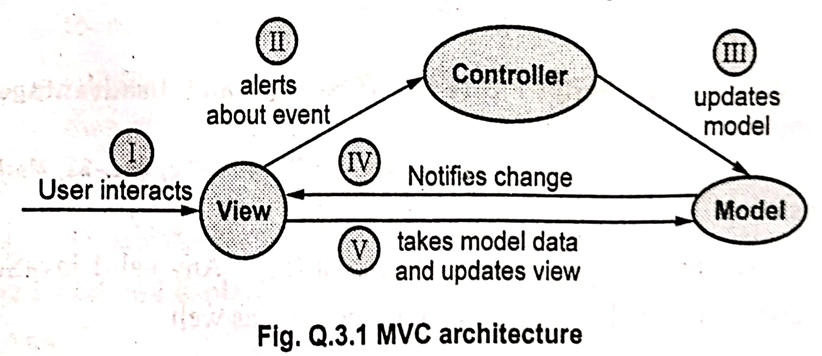
UNIT 3

Q1) **What is a web framework? Give the reasons for using web framework.**Web framework is a software framework which is designed to support the development of web applications including web services, web resources and Web APIs.   
Web framework is basically a software library that enables developers to write software that runs on the web.   
Web frameworks may be written in different languages and using different methodologies. The term 'stack' is applied to refer to the collection of different languages, software, and frameworks in use behind a specific service.   
**Why Web Framework?**Following are some important reason which indicate why do we use web frameworks -1. Saves time: The most striking feature of web framework is that it saves time and energy in developing any app because the developer doesn't need to worry about session handling, error handling and authentication logic. These functions are taken care of by web frameworks. 2. Well organized app: The web framework itself takes care of managing directories and files. This makes the complete application well organized. 3. Flexibility and customizable Add-ons, themes, plugins, widgets enable rapid customization on the web application. This brings out a lot of flexibility in code development. 4. Code reusability: Framework also promotes the reuse of code. 5. Security: Framework makes developer sure that the application uses good security measurements because framework itself takes care of it.  
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------Q2) List and explain the features of any three popular web frameworks.   
**1) Bootstrap**  
**Very well known and built by Twitter:** Bootstrap gained significant popularity due to its origin from Twitter and its comprehensive set of front-end components.  
**Easy to learn and looks professional:** It provides pre-designed templates and components that are easy to implement, allowing developers to create professional-looking websites quickly.  
**Can be difficult to customize components:** While offering ease of use, deep customization of Bootstrap's default components can sometimes be challenging.

**2) Angular**  
**Built by Google:** Developed and maintained by Google, Angular is a robust framework.  
**Well supported:** It benefits from strong backing and continuous updates from Google, along with a large community.  
**Encourages reusability:** Angular promotes building applications with reusable components, which improves efficiency and maintainability.  
**Improves application scalability:** Its structured approach and features like dependency injection contribute to building scalable applications.

**3) React**  
**Built by Facebook:** Developed by Facebook (now Meta), React is a widely used JavaScript library for building user interfaces.  
**Bundles frontend code into components:** React's core concept is based on components, allowing developers to break down the UI into smaller, manageable, and reusable pieces.  
**Organizes code and data to make code more reusable:** By using a component-based architecture and managing data flow, React facilitates code reusability.  
**Large learning curve:** While powerful, React can have a relatively steep learning curve, especially for developers new to its concepts like JSX and the Virtual DOM.

4) Express   
**Uses JavaScript:** Express.js is a back-end web application framework for Node.js, and it uses JavaScript as its programming language.   
**Very customizable:** Express is known for its flexibility and minimalistic nature, allowing developers a high degree of customization.   
**Very lightweight:** It is designed to be a lightweight framework, providing core features without imposing too many opinions or unnecessary overhead.   
**Less built-in features:** Compared to some other frameworks, Express provides a relatively small set of built-in features, adhering to its minimalist philosophy and allowing developers to choose and integrate external libraries as needed.   
**Node is very fast:** As Express runs on Node.js, it benefits from Node.js's non-blocking, event-driven architecture, which is known for its speed and efficiency in handling concurrent requests.  
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
q3) **MVC Architecture and features**

**  
What is MVC?** : MVC stands for Model, View, and Controller. It is a pattern for the architectural framework of an application.

**MVC Architecture**: The MVC architecture consists of three parts:  
1. **Model:** This part of the architecture is responsible for managing the application data. The Model responds to requests made from the View and gives instructions to the Controller to update when a request is made.  
2. **View:** This part takes care of the presentation of data. The data is presented in the desired format with the help of the View. This can be implemented using scripting technologies like JSP, ASP, PHP, and so on.  
3. **Controller:** The Controller receives input, validates it, and then performs business operations that modify the state of the data model. The Controller basically responds to user requests and performs interaction with the Model.  
In essence, the Model represents the data, the View is the user interface, and the Controller is the request handler.

**Features of the MVC Framework**  
The MVC framework offers several features:  
1. **Separation of logic:** There is a clear separation of application tasks such as input logic, business logic, and UI logic. This separation makes testing and debugging easier, as modifications in one component do not affect the others.  
2. **Ability to provide multiple views:** In the MVC model, you can create multiple views for the same data, allowing for different presentations of the information.  
3. **Faster development process:** MVC supports rapid and parallel development. When using an MVC model to develop a web application, different programmers can work simultaneously on the View and the Controller, accelerating the development process.  
4. **Returns data without formatting:** The MVC pattern returns data without applying any specific formatting. This allows the same components to be used and called for use with any interface.  
5. **Customization:** It is an extensible and pluggable framework. MVC frameworks are designed so that the components can be easily replaced or customized.  
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
q4)**TypeScript  
What is TypeScript?**  
TypeScript is a typed superset of JavaScript. Any valid JavaScript code is basically a valid code for TypeScript as well. TypeScript extends JavaScript by adding data types, classes, and other object-oriented features with type-checking. The TypeScript code compiles to plain JavaScript code. It is a programming language developed and maintained by Microsoft and may be used to develop JavaScript applications for both client-side and server-side execution.

**Advantages of Using TypeScript:**  
1. **Open source with continuous development:** It is an open-source language that is constantly being developed and improved.  
2. **Runs on any web browser:** Since TypeScript compiles to plain JavaScript, the resulting code can be executed on any web browser.  
3. **Can be called from existing JavaScript code:** TypeScript is interoperable with JavaScript, allowing you to use existing JavaScript code within a TypeScript project and vice versa. It can also work with JavaScript frameworks and libraries without issue.  
4. **Support for latest JavaScript features:** TypeScript has support for the latest JavaScript features from ECMAScript2015 (ES6) and beyond. ECMAScript is a standardized specification of a scripting language, ensuring the interoperability of web pages across different web browsers.

**Disadvantages of Using TypeScript:**  
1. **Browsers do not understand TypeScript code directly:** Web browsers can only execute JavaScript. Therefore, TypeScript code must be compiled into JavaScript before it can be run in a browser.  
2. **Does not support abstract classes:** According to the provided image, TypeScript does not support abstract classes. (Note: This information might be outdated as TypeScript has evolved, but it is presented as a disadvantage in the provided text).  
3. **TypeScript takes a time to compile the code:** The compilation process from TypeScript to JavaScript adds an extra step to the development workflow, which can take some time.

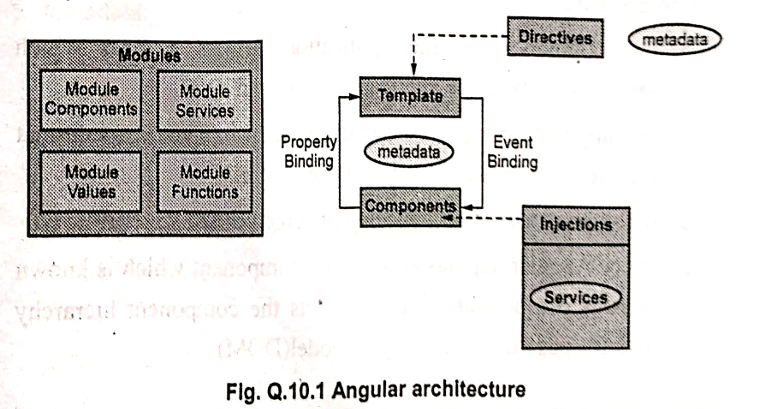
**Features of TypeScript:**  
1. **Portability:** TypeScript is portable because it can be executed on any browser or operating system. It can run in any environment where JavaScript runs.  
2. **JavaScript compatibility:** Code written in JavaScript with a .js extension can be converted to TypeScript by simply changing the extension from .js to .ts.  
3. **Static Type checking:** This is a key feature that allows for type checking at compile time. This helps catch type-related errors early in the development process.  
4. **Support for Library:** For developing with TypeScript, you can use all JavaScript frameworks, tools, and libraries easily.  
5. **Support for OOP:** TypeScript can make use of various features of object-oriented programming languages such as classes, interfaces, inheritance, and so on.  
6. **Client and Server side Development:** TypeScript allows the development of both server-side and client-side programming, making it versatile for full-stack development.  
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
q5) Write a sample application in TypeScript to demonstrate the use of modules.  
**What is a Module?**  
A module in TypeScript is used to set code written in TypeScript. A module can be defined in a separate .ts file which can contain functions, classes, interfaces, and variables. We use the prefix export with all the definitions which we want to include in a module and want to access from other modules.

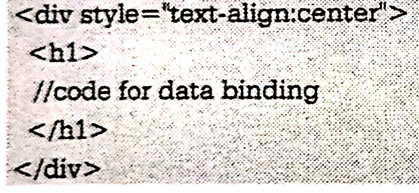
**Demo Example:**  
**Step 1: Create calculator.ts module**  
This step involves creating a file named calculator.ts which will serve as our module. This module contains a class named Addition. The export keyword is used before the class definition to make it accessible from other modules.  
A screen shot of a computer code

AI-generated content may be incorrect.  
The Addition class has two number properties a and b, a constructor to initialize these properties, and a display method that prints the sum of a and b to the console.  
**Step 2: Create app.ts for importing and using the class**  
Now, we will create another file named app.ts to import and use the Addition class from the calculator.ts module. A module can be used in another module using an import statement.  
The syntax for importing is: **import { exportName } from 'file\_path\_without\_extension';**Following this syntax, the app.ts file will look like this:  
A close up of text

AI-generated content may be incorrect.  
In app.ts, we import the Addition class from the ./calculator module (the file extension .ts is omitted in the import path). Then, we create a new instance of the Addition class with values 100 and 200 and call the display method on this object, which will output "Addition: 300" to the console when the compiled JavaScript code is executed.  
-----------------------------------**----------------------------------------------------------------------------------------------------------------------------------------------  
q6) Angular and its Features ,** **Angular CLI:**  
**What is Angular?**  
Angular 10+ is a JavaScript framework used to create single-page applications (SPAs). A single-page application dynamically rewrites the current page rather than loading entire new pages from a server. SPAs provide users with a very fluid, reactive, and fast experience. Angular is completely based on components, and the applications created in Angular are designed to give a reactive experience. Angular applications are created with the help of HTML and TypeScript.

**Features of Angular:**  
1. **Support for multiple platforms:** Angular is a cross-platform language, allowing developers to create desktop applications and progressive web applications in addition to web applications.  
2. **High speed web applications:** Angular applications are designed to be fast and load quickly with the help of its component router. Angular is particularly used as a front-end web development framework for programming languages like Node.js, .Net, PHP, Struts, Spring, and other servers for instant rendering by just using HTML and CSS.  
3. **Productivity:** Angular enhances productivity by using template syntax and command-line tools for adding and testing components.  
4. **Localization:** One of the key features of Angular 10+ is its support for merging multiple translation documents, which is an improvement over previous versions that could only load one file.  
5. **Dynamic development:** The Angular team releases two versions per year, providing developers with timely and updated functionalities with each new release.  
6. **Full stack development:** Angular is a complete framework of JavaScript and is relevant in the context of full-stack development. It provides features for testing, animations, and accessibility.

**Angular CLI:**  
The Angular CLI (Command-Line Interface) is a command-line interface tool used to initialize, develop, scaffold, and maintain Angular applications directly from a command prompt or terminal.     
It provides commands and schematics that facilitate faster code generation. Most frameworks and applications typically come with a CLI.     
You can also install Angular with the help of the Angular CLI.  
-----------------------------------**---------------------------------------------------------------------------------------------------------------------------------------------  
q7)** Explain the Angular architecture with suitable block diagram.  
The main building blocks of Angular architecture are - (Refer Fig. Q.10.1)   
1. Modules 2. Template   
3. Components 4. Data Binding   
5. Directives 6. Metadata   
7. Injections and Services.   
**1. Modules:**   
Every Angular App has a root module which is called as AppModule. It provides bootstrap mechanism that launches the application.   
Organizing your code into distinct functional modules helps in managing the development of complex applications.   
If we want to use another custom Angular module, then we need to register that module inside the app.module.ts file.   
An app typically contains many functional modules.

**2. Template**Template is basically used to combine HTML with Angular Markup and modify HTML element before displaying them. Template directives provide program logic and binding markup which connects application data and DOM. Typically in out Angular App the template looks like this -  
****

**3. Components**The component in the application defines a class which, basically contains the application logic and data. The application logic is normally written in TypeScript format and view of the page is in HTML template. A component controls a display of screen called a view. Every Angular app has at least one component which is known as root component which connects the component hierarchy with the page document object model(DOM).

**4. Data Binding**

There are two types of data bindings -  
1. Property Binding: This type of binding allows to pass the interpolated values from application data to HTML. The interpolated values are specified in {{ and } bracket pair. For instance the student.name is interpolating value.

<p>Name: {{student.name}}</p>

2. Event Binding: Event binding is used to capture events on the user's end on the app and respond to it in the target environment by updating the application data.

**5. Directives**The directives extend the HTML with the help of new syntax.  The directives are written using the prefix ng. These directives attach certain behaviour to the elements. There are two commonly used directives ng-model and ng-bind. ng-model: The ng-model binds the value of the HTML control with the specified AngularJS expression value. ng-bind: This directive replaces the HTML control value with a specified AngularJS expression value.

**6. Metadata**Metadata means data about data. Metadata tells Angular how to process a class. It is used to decorate the class so that it can configure the expected behavior of a class. The metadata for a component class has a template that defines a view.  
The metadata for a service class consists of information Angular needs to make it available to components through dependency injection (DI).

**7. Injections and Services**In Angular, it is a practice for data or logic that it should not be associated with a specific view. It is created as a service class. Due to this, the data or logic is shared across components.   
By dependency injection, we mean to allow access to the service by subscribing it. It acts as a delegate to the service.   
The @Injectable decorator immediately precedes the service class definition. The decorator provides the metadata that allows our service to be injected into client components as a dependency.  
-----------------------------------**---------------------------------------------------------------------------------------------------------------------------------------------  
Q8)** **Sample Layout of an Angular Application with Multiple Components**An Angular application is built using components as fundamental building blocks. These components are associated with templates and define different aspects of the user interface. A typical organized page in an Angular application that contains multiple components can be visualized as follows:  
A close-up of a computer code

AI-generated content may be incorrect.  
This layout demonstrates how a page can be composed of distinct components like a navigation bar, a sidebar, a main content section, and a footer.  
Each component typically consists of:  
An HTML template that declares what renders on the page.  
A TypeScript class that defines the component's behavior and logic.  
A CSS selector that defines how the component is used in a template.

**How to Create and Use Components in Angular**  
The standard way to create components in Angular is by using the Angular CLI (Command Line Interface).

1. **Creating a Component using Angular CLI:** You can create a new component using the **ng generate component** command followed by the desired component name. For example, to create a component named 'welcome', you would use:  
   **ng generate component welcome**This command will generate the necessary files for the 'welcome' component (e.g., welcome.component.ts, welcome.component.html, welcome.component.css, and welcome.component.spec.ts) and automatically declare it in the appropriate Angular module
2. **Using a Component in a Template:**Once a component is created, you can use it within the template of another component, typically the app.component.html file for root-level components or within other components' templates.

Component Selector: Every component has a selector defined in its @Component decorator. This selector is the custom HTML tag you use to insert the component into a template. For the 'welcome' component created above, the default selector would be app-welcome.

Embedding the Component: To use the 'welcome' component in your app.component.html, you would add its selector as a tag:

A close-up of a computer code

AI-generated content may be incorrect.  
When Angular renders app.component, it will find the <app-welcome> tag and render the content of the welcome.component.html template in its place.

**Example:**

Let's consider a simple 'welcome' component:

welcome.component.ts:  
A screen shot of a computer code

AI-generated content may be incorrect.The selector: 'app-welcome' defines the tag used to embed this component. The currentUser property is data available within the component.

welcome.component.html:  
<p>welcome {{currentUser}}</p>

This template uses interpolation {{currentUser}} to display the value of the currentUser property from the component's class.

A screenshot of a chat

AI-generated content may be incorrect.  
-----------------------------------**---------------------------------------------------------------------------------------------------------------------------------------------Q9) Explain event binding and property binding in angular with example.**In Angular, binding is a mechanism that allows communication between the component's TypeScript class and its HTML template. Two common types of binding are property binding and event binding.  
**Property Binding**: Property binding is used to pass data from the component class to the HTML template (Data Flow: Component -> DOM). It allows you to set the value of an HTML element's property using a property from your component class. Property binding is achieved using square brackets [] around the HTML element's property.  
**Syntax:** <element [property]="componentProperty"></element>

**Example:** Let's say you have a component with a property imageUrl and you want to bind this property to the src attribute of an <img> tag.  
**Component (.ts file):**A computer screen shot of a computer code

AI-generated content may be incorrect.  
**Explanation:**  
[src]="imageUrl" binds the src property of the <img> tag to the imageUrl property in the component class. When the value of imageUrl changes in the component, the image displayed in the template will update.  
[disabled]="isButtonDisabled" binds the disabled property of the <button> tag to the isButtonDisabled property in the component class. If isButtonDisabled is true, the button will be disabled; otherwise, it will be enabled.

**Event Binding**  
Event binding is used to respond to events triggered in the HTML template and execute a method in the component class (Data Flow: DOM -> Component). It allows you to listen for events like clicks, keypresses, mouseovers, etc., and run specific code in your component when those events occur. Event binding is achieved using parentheses () around the event name.  
**Syntax:** <element (event)="componentMethod($event)"></element>

**Example:** Let's say you have a button and you want to call a method in your component when the button is clicked.

* **Component (.ts file):**

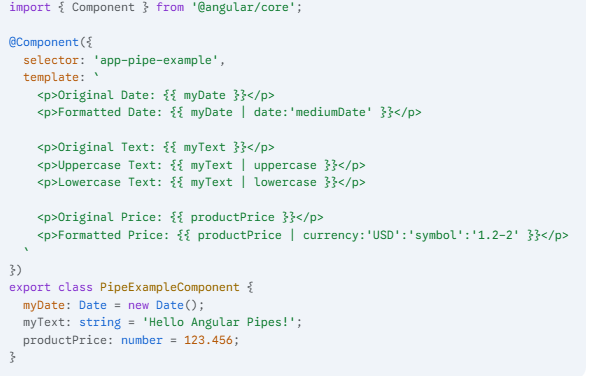
A screen shot of a computer code

AI-generated content may be incorrect.

* **Explanation:**
  + (click)="onClickMe()" binds the click event of the <button> tag to the onClickMe() method in the component class. When the button is clicked, the onClickMe() method will be executed, updating the message property.
  + The {{ message }} uses interpolation to display the value of the message property in the paragraph.

**Key Difference:**  
**Property Binding:** Data flows from the **component to the DOM**. Used to set HTML element properties based on component data.  
**Event Binding:** Data/action flows from the **DOM to the component**. Used to execute component methods in response to DOM events.  
-----------------------------------**---------------------------------------------------------------------------------------------------------------------------------------------  
Q10)** **What is pipe? Explain with example.**In Angular, a **Pipe** is a feature that allows you to transform data in your templates before displaying it. Pipes are a simple way to format data for presentation to the user without changing the original data source. They are typically used within interpolation or property binding expressions in your HTML templates.  
Think of a pipe as a function that takes data as input, performs an operation on it, and returns the transformed output.  
**Syntax for using Pipes:** You use the pipe operator (|) followed by the pipe's name within your template expressions:  
 **{{ data | pipeName }}**Some pipes can also accept arguments, which are passed after the pipe name, separated by colons (:):{{ data | pipeName: argument1 : argument2 }}

Exp**lanation with Example:** Angular provides several built-in pipes for common data transformations. Let's look at an example using the DatePipe, UpperCasePipe, and CurrencyPipe.

Suppose you have a component with the following properties:  
  
**Explanation of the Template (.html) and Pipe Usage:**  
1. **{{ myDate | date:'mediumDate' }}**:  
myDate is the input data (a Date object).  
| is the pipe operator.  
date is the name of the built-in DatePipe.  
:'mediumDate' is an argument passed to the DatePipe, specifying the desired format for the date. 'mediumDate' is a predefined format.

2.**{{ myText | uppercase }}**:  
myText is the input data (a string).  
| is the pipe operator.  
uppercase is the name of the built-in UpperCasePipe. This pipe doesn't require any arguments.

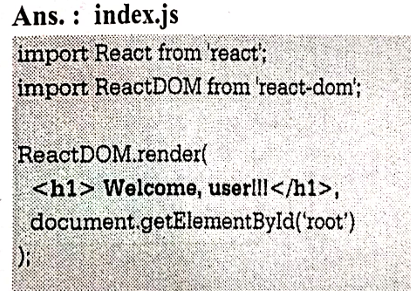
3.**{{ myText | lowercase }}**:  
myText is the input data (a string).  
| is the pipe operator.  
lowercase is the name of the built-in LowerCasePipe. This pipe also doesn't require any arguments.

4.**{{ productPrice | currency:'USD':'symbol':'1.2-2' }}**:  
productPrice is the input data (a number).  
| is the pipe operator  
currency is the name of the built-in CurrencyPipe.  
:'USD' is the first argument, specifying the currency code (US Dollars).  
:'symbol' is the second argument, indicating that the currency symbol should be displayed.  
:'1.2-2' is the third argument, specifying the format of the number (minimum 1 digit before the decimal, minimum 2 digits after, maximum 2 digits after).

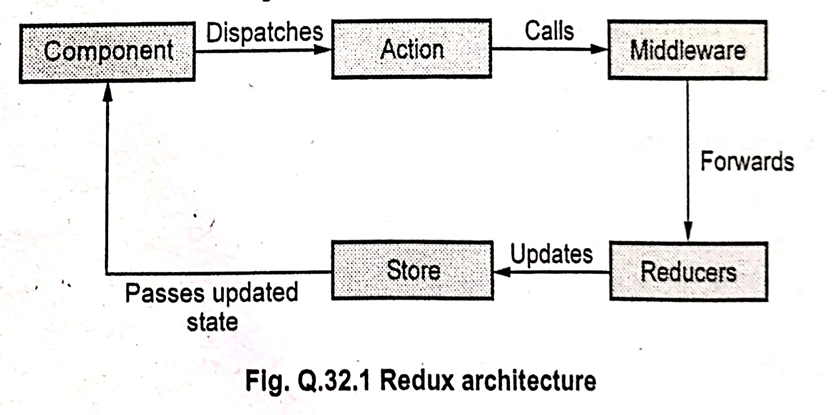
**Output of the Example:** Assuming the current date is something like May 13, 2025, the output rendered in the browser would be similar to:  
A screenshot of a computer

AI-generated content may be incorrect.  
As you can see, pipes transform the data (myDate, myText, productPrice) into a more readable and desired format for display in the template without altering the original values in the component class.  
Angular also allows you to create custom pipes to perform specific data transformations that are not covered by the built-in pipes.  
-----------------------------------**---------------------------------------------------------------------------------------------------------------------------------------------  
Q11)  
What is ReactJS?**  
ReactJS is an open-source, component-based front-end JavaScript library that is maintained by Facebook. It is primarily responsible for the view layer of an application. This means ReactJS is used for building user interfaces.  
**Features of ReactJS:**  
1. **Virtual DOM:** React uses a Virtual DOM (Document Object Model). The Virtual DOM is a representation of the original DOM. When a user updates something on the web application, React updates the Virtual DOM first. Unlike traditional JavaScript frameworks that might update the whole DOM (which can be slower), React updates only the part of the DOM that is required to be updated. This property makes DOM manipulation much faster than in many other frameworks.  
2. **Components:** This feature allows the web developer to create custom elements which can be reused in HTML. React applications are built using components, which are self-contained, reusable blocks of code that manage their own state and can be composed together to build complex UIs.  
3. **JSX:** JSX is an extension of JavaScript syntax. It can be seen as a combination of JavaScript and XML (or HTML). The syntax of JSX is very simple and makes writing components and defining their structure very easy and readable within the JavaScript code.  
4. **One-way Data Binding:** ReactJS is designed in such a way that it follows unidirectional or one-way data binding. This means that data is allowed to flow in one direction at a time, typically from parent components to child components. This helps in achieving greater control over the application and improves the efficiency of the application.

**Simple ReactJS Application Example (Displaying a Welcome User Message):**

  
-----------------------------------**---------------------------------------------------------------------------------------------------------------------------------------------**  
Q12)  
A paper with text on it

AI-generated content may be incorrect.  
-----------------------------------**---------------------------------------------------------------------------------------------------------------------------------------------  
Q13)** **Redux architecture:  
What is Redux?**  
Redux is a predictable state container for JavaScript applications. It is commonly used with libraries like ReactJS for managing the application's state in a centralized and predictable manner. Redux helps in managing complex application states, making it easier to understand how the state changes and to debug issues.

**Redux Architecture Core Concepts:**

Redux works on the following core concepts:

1. **Action:**
   * An Action is a simple JavaScript object that represents an intent to change the state.
   * Actions are the only way to send (dispatch) information from your application (e.g., from a React component) to the Redux store.
   * They must have a type property that indicates the type of action being performed (e.g., 'ADD\_TODO', 'FETCH\_USER\_DATA').
   * Actions can also carry a payload of data that is needed to update the state.
   * Examples of actions include fetching user data, logging a user in, adding an item to a cart, and so on.
2. **Reducer:**
   * A Reducer is a pure synchronous function that determines how the application's state should change in response to an action.
   * It takes two arguments: the current state and the action object that was dispatched.
   * It returns the *new* state based on the action type and payload.
   * Reducers must be pure functions, meaning they should not have side effects (like making API calls or modifying the existing state directly) and should return the same output for the same input. They should only calculate the next state.
3. **Store:**
   * The Store is a centralized place that holds the entire application's state tree.
   * There is only one store in a typical Redux application.
   * The store's responsibilities include:
     + Holding the application state.
     + Allowing access to the state via getState().
     + Allowing the state to be updated via dispatch(action).
     + Registering listeners via subscribe(listener).
     + Handling unregistering of listeners via the function returned by subscribe(listener).
   * Developers can access and update the store with the help of various methods provided by Redux.
4. **Middleware:**
   * Middleware provides a way to intercept actions before they reach the reducer.
   * It is used to execute asynchronous code or perform side effects between the dispatching of an action and the moment it reaches the reducer.
   * Common use cases for middleware include logging, handling asynchronous operations (like API calls using redux-thunk or redux-saga), and routing.

**Redux Architecture Data Flow:**

The data flow in Redux follows a strict unidirectional cycle, as illustrated in the provided diagram:

1. **Component:** A React Component (or any part of your application) dispatches an **Action**. This typically happens in response to a user interaction (e.g., clicking a button) or an event (e.g., a successful API call).
2. **Dispatches Action:** The component calls the store.dispatch(action) method, sending the action object to the store.
3. **Middleware (Optional):** If middleware is configured, the action first passes through the middleware. Middleware can inspect the action, perform asynchronous operations, or even stop or modify the action before forwarding it.
4. **Calls Middleware:** The dispatch process can involve calls to configured middleware.
5. **Forwards:** Middleware can forward the action to the next piece of middleware or to the reducers.
6. **Reducers:** The dispatched action reaches the Reducers. The root reducer, which is a combination of all your individual reducers, receives the current state and the action. Each individual reducer is responsible for updating a specific slice of the state based on the action type.
7. **Updates State:** The reducers compute the new state based on the action and return the new state object.
8. **Store:** The Store receives the new state from the reducers and updates its internal state tree.
9. **Passes Updated State:** The Store notifies any subscribed components that the state has changed.
10. **Component:** Components that are connected to the Redux store and are interested in the changed state receive the updated state as props or through hooks. This triggers a re-render of the components, updating the user interface to reflect the new state.

This unidirectional data flow makes the application's state changes predictable and easier to track, which simplifies debugging and development.  
-----------------------------------**---------------------------------------------------------------------------------------------------------------------------------------------  
Q14) React Hooks?**Hooks were added to React in version 16.8. They are functions that allow function components to have access to state and other React features that were previously only available in class components (like lifecycle methods). Essentially, Hooks allow you to "hook into" React features such as state and lifecycle methods from functional components, removing the need for class components in many cases.

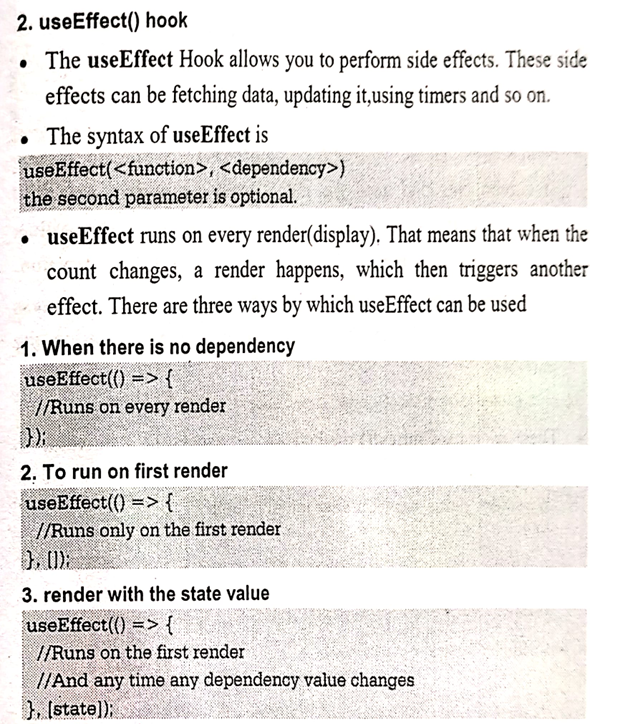
**Rules of Hooks:**  
There are a few important rules to follow when using Hooks:

1. **Only call the hooks at the top level:** Do not call Hooks inside loops, conditions, or nested functions. Call them at the top level of your React function component.
2. **Only call the hooks from React Functions:** Do not call Hooks from regular JavaScript functions. Call them from React function components or custom Hooks.
3. **Hooks cannot be conditional:** Do not call Hooks inside conditional blocks (if statements). Hooks must be called in the same order on every render.

**Different Types of Hooks:**  
1. **useState hook:**  
**Purpose:** The useState hook allows you to add state to functional components. It returns a stateful value and a function to update that value.  
**Example:** The provided image shows an example of a simple counter component using useState:  
A screenshot of a computer program

AI-generated content may be incorrect.  
**Explanation:**

* useState(0): Initializes a state variable named count with an initial value of 0. It returns an array containing the current state value (count) and a function to update it (setCount).
* onClick={() => setCount(count + 1)}: When the button is clicked, the setCount function is called with count + 1, which updates the count state variable. React re-renders the component with the new count value.

****A close-up of a document

AI-generated content may be incorrect.