1. **What and Why Do We Use @Composable?**
   * **@Composable** is an annotation used in Jetpack Compose, indicating that the function can define and manage a part of the UI. These functions are special because they can be recomposed by the Compose runtime when the data they depend on changes. This allows for dynamic and responsive UIs that automatically update when the underlying data changes.
2. **What Is Surface?**
   * **Surface** is a Composable in Jetpack Compose that provides a simple way to apply common visual elements like background color, elevation, shape, and borders to the layout. It's often used as a container for other Composables and can help standardize the appearance and behavior of different UI elements.
3. **Why Would You Use Preview?**
   * The **@Preview** annotation is used to create a preview of Composable functions directly in the Android Studio IDE. This feature is invaluable during development as it allows developers to see a visual representation of the UI components without needing to run the app on a device or emulator. It speeds up the development and design process significantly.
4. **How to Add Components Such as Text and Button?**
   * In Jetpack Compose, UI components are added by calling Composable functions. For example, **Text("Hello World")** creates a text view, and **Button(onClick = { /\* handle click \*/ }) { Text("Click Me") }** creates a button with a text label. These functions can be called within any Composable function or layout.

import androidx.compose.material.Text

import androidx.compose.material.Button

import androidx.compose.runtime.Composable

@Composable

fun MyScreenContent() {

// Adding a Text component

Text("Hello World")

// Adding a Button component with Text

Button(onClick = { /\* handle click \*/ }) {

Text("Click Me")

}

}

1. **How to Create Columns and Rows in Kotlin?**
   * In Jetpack Compose, you can create columns and rows using the **Column** and **Row** composables, respectively. These function as layout containers that arrange their children vertically (**Column**) or horizontally (**Row**). You can add children composables within them to create structured layouts.

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Row

import androidx.compose.material.Text

import androidx.compose.runtime.Composable

@Composable

fun MyLayout() {

// Creating a Column

Column {

Text("Text in a Column") // This text will be on top

Text("Another Text in a Column") // This text will be below the first one

}

// Creating a Row

Row {

Text("Text in a Row") // This text will be on the left

Text("Another Text in a Row") // This text will be to the right of the first one

}

}

1. **What Is the State of a Component in Compose?**
   * The state in Compose refers to the data a composable function reads or writes during composition. The Compose runtime observes this state. When the state changes, Compose automatically recomposes the affected components, updating the UI to reflect the new state.
2. **What Is a Lazy List?**
   * In Jetpack Compose, a **LazyColumn** or **LazyRow** is used to efficiently display a large, scrollable list of items. Unlike a regular **Column** or **Row**, which renders all child composables immediately, **LazyColumn** and **LazyRow** only compose and render the items that are currently visible on screen, improving performance for large lists.
3. **Differences Between Android Java Way to Build UI and Jetpack Compose for Kotlin?**
   * Traditional Android UI development uses XML layouts to define the UI structure and Java/Kotlin code for logic and event handling. In contrast, Jetpack Compose, which is Kotlin-based, uses a declarative approach where UI and logic are defined together in Composable functions. Compose offers more intuitive and concise UI development with less boilerplate, automatic UI updates based on state changes, and a more reactive programming style.

Question 9:

