in from left/right; used for menus or filters

5. **Listview Layout**
        - <ul data-role="listview"> for menus, lists, navbars

- **Notes on Theming & Responsiveness**
    - Swatches (a, b, c, etc.) for colors: data-theme="b"
    - Automatic enhancement of form elements and buttons
    - Responsive behavior handled via CSS media queries

Here's a **concise exam-style answer** for **Q6 a)** in notes format:

Q6 a) Why CSS is important in designing mobile websites?

[5 Marks]

Why CSS is Important for Mobile Website Design:

- **Responsive Design:** CSS allows websites to adjust layout and content based on screen size (e.g., mobiles, tablets, desktops).
- **Better User Experience:** Improves readability, navigation, and usability on small screens.
- **Faster Loading:** CSS can optimize styling without extra images or heavy code, improving page speed on mobile.
- **Flexible Layouts:** Enables flexible grids, flexible images, and media queries to make sites adapt seamlessly.
- **Consistency:** Maintains consistent look and feel across different devices.

Example of CSS class for Mobile Website Design:

```
/* Example: Responsive container with padding for mobile screens */

.mobile-container {

 width: 100%;
```

```
  padding: 10px;
  box-sizing: border-box;
}

/* Media query to target devices with max width 600px (mobiles) */
@media only screen and (max-width: 600px) {
  .mobile-container {
    background-color: #f2f2f2;
    font-size: 16px;
  }
}
```

Explanation:

- .mobile-container ensures content fits the screen width on mobiles.
- The @media query applies styles only when screen width is 600px or less (typical mobile size).

If you want, I can also give a brief answer for other parts or write it in paragraph form!

Here's a **clear, exam-style answer** for **Q6 b)** in notes format:

Q6 b) List any five widgets in jQuery Mobile and explain any two briefly.

[5 Marks]

Five jQuery Mobile Widgets:

1. **Button**
2. **Listview**
3. **Collapsible**
4. **Popup**

5. **Slider**

Brief Explanation of Any Two:

1) Button Widget:

- Converts standard HTML buttons or links into styled, touch-friendly buttons.
- Provides visual feedback and themes suitable for mobile devices.
- Easy to customize with icons and text.

Example:

```
<a href="#" data-role="button">Click Me</a>
```

---

2) Listview Widget:

- Creates a stylized, scrollable list of items.
- Supports nested lists, dividers, icons, and filtering.
- Enhances usability for menus or content lists on mobile.

Example:

```
<ul data-role="listview">
  <li>Home</li>
  <li>About</li>
  <li>Contact</li>
</ul>
```

Let me know if you want detailed examples or explanation for the other widgets!

Q7 a) What is EC2? How to Deploy a Website on EC2

What is EC2 (Elastic Compute Cloud)?

- Amazon EC2 is a cloud service that provides **resizable virtual servers** (instances) to run applications.
- It offers full control over the OS and server, similar to having a physical machine.

- Supports Windows & Linux instances.

Steps to Deploy a Website on EC2

1. **Login to AWS Console**
    - Visit https://aws.amazon.com
    - Sign in to AWS Management Console
2. **Launch EC2 Instance**
    - Go to **EC2 Dashboard** → Click **Launch Instance**
    - Choose **Amazon Machine Image (AMI)** (e.g., Ubuntu or Amazon Linux)
    - Select **Instance Type** (e.g., t2.micro – Free Tier eligible)
    - Configure **Key Pair** for SSH access
3. **Configure Security Group**
    - Allow **Inbound Rules**:
        - **HTTP (Port 80)** – for website access
        - **SSH (Port 22)** – for terminal access
        - Optional: **HTTPS (Port 443)**
4. **Connect to EC2 Instance**
    - Use terminal or SSH client:
    - ssh -i "your-key.pem" ec2-user@<Public-IP>
5. **Install Web Server**
    - For Apache (Amazon Linux):
    - sudo yum update -y
    - sudo yum install httpd -y
    - sudo systemctl start httpd
    - sudo systemctl enable httpd
    - For Ubuntu:
    - sudo apt update
    - sudo apt install apache2 -y
    - sudo systemctl start apache2

    - sudo systemctl enable apache2

6. **Deploy Website Files**
    - Copy website files to:
        - Amazon Linux: /var/www/html/
        - Ubuntu: /var/www/html/
    - Use scp or upload via FileZilla (SFTP)

7. **Access Website**
    - Open browser and visit:
    - http://<Public-IP>

Q7 b) What is AWS Cloud & Its Services

What is AWS Cloud?

- AWS (Amazon Web Services) is a **cloud computing platform** by Amazon.
- Offers **on-demand** IT resources like servers, storage, databases, and AI tools via the internet.
- Based on **pay-as-you-go** pricing.

Major AWS Services Categories

- **Compute** → EC2, Lambda, Elastic Beanstalk
- **Storage** → S3, EBS, Glacier
- **Database** → RDS, DynamoDB, Aurora
- **Networking** → VPC, CloudFront, Route 53
- **Security** → IAM, KMS, Shield
- **Developer Tools** → CodeBuild, CodeDeploy
- **Analytics & AI** → SageMaker, Athena, Rekognition

Explanation of Two AWS Services

1. **Amazon S3 (Simple Storage Service)**
    - Object storage for files, backups, media, logs, etc.

    - Unlimited scalability, 99.999999999% durability.
    - Files are stored as **objects** in **buckets**.
    - Supports static website hosting.
2. **Amazon RDS (Relational Database Service)**
    - Managed SQL database service (MySQL, PostgreSQL, etc.)
    - Automates backups, patching, and scaling.
    - High availability with Multi-AZ deployments.

Here's a concise, exam-ready **notes format answer** for **Q8 a & b**:

Q8 a) What is ELB? What are the ELB types? List advantages of ELB

[9 Marks]

What is ELB?

- **ELB (Elastic Load Balancer)** is a service provided by cloud platforms (like AWS) to distribute incoming network traffic across multiple servers (instances).
- It helps improve application availability, fault tolerance, and scalability.

Types of ELB:

1. **Application Load Balancer (ALB):**
    - Operates at **Layer 7 (HTTP/HTTPS)**
    - Supports advanced routing (path-based, host-based)
2. **Network Load Balancer (NLB):**
    - Operates at **Layer 4 (TCP/UDP)**
    - Handles millions of requests with ultra-low latency
3. **Classic Load Balancer (CLB):**
    - Operates at both Layer 4 and Layer 7
    - Legacy option, now mostly replaced by ALB and NLB

Advantages of ELB:

- **Improves fault tolerance:** Routes traffic only to healthy instances
- **Scalability:** Automatically distributes traffic to handle load changes
- **High availability:** Ensures continuous service by balancing requests
- **Security:** Supports SSL termination and integration with security groups
- **Better resource utilization:** Avoids overload on single servers

---

Q8 b) What is VPC? What are the components of VPC?

[6 Marks]

What is VPC?

- **VPC (Virtual Private Cloud)** is a logically isolated network within a cloud provider's infrastructure where you can launch resources securely.
- It gives you control over your virtual network environment.

Components of VPC:

1. **Subnets:** Segments within a VPC for grouping resources (public/private)
2. **Route Tables:** Define traffic routing rules between subnets and gateways
3. **Internet Gateway (IGW):** Allows communication between VPC and the internet
4. **NAT Gateway:** Enables private subnet instances to access the internet securely
5. **Security Groups:** Virtual firewalls controlling inbound/outbound traffic
6. **Network ACLs (Access Control Lists):** Optional layer of stateless traffic filtering
7. **Elastic IP Addresses:** Static public IPs for resources in the VPC

Let me know if you want me to explain any point with diagrams or examples!

Here's an **exam-ready answer in notes format** for your **Q1** parts a, b, and c:

Q1 a) Explain MVC Architecture with Diagram

[6 Marks]

MVC Architecture:

- **M** – Model
  - Represents data and business logic.
  - Handles data storage, retrieval, and rules.
- **V** – View
  - Displays data to the user (UI).
  - Reflects changes in the Model.
- **C** – Controller
  - Acts as an intermediary between Model and View.
  - Handles user inputs and updates Model/View accordingly.

Working:

1. User interacts with **View**.
2. **Controller** receives input, processes it.
3. Controller updates the **Model**.
4. **Model** notifies the **View** about data changes.
5. View updates the UI.

Diagram:

```
User <---> View <---> Controller <---> Model

     ↑            ↓

   Updates      Data Storage
```

Q1 b) List and Explain Different Types of Structural Directives in Angular

[6 Marks]

Structural Directives change the DOM layout by adding/removing elements.

---

1. ***ngIf**
    - Conditionally includes a template based on a boolean expression.
    - Example:
    - <div *ngIf="isLoggedIn">Welcome User</div>
2. ***ngFor**
    - Loops over a collection and renders a template for each item.
    - Example:
    - <li *ngFor="let item of items">{{ item }}</li>
3. ***ngSwitch**
    - Conditionally switches between alternative views.
    - Works with ngSwitchCase and ngSwitchDefault.
    - Example:
    - <div [ngSwitch]="color">
    -   <p *ngSwitchCase="'red'">Red selected</p>
    -   <p *ngSwitchCase="'blue'">Blue selected</p>
    -   <p *ngSwitchDefault>Select a color</p>
    - </div>

Q1 c) How to Design a Simple Application in TypeScript to Demonstrate Use of Modules?

Steps:

1. **Create a Module File**
    - Export functions, classes, or variables.
    - Example: mathUtils.ts

- export function add(a: number, b: number): number {
- return a + b;
- }
-
- export function subtract(a: number, b: number): number {
- return a - b;
- }

2. **Create Main Application File**
   - Import the exported members from module.
   - Example: main.ts
   - import { add, subtract } from './mathUtils';
   -
   - console.log("Addition:", add(5, 3));   // Output: 8
   - console.log("Subtraction:", subtract(5, 3)); // Output: 2
3. **Compile & Run**
   - Use tsc to compile TypeScript files.
   - Run the generated JavaScript file using Node.js.

Let me know if you want me to add diagrams or a more detailed explanation!

Q2

---

a) Features of Three Popular Web Frameworks [6]

1. **Angular**
   - **Component-Based Architecture**: Encapsulates UI into reusable components.
   - **Two-Way Data Binding**: Synchronizes model and view automatically.
   - **Dependency Injection**: Makes services easily injectable, improving testability.

- **CLI Tooling**: ng commands for scaffolding, building, testing and deployment.
- **RxJS Integration**: Reactive programming support for handling async data.

2. **React**
   - **Virtual DOM**: Efficient diffing and updating of the real DOM for high performance.
   - **JSX Syntax**: Declarative UI definition blending JavaScript and HTML.
   - **Unidirectional Data Flow**: Predictable state changes via props and state.
   - **Rich Ecosystem**: Large community, many third-party libraries (e.g., Redux, React Router).
   - **Hooks API**: Encapsulate stateful logic in functional components.
3. **Django**
   - **MTV Pattern**: Model–Template–View architecture for clear separation of concerns.
   - **Built-in ORM**: Define models in Python, with automatic SQL generation.
   - **Admin Interface**: Auto-generated CRUD UI for database models.
   - **Batteries-Included**: Authentication, routing, forms, and security features out of the box.
   - **Scalability & Security**: CSRF protection, SQL injection prevention, and easy scaling.

---

b) Using the Term "TypeScript" & Its Pros/Cons [6]

- **What is TypeScript?**
  A superset of JavaScript that adds **static typing**, **interfaces**, and **compile-time checks**, transpiling down to plain JS.
- **Advantages**
  1. **Early Error Detection**: Catches type mismatches at compile time.
  2. **Enhanced IDE Support**: Autocomplete, refactoring, and inline documentation.
  3. **Maintainability**: Clear contracts via interfaces and types, easing large-scale codebases.

4. **Modern Syntax**: Supports upcoming ECMAScript features with backward-compatibility.

- **Disadvantages**
  1. **Setup Overhead**: Requires compiler configuration (tsconfig.json) and build step.
  2. **Learning Curve**: Developers must learn type annotations and generics.
  3. **Verbose Code**: Type annotations can add boilerplate, especially for simple scripts.
  4. **Compilation Delay**: Extra compile step can slow down development feedback loop.

---

c) Two React Hooks with Examples [6]

1. **useState**
   - **Purpose**: Add local state to a functional component.
   - **Example**:

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

   - **Notes**:
      - count holds current state.
      - setCount updates it and triggers a re-render.

2. **useEffect**
   - **Purpose**: Perform side effects (data fetching, subscriptions) after render.
   - **Example**:

```
import React, { useState, useEffect } from 'react';

function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://api.example.com/items')
      .then(res => res.json())
      .then(json => setData(json));
  }, []); // empty array → run once on mount

  return (
    <div>
      {data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Loading...'}
    </div>
  );
}
```

   - **Notes**:
      - Runs **after** initial render and on dependency changes.
      - Cleanup function (optional) handles teardown (e.g., unsubscribing).
