Modern Android Development Lab

# LAB EXERCISES

[1. Create an android application using](#_LAB_1 ) Composables, Modifiers and Event-Listeners

[2. Create an android application using Material Theme and Material Widgets](#_LAB_2:)

[3. Create an android application using Modal Drawer and Page Navigation](#_LAB_3)

[4. Create an android application with Color, Size and Visibility Animations](#_LAB_4)

[5. Create an android application performing CURD operation on the local Storage](#_LAB_5)

[6. Create an android application showing map of current location using WebView](#_LAB_6:)

7[. Create an android application displaying gyroscopic sensor realtime values](#_LAB_7:)

## Quick Reference:

* [Quick Overview](https://developer.android.com/jetpack/compose/tutorial)
* [Jetpack Compose Basics](https://www.youtube.com/live/qvDo0SKR8-k?feature=share)
* [Basics Playlist](https://youtube.com/playlist?list=PLWz5rJ2EKKc94tpHND8pW8Qt8ZfT1a4cq)
* [IceBreaker](https://youtube.com/playlist?list=PLWz5rJ2EKKc-CG9riunK996aI6cRhXFDC)
* [Event Listerners](https://youtu.be/cCOL7MC4Pl0?si=1118bYa-lfUzEAIu)
* [Course](https://developer.android.com/courses/android-basics-compose/course)

https://developer.android.com/training/kotlinplayground

<https://play.kotlinlang.org/>

Github Links: [github.com/KarthiDreamr](http://github.com/KarthiDreamr)

<https://github.com/KarthiDreamr/TapCounter-Android-Jetpack>

<https://github.com/KarthiDreamr/MaterialApp-Android-Jetpack>

<https://github.com/KarthiDreamr/ScreenNavigation-Android-Jetpack>

<https://github.com/KarthiDreamr/AndroidAnimation-Android-Jetpack><https://github.com/KarthiDreamr/DataStore-Android-Jetpack>

<https://github.com/KarthiDreamr/MapWebView-Android-Jetpack>

<https://github.com/KarthiDreamr/GyroSensing-Android-Jetpack>

Document Formatting Shortcut :

Ctrl + Alt + L

# LAB 1

AIM:

Create an android application using Composables, Modifiers and Event-Listeners

## PROCUDURE:

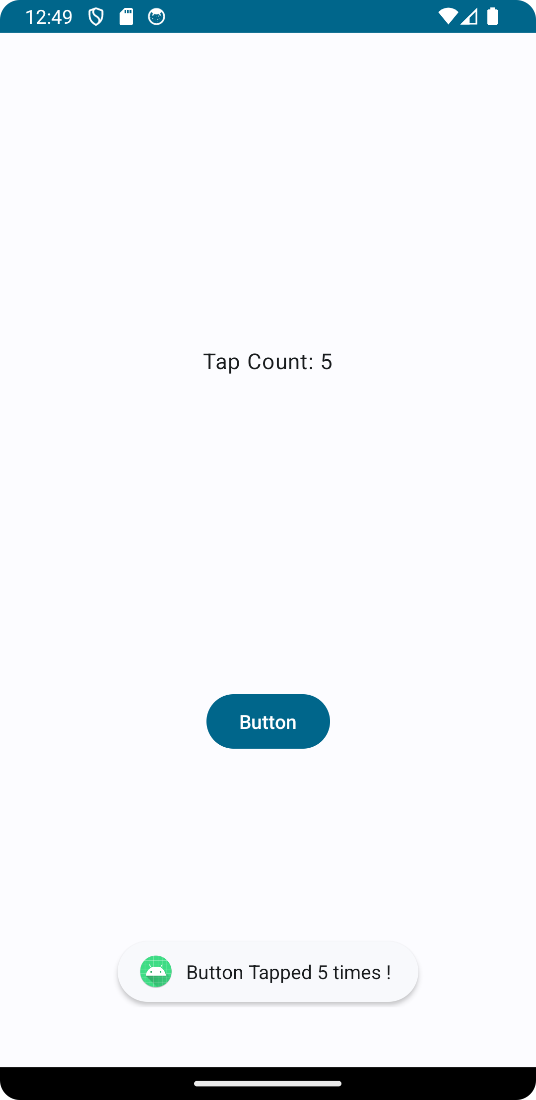
1. Create a new Android project: Open Android Studio and create a new project. Choose the “Empty Compose Activity” template.
2. Define your MainActivity: This is the entry point of your application. In your MainActivity.kt file, override the onCreate method and set the content to your custom composable function (in this case, Greeting()).
3. Create your Composable function: A composable function is a special kind of function that can be used to describe your UI. In this case, you’ve created a Greeting composable function. This function contains a Column which is a vertical layout composable that places its children in a vertical sequence.
4. Use Modifiers: Modifiers are used to modify the appearance and behavior of composables. In this case, you’ve used Modifier.fillMaxSize() to make the Column fill the entire available space.
5. Handle user interactions with EventListeners: In this case, you’ve added an onClick listener to the Button composable. When the button is clicked, it increments the tapCount and shows a toast message.
6. Use mutable state: In Jetpack Compose, you can use mutable state to hold values that can change over time. In this case, you’ve used mutableStateOf(0) to hold the tapCount.
7. Preview your Composables: You can preview your composables in Android Studio using the @Preview annotation.

## CODE:

class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContent {  
 Exercise1Theme {  
   
 Surface(  
 modifier = Modifier.fillMaxSize(),  
 color = MaterialTheme.colorScheme.background  
 ) {  
 Greeting()  
 }  
 }  
 }  
 }  
}  
  
@Composable  
fun Greeting(){  
 var tapCount by remember { mutableStateOf(0) }  
 val context = LocalContext.current  
  
  
 Column(  
 modifier = Modifier.fillMaxSize(),  
 verticalArrangement = Arrangement.SpaceEvenly,  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) {  
 Text("Tap Count: $tapCount")  
 Button(  
 onClick = {  
 tapCount++  
 Toast.makeText(context, "Button Tapped $tapCount times ! ", Toast.LENGTH\_SHORT).show()  
 }  
 ) {  
 Text("Button")  
 }  
 }  
  
}

@Preview(showBackground = true)  
@Composable  
fun GreetingPreview() {  
 Exercise1Theme {  
 Greeting()  
 }  
}

## OUTPUT:



# LAB 2:

AIM:

Create an android application using Material Theme and Material Widgets

## PROCEDURE:

1. **Root Composable Call - HomePage()**: This is the main entry point to your UI. When you call HomePage(), it creates a hierarchy of composables that define your UI.
2. **MaterialTheme Composable**: This is a composable function that applies theming to its children composables. It sets the colors, typography, and shapes that will be used by the child composables.
3. **Scaffold Composable**: This is a Material Design layout structure which provides a fixed top bar, an optional bottom bar, and content space in between. The top bar usually contains the app bar, and the content space is where you put your main UI content.
4. **TopAppBar Composable**: This is a composable function that creates a top app bar with a title. In your case, the title is “My App”.
5. **FloatingActionButton Composable**: This is a composable function that creates a floating action button with an icon. The onClick parameter should be replaced with the actual action you want to perform when the button is clicked.
6. **BottomAppBar Composable**: This is a composable function that creates a bottom app bar. You’ve added a Row of IconButtons in it.
7. **IconButton Composable**: This is a composable function that creates an icon button. You’ve created a custom IconButton called CustomIconButton. The imageVector parameter specifies the icon to display, and the onClick parameter should be replaced with the actual action you want to perform when the button is clicked.
8. **Content of Scaffold**: The last parameter of Scaffold is a trailing lambda where you define your main UI content. You’ve created a Column with some texts and spacers in it.
9. Remember, in Jetpack Compose, UI elements are functions annotated with @Composable. These functions can call other composable functions to create reusable pieces of UI, and can react to state changes in an efficient way.

## CODE:

@OptIn(ExperimentalMaterial3Api::class)  
@Composable

fun HomePage() {  
 MaterialTheme() {  
 Scaffold(  
 topBar = {  
 TopAppBar(  
 title = {  
 Text("My App")  
 }  
 )  
 },  
 floatingActionButton = {  
 FloatingActionButton(onClick = { /\* ... \*/ }) {  
 Icon(imageVector = Icons.Filled.Add, contentDescription = null)  
 }  
 },  
 bottomBar = {  
 BottomAppBar {

Row(

modifier = Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.SpaceEvenly

) {

CustomIconButton(imageVector = Icons.Filled.Favorite, onClick = { /\* Handle onClick \*/ })

CustomIconButton(imageVector = Icons.Filled.Call, onClick = { /\* Handle onClick \*/ })

CustomIconButton(imageVector = Icons.Filled.Email, onClick = { /\* Handle onClick \*/ })

CustomIconButton(imageVector = Icons.Filled.Send, onClick = { /\* Handle onClick \*/ })

CustomIconButton(imageVector = Icons.Filled.Build, onClick = { /\* Handle onClick \*/ })

}

}  
 }  
 ) { contentPadding ->  
  
 Column(  
 modifier = Modifier  
 .padding(contentPadding)  
 .padding(horizontal = 17.dp)  
 .fillMaxSize(),  
 verticalArrangement = Arrangement.SpaceEvenly  
 ) {  
  
 Text(  
 "Kotlin was replaced as the Official language for Android Development by Google on 2017",  
 fontWeight = FontWeight.ExtraBold  
 )  
 Spacer(modifier = Modifier.size(10.dp))  
 Text("Kotlin is a cross-platform, statically typed, general-purpose high-level programming language with type inference. ")  
 Spacer(modifier = Modifier.size(10.dp))  
 Text("Jetpack Compose is a modern toolkit for building native Android UI. Jetpack Compose simplifies and accelerates UI development on Android with less code, powerful tools, and intuitive Kotlin APIs.")  
 Spacer(modifier = Modifier.size(10.dp))  
 Text(  
 "Advantages of Kotlin over Java",  
 fontWeight = FontWeight.Bold  
 )  
 Text("1.Conciseness and expressiveness:")  
 Text("2.Safety and reliability")  
 Text("3.Speed and efficiency")  
 Text("4.Modern and innovative")  
  
 }  
 }  
 }  
}

@Composable

fun CustomIconButton(imageVector: ImageVector, onClick: () -> Unit) {

IconButton(

onClick = onClick

) {

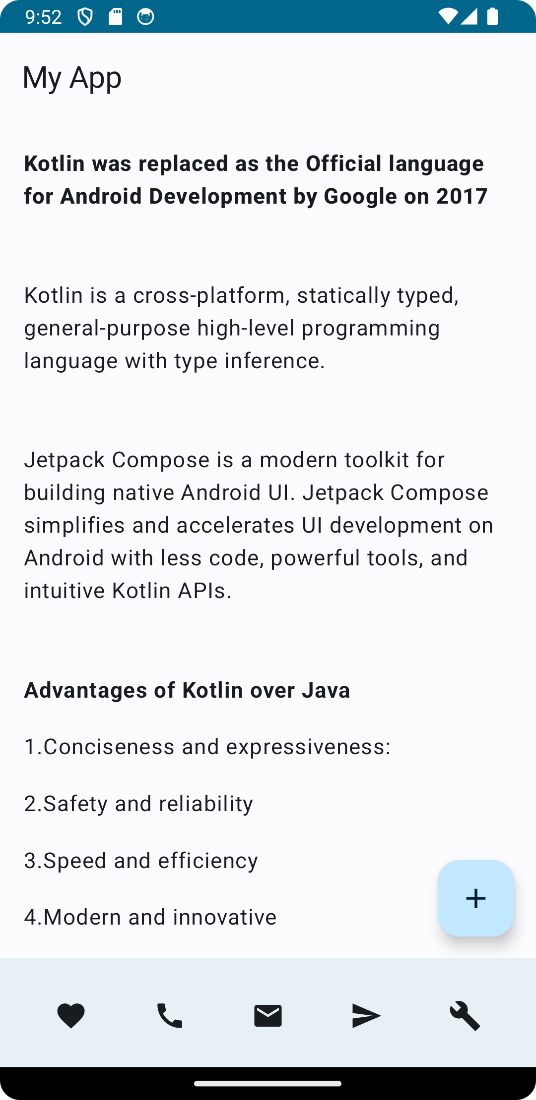
Icon(

imageVector = imageVector,

contentDescription = null

) } }

## OUTPUT:

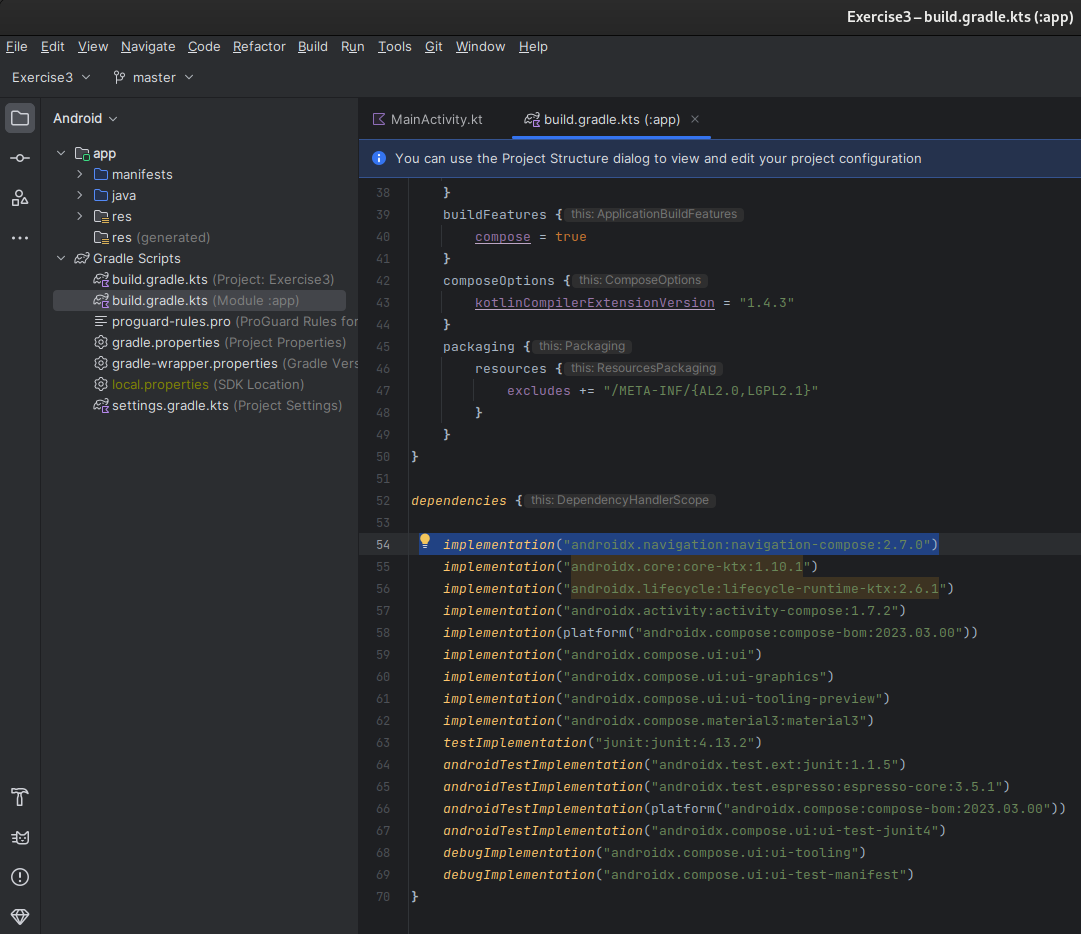


# LAB 3

AIM:

Create an android application using Modal Drawer and Page Navigation

## PROCEDURE:



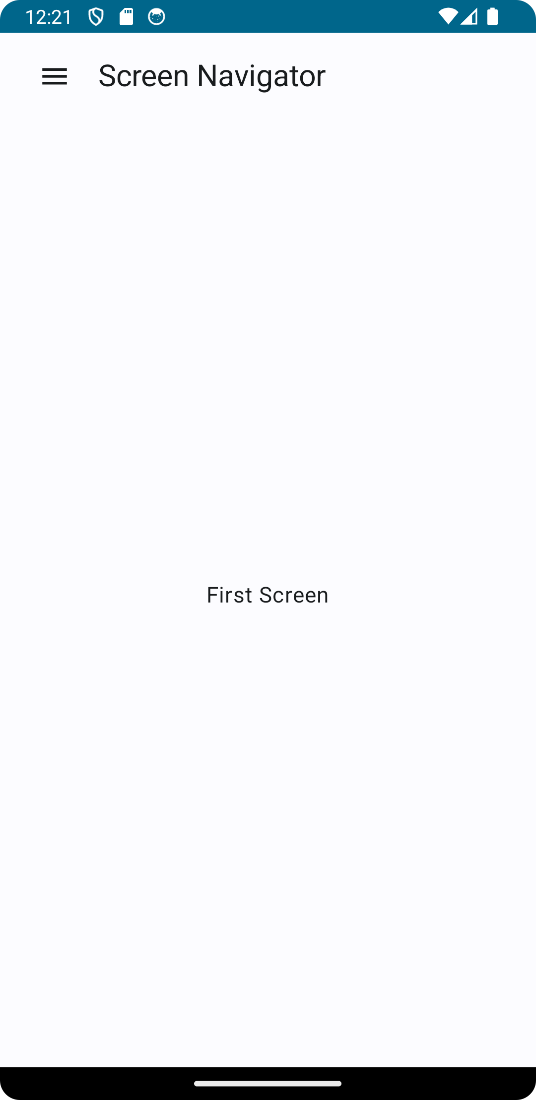
\*make sure you add this line here for navigation dependency import

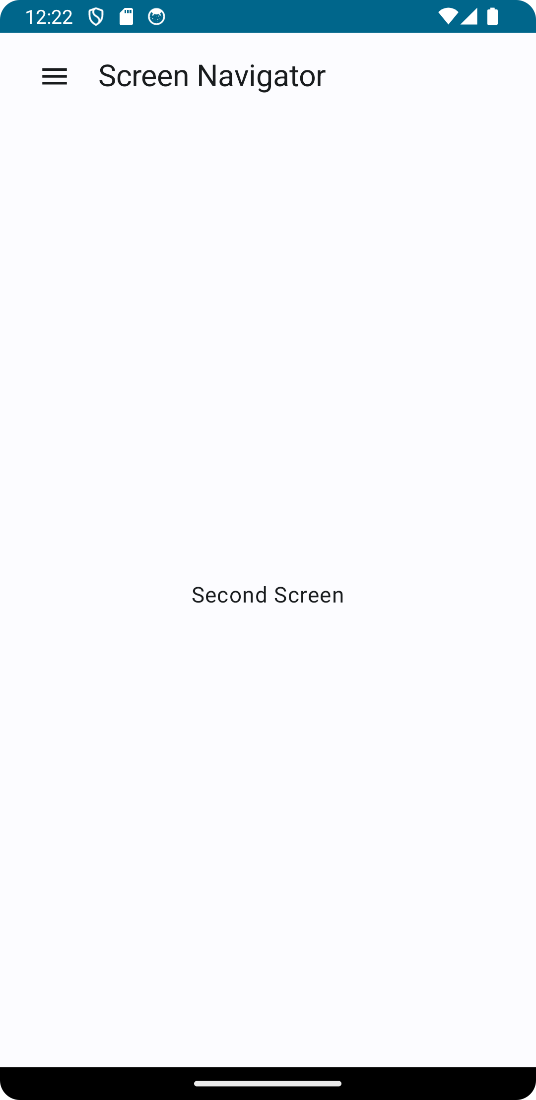
1. **Greeting() Composable**: This is the main entry point to your UI. It creates a NavHost which is a container for navigation within a Compose application. It has a NavController which controls the navigation within the NavHost.
2. **NavController**: This is an object that manages app navigation within a NavHost. It keeps track of the back stack of composables.
3. **NavHost Composable**: This is a composable that hosts a navigation graph. It takes in a NavController and a start destination. The start destination is the route of the first composable that should be displayed.
4. **composable Function**: This function is used to define a destination in your navigation graph. It takes in a route and a trailing lambda where you define what UI should be displayed at this destination.
5. **PageBuilder() Composable**: This is a composable function that you’ve defined which creates a ModalNavigationDrawer and a Scaffold. It takes in a NavController and a page name.
6. **ModalNavigationDrawer Composable**: This is a Material Design modal drawer that slides in from the left edge of the screen. You’ve defined its content to be a list of items that navigate to different screens when clicked.
7. **Scaffold Composable**: This is a Material Design layout structure which provides a fixed top bar, an optional bottom bar, and content space in between. The top bar contains an app bar with a menu icon and title, and the content space contains your main UI content.
8. **IconButton Composable**: This is a composable function that creates an icon button. In your top app bar, you’ve created an icon button with a menu icon that opens the drawer when clicked.
9. **Content of Scaffold**: The last parameter of Scaffold is a trailing lambda where you define your main UI content. You’ve created a Box with some text in it.
10. Remember, in Jetpack Compose, UI elements are functions annotated with @Composable. These functions can call other composable functions to create reusable pieces of UI, and can react to state changes in an efficient way.

## CODE:

@Composable  
fun Greeting() {  
 val navController = rememberNavController()  
 NavHost(  
 navController = navController,  
 startDestination = "first\_screen"  
 ) {  
 composable("first\_screen") {  
 PageBuilder(  
 navController = navController,  
 pageName = "First Screen"  
 )  
 }  
 composable("second\_screen") {  
 PageBuilder(  
 navController = navController,  
 pageName = "Second Screen"  
 )  
 }  
  
 }  
}  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun PageBuilder(  
 navController: NavController,  
 pageName: String  
) {  
  
 val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)  
 val scope = rememberCoroutineScope()  
  
  
 ModalNavigationDrawer(  
 drawerState = drawerState,  
 drawerContent = {  
 ModalDrawerSheet {  
 Text("Navigation Drawer", modifier = Modifier.padding(16.dp))  
 Divider()  
 NavigationDrawerItem(  
 label = { Text(text = "First Screen") },  
 selected = false,  
 onClick = {  
 navController.navigate("second\_screen")  
 }  
 )  
 NavigationDrawerItem(  
 label = { Text(text = "Second Screen") },  
 selected = false,  
 onClick = {  
 navController.navigate("second\_screen")  
 }  
 )  
 }  
 }  
 ) {  
 Scaffold(  
 topBar = {  
 TopAppBar(  
 title = {  
 Row {  
 IconButton(  
 onClick = {  
 scope.launch {  
 drawerState.open()  
 }  
 }  
 ) {  
 Icon(  
 imageVector = Icons.Default.Menu,  
 contentDescription = null  
 )  
 }  
 Text(  
 "Screen Navigator", modifier = Modifier.padding(  
 8.dp  
 )  
 )  
 }  
 }  
 )  
 }  
 ) { contentPadding ->  
  
 Box(  
 contentAlignment = Alignment.Center,  
 modifier = Modifier  
 .padding(contentPadding)  
 .fillMaxSize()  
 ) {  
 Text(  
 pageName  
 )  
 }  
 }  
 }  
}

## OUTPUT:



## REFERENCES:

* <https://developer.android.com/jetpack/compose/navigation#kts>
* <https://developer.android.com/jetpack/compose/layouts/material>
* <https://youtu.be/Y0Cs2MQxyIs?si=sHM0u4oanp_KJr4H>

# LAB 4

AIM:

Create an android application with Color, Size and Visibility Animations

## PROCEDURE:

1. **MainActivity**: This is the main activity of your application. It sets the content to be a Surface containing the HomePage composable.
2. **HomePage Composable**: This is the main composable of your application. It creates a Column with three buttons and a box. The buttons change the color, size, and visibility of the box when clicked.
3. **State Variables**: You’ve defined three state variables using remember { mutableStateOf(...) }. The blue variable controls the color of the box, the bigBox variable controls the size of the box, and the visibeBox variable controls the visibility of the box.
4. **Animating State Changes**: You’ve used animateColorAsState, updateTransition, and animateDpAsState to animate changes to the color and size of the box. When you click the “Change Color” button, it toggles the blue state variable which triggers an animation to a new color. Similarly, when you click the “Change Size” button, it toggles the bigBox state variable which triggers an animation to a new size.
5. **Buttons**: You’ve created three buttons using the Button composable. Each button has an onClick listener that toggles one of the state variables when clicked.
6. **Animated Box**: You’ve created a box using the Box composable. The box has a dynamic size and color that animate when their respective state variables change. The visibility of the box is controlled by an AnimatedVisibility composable which animates visibility changes.
7. Remember, in Jetpack Compose, UI elements are functions annotated with @Composable. These functions can call other composable functions to create reusable pieces of UI, and can react to state changes in an efficient way.

## CODE:

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

Exercise4Theme {

// A surface container using the 'background' color from the theme

Surface(

modifier = Modifier.fillMaxSize(),

color = MaterialTheme.colorScheme.background

) {

HomePage()

}

}

}

}

}

@Composable

fun HomePage() {

var blue by remember { mutableStateOf(true) }

val BoxColor by animateColorAsState(if(blue) Color.Green else Color.Magenta, label = "BlueOrange")

var bigBox by remember { mutableStateOf(true) }

val transition = updateTransition(targetState = bigBox, label = "ChangeSize")

val BoxSize by animateDpAsState(if(bigBox) 80.dp else 120.dp, label = "BigSmall")

var visibeBox by remember { mutableStateOf(true) }

Column(

modifier = Modifier

.fillMaxSize()

.padding(horizontal = 30.dp, vertical = 30.dp)

) {

Button(

onClick = {

blue = !blue

}

) {

Text("Change Color")

}

Spacer(modifier = Modifier.size(20.dp))

Button(

onClick = {

bigBox = !bigBox

}

) {

Text("Change Size")

}

Spacer(modifier = Modifier.size(20.dp))

Button(

onClick = {

visibeBox = !visibeBox

}

) {

Text("Hide / Show")

}

Spacer(modifier = Modifier.size(20.dp))

AnimatedVisibility(visibeBox){

Box (

modifier = Modifier

.size(BoxSize)

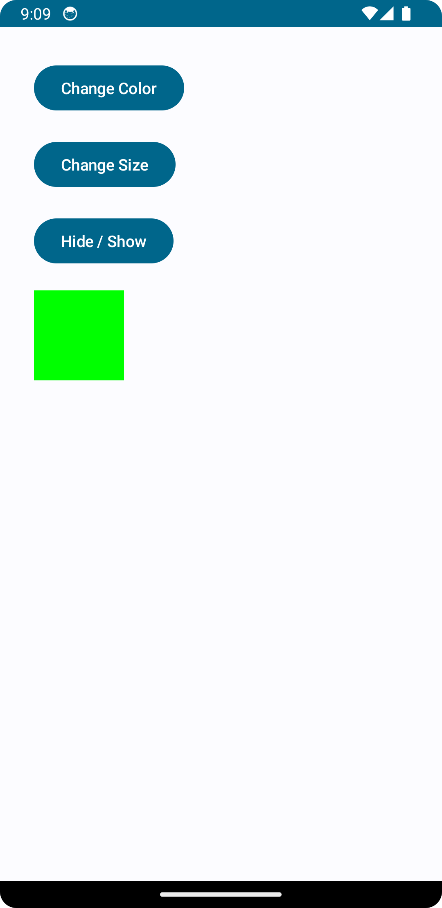
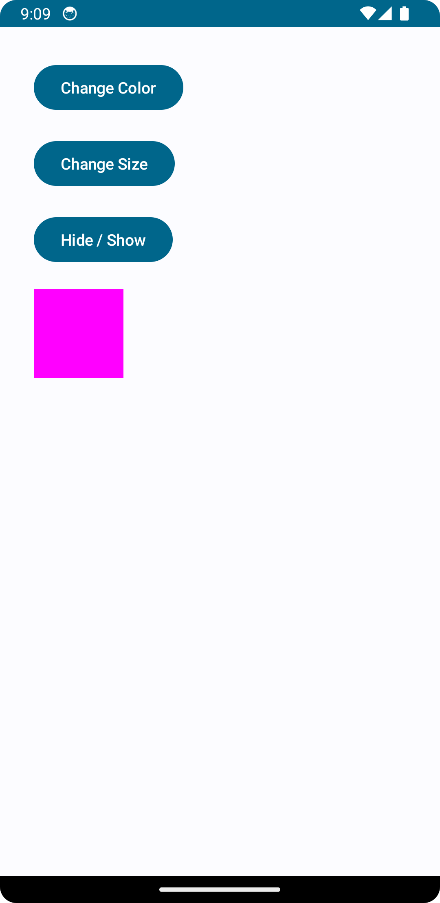
.background(BoxColor)

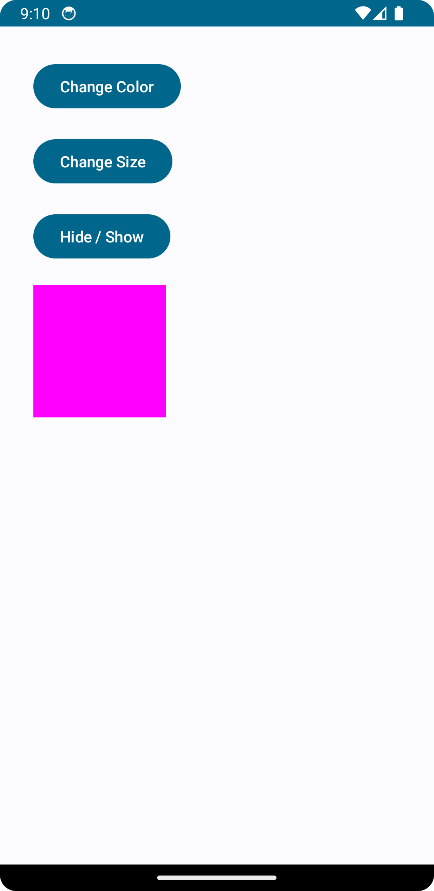
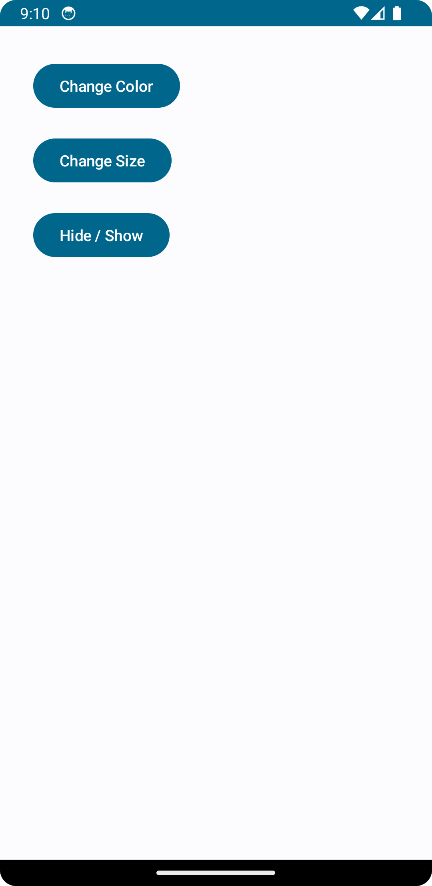
)

}

} }

## OUTPUT:

## REFERENCES:

[Jetpack Compose: Animation](https://www.youtube.com/watch?v=7yY2OocGiQU)

[](https://www.youtube.com/watch?v=7yY2OocGiQU)

# LAB 5

AIM:

Create an android application with Color, Size and Visibility Animations

## PROCEDURE:

1. **MainViewModel**: This is a ViewModel class that contains a mutable state variable loggedIn. This variable is observed by the UI and will cause recomposition whenever its value changes.
2. **MainActivity**: This is the main activity of your application. It sets up the content view using Jetpack Compose’s setContent function. The content view is a DataStoreTheme with a Surface that fills the entire screen. Inside the Surface, it calls the AppScreen composable function.
3. **AppScreen**: This composable function sets up a NavHost with two routes: “login\_screen” and “home\_screen”. The start destination of the NavHost depends on whether the user is logged in or not.
4. **LoginScreen**: This composable function displays a form with two text fields for the user to enter their name and age. When the user clicks the “Log In” button, it saves the user’s name and age using DetailStorage, navigates to the home screen, and sets loggedIn to true.
5. **HomeScreen**: This composable function displays a greeting message with the user’s name and age, which are retrieved from DetailStorage. It also displays a “Log Out” button that navigates back to the login screen and sets loggedIn to false when clicked.
6. **DetailStorage**: This class provides functions for saving and retrieving user details (name and age) using DataStore. The saveInfo function saves the user’s name and age, and the getInfo function retrieves these details.
7. **StudentDetail**: This is a data class that represents a student’s details, including their name and age.

## CODE:

MainActivity.kt

class MainViewModel : ViewModel() {  
 var loggedIn by mutableStateOf(false)  
}  
  
class MainActivity : ComponentActivity() {  
 private val viewModel by viewModels<MainViewModel>()  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContent {  
 DataStoreTheme {  
 Surface(  
 modifier = Modifier.fillMaxSize(),  
 color = MaterialTheme.colorScheme.background  
 ) {  
 AppScreen(viewModel)  
 }  
 }  
 }  
 }  
}  
  
@Composable  
fun AppScreen(viewModel: MainViewModel) {  
 val navController = rememberNavController()  
 NavHost(  
 navController = navController,  
 startDestination = if (viewModel.loggedIn) "dashboard" else "form\_page"  
 ) {  
 composable("form\_page") {  
 FormPage(navController, viewModel)  
 }  
 composable("dashboard") {  
 Dashboard(navController, viewModel)  
 }  
 }  
}

FormPage.kt

@Composable  
fun FormPage(navController: NavController, viewModel: MainViewModel) {  
 val detailStorage = DetailStorage(LocalContext.current)  
 var name by remember { mutableStateOf("") }  
 var age by remember { mutableStateOf("") }  
  
 Column(  
 modifier = Modifier  
 .fillMaxSize(),  
 horizontalAlignment = Alignment.CenterHorizontally,  
 verticalArrangement = Arrangement.Center  
 ) {  
  
 TextField(  
 value = name,  
 onValueChange = { name = it },  
 label = { Text("Name") }  
 )  
 TextField(  
 value = age,  
 onValueChange = {  
 if (it.toInt() in 1..100) {  
 age = it  
 }  
 },  
 label = { Text("Age") },  
 singleLine = true,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),  
 )  
  
 Spacer(modifier = Modifier.size(30.dp))  
  
 Button(  
 onClick = {  
 if (name.isNotBlank() && age.isNotBlank()) {  
 CoroutineScope(Dispatchers.IO).launch {  
 detailStorage.saveInfo(name, age)  
 withContext(Dispatchers.Main) {  
 navController.navigate("dashboard")  
 viewModel.loggedIn = true  
 }  
 }  
 }  
 }  
 ) {  
 Text(text = "Save Details")  
 }  
  
 }  
}

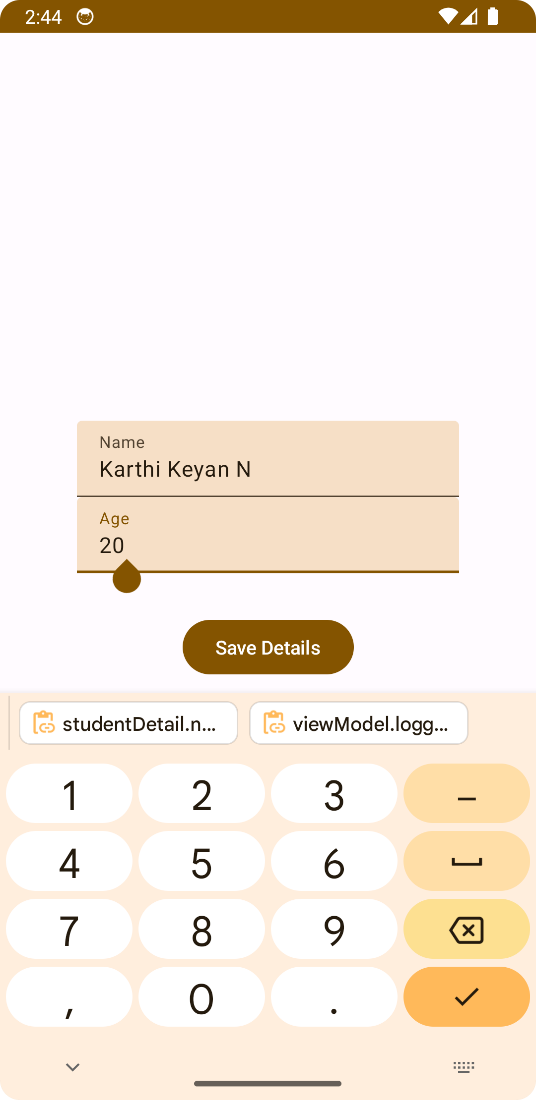
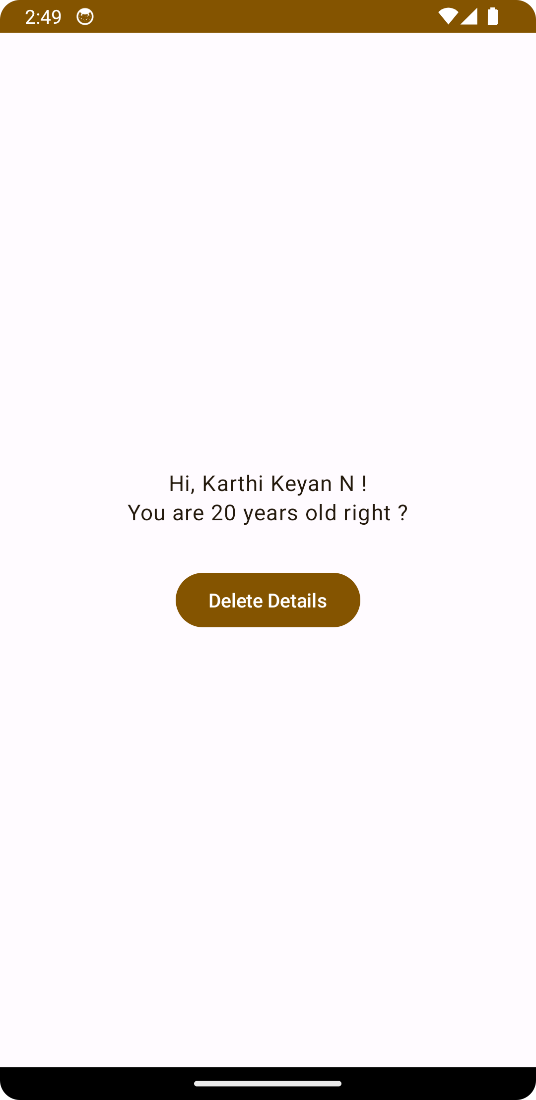
DetailStorage.kt

class DetailStorage(private val context: Context) {  
  
 companion object {  
 private val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name = "settings")  
 val NAME = stringPreferencesKey("user\_name")  
 val AGE = stringPreferencesKey("user\_age")  
 }  
  
 fun getInfo(): Flow<StudentDetail> {  
 return context.dataStore.data  
 .map {  
 StudentDetail(  
 name = it[NAME] ?: "",  
 age = it[AGE] ?: "",  
 )  
 }  
 }  
  
 suspend fun saveInfo(name: String, age: String) {  
 context.dataStore.edit { preferences ->  
 preferences[NAME] = name  
 preferences[AGE] = age  
 }  
 }  
  
}  
  
data class StudentDetail(  
 val name: String = "",  
 val age: String = "",  
)

Dashboard.kt

@Composable  
fun Dashboard(navController: NavController, viewModel: MainViewModel) {  
 val detailStorage = DetailStorage(LocalContext.current)  
 val result by detailStorage.getInfo().collectAsState(initial = StudentDetail())  
  
 Column(  
 modifier = Modifier  
 .fillMaxSize(),  
 horizontalAlignment = Alignment.CenterHorizontally,  
 verticalArrangement = Arrangement.spacedBy(10.dp)  
 ) {  
 result.let { studentDetail ->  
 Text("Hi, ${studentDetail.name} !")  
 Text("You are ${studentDetail.age} years old right ?")  
 }  
  
 Spacer(modifier = Modifier.size(30.dp))  
  
 Button(  
 onClick = {  
 navController.navigate("form\_page")  
 viewModel.loggedIn = false  
 }  
 ) {  
 Text(text = "Delete Details")  
 }  
  
 }  
}

## OUTPUT:

 n

# LAB 6:

## AIM:

To create an android application that shows the map of the current location using WebView. This application will use the Google Maps API to load the map in a WebView component and display the user’s location using the LocationManager and LocationListener classes. The application will also request the necessary permissions for accessing the internet and the location services.

## PROCEDURE:

1. @Composable: This is an annotation that indicates the function is a Composable function. Composable functions are the building blocks of Jetpack Compose UI toolkit.
2. fun BingMap() {: This defines a new Composable function named BingMap.
3. AndroidView(factory = { context ->: This creates an AndroidView, which is a Jetpack Compose function that allows you to use traditional Android views in your Compose layout. The factory parameter is a lambda function that creates the view. In this case, it’s creating a WebView.
4. WebView(context).apply {: This line creates a new WebView and applies the following settings and data to it.
5. settings.apply { javaScriptEnabled = true; setGeolocationEnabled(true) }: These lines enable JavaScript and geolocation in the WebView. JavaScript is needed to load and interact with the Bing Maps API, and geolocation is used to get the user’s current location.
6. webChromeClient = object : WebChromeClient() {...}: This sets a WebChromeClient for the WebView, which handles various events related to the WebView’s UI, such as permission requests. In this case, it’s handling geolocation permission requests.
7. loadDataWithBaseURL(null, """...""", "text/html", "UTF-8", null): This line loads the HTML data into the WebView. The HTML data includes a script tag that loads the Bing Maps API and another script tag that initializes the map.
8. var map = new Microsoft.Maps.Map(document.getElementById('myMap'), {...});: This line creates a new Bing Maps Map object and attaches it to the div with id ‘myMap’. The map object is configured with your Bing Maps API key and set to show a “Locate Me” button.
9. map.setView({ center: new Microsoft.Maps.Location(11.074800, 77.002100), zoom: 18 });: This line sets the initial view of the map to a specific location (latitude 11.074800, longitude 77.002100) and zoom level (18).
10. <div id="myMap" style="position:relative;width:100%;height:800px;"></div>: This line creates a div that will contain the map. The div is styled to take up 100% of the width of its parent container and have a height of 800 pixels.

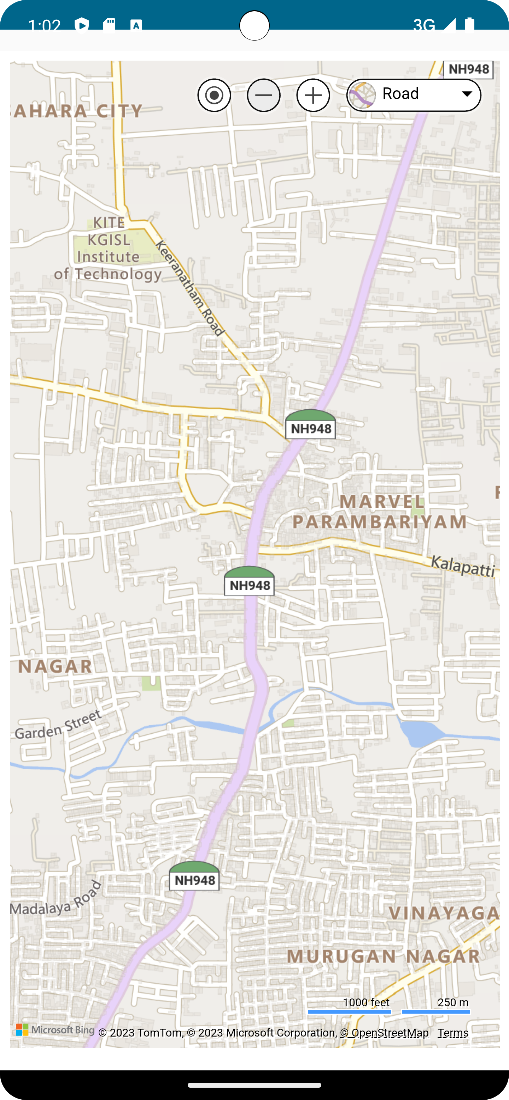
Note:-

Please replace 'YOUR\_BING\_MAPS\_KEY' with your actual Bing Maps API key before running this code.

## CODE:

package com.example.exercise5  
  
import android.os.Bundle  
import android.webkit.GeolocationPermissions  
import android.webkit.WebChromeClient  
import android.webkit.WebView  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.Surface  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.viewinterop.AndroidView  
import com.example.exercise5.ui.theme.Exercise5Theme  
  
class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContent {  
 Exercise5Theme {  
 // A surface container using the 'background' color from the theme  
 Surface(modifier = Modifier.fillMaxSize(), color = MaterialTheme.colorScheme.background) {  
 BingMap()  
 }  
 }  
 }  
 }  
}  
  
@Composable  
fun BingMap() {  
 AndroidView(factory = { context ->  
 WebView(context).apply {  
 settings.apply {  
 javaScriptEnabled = true  
 setGeolocationEnabled(true)  
 }  
 webChromeClient = object : WebChromeClient() {  
 override fun onGeolocationPermissionsShowPrompt(  
 origin: String,  
 callback: GeolocationPermissions.Callback  
 ) {  
 callback.invoke(origin, true, false)  
 }  
 }  
 loadDataWithBaseURL(null, """  
 <!DOCTYPE html>  
 <html>  
 <head>  
 <title></title>  
 <meta charset="utf-8" />  
 <script type='text/javascript'  
 src='https://www.bing.com/api/maps/mapcontrol?callback=GetMap&key=YOUR\_BING\_MAPS\_KEY'   
 async defer></script>  
 <script type='text/javascript'>  
 function GetMap()  
 {  
 var map = new Microsoft.Maps.Map(document.getElementById('myMap'), {  
 credentials: 'Agk8\_TlNOjusAEKMyoAJ6sW2CoNtPLpbYjZM\_xo5AMbGIp0J1RbmW8kKsoplzUvg',  
 showLocateMeButton: true  
 });  
 map.setView({ center: new Microsoft.Maps.Location(11.074800, 77.002100), zoom: 18 });  
 }  
 </script>  
 </head>  
 <body>  
 <div id="myMap" style="position:relative;width:100%;height:800px;"></div>  
 </body>  
 </html>  
 """, "text/html", "UTF-8", null  
 )  
 }  
 })  
}

## OUTPUT:



# LAB 7:

## AIM:

To create an android application that displays the gyroscopic sensor real-time values. This application will use the SensorManager and SensorEventListener classes to access the gyroscope sensor and get the angular speed of rotation around the x, y, and z axes. The application will display the sensor values in a Composable function using icons and text. The application will also register and unregister the sensor listener appropriately to save battery and resources.

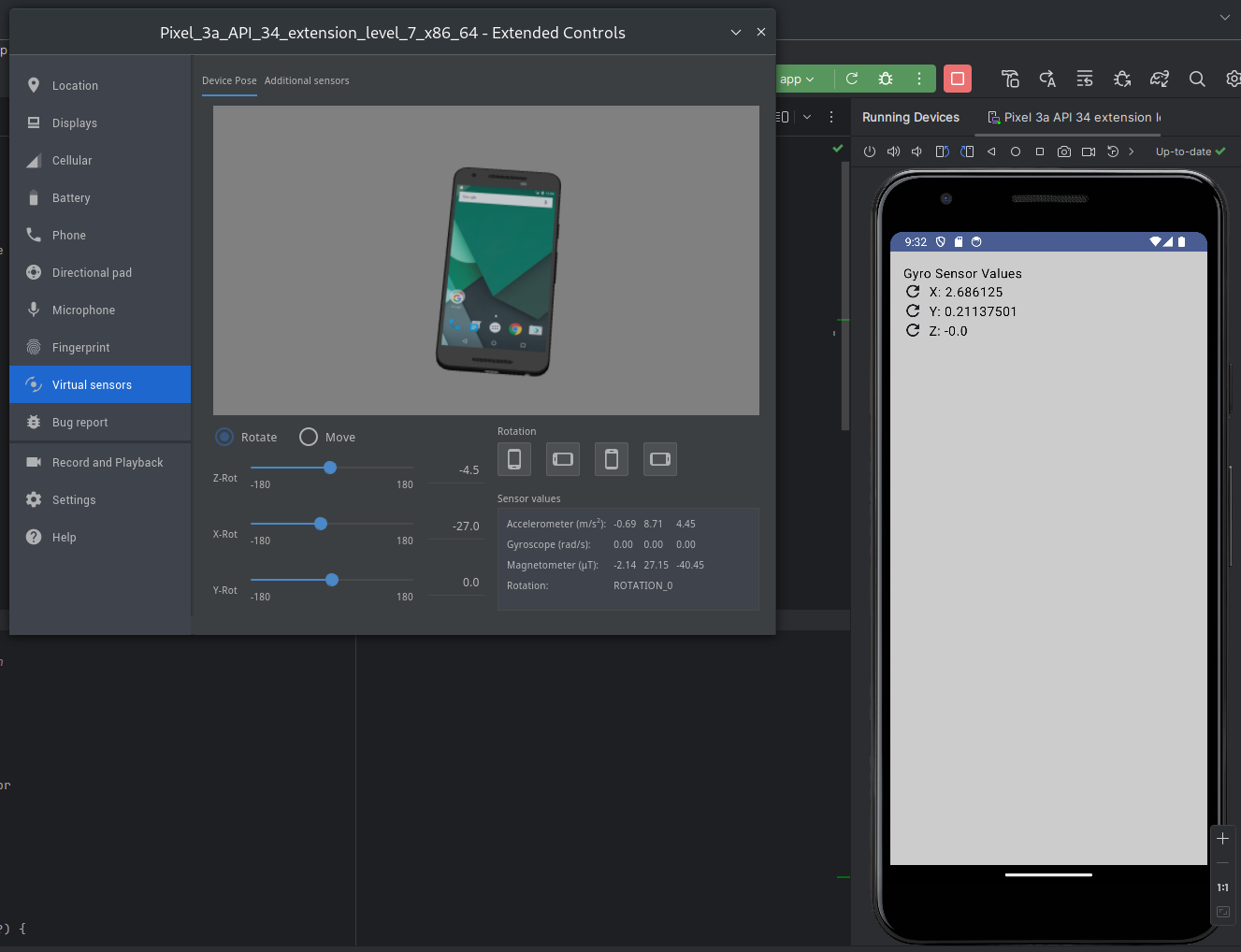
## PROCEDURE:

1. To use the gyroscope sensor in Android Compose, you can follow these steps:
2. [Create a SensorManager object and get the default gyroscope sensor from it using the TYPE\_GYROSCOPE constant1](https://code.tutsplus.com/android-sensors-in-depth-proximity-and-gyroscope--cms-28084t).
3. Create a SensorEventListener object and override the onSensorChanged method to handle the sensor events. [You can access the raw data from the gyroscope sensor in the event.values array, which contains the angular speed of rotation around the x, y, and z axes in radians per second1](https://code.tutsplus.com/android-sensors-in-depth-proximity-and-gyroscope--cms-28084t).
4. Register the listener for the gyroscope sensor using the registerListener method of the SensorManager object. [You can specify the sampling rate using one of the SENSOR\_DELAY constants1](https://code.tutsplus.com/android-sensors-in-depth-proximity-and-gyroscope--cms-28084t).
5. In your Composable function, use the remember function to create a mutable state that holds the gyroscope values. [You can update this state in the onSensorChanged method of the listener](https://code.tutsplus.com/android-sensors-in-depth-proximity-and-gyroscope--cms-28084t)[2](https://stackoverflow.com/questions/66834234/access-sensors-from-within-composables-in-jetpack-compose).
6. Use the gyroscope values in your Composable function to display them or use them for some logic. [For example, you can create a row of icons and text that show the gyroscope values along each axis](https://code.tutsplus.com/android-sensors-in-depth-proximity-and-gyroscope--cms-28084t)[2](https://stackoverflow.com/questions/66834234/access-sensors-from-within-composables-in-jetpack-compose).
7. Unregister the listener for the gyroscope sensor when you don’t need it anymore using the unregisterListener method of the SensorManager object. [You can do this in the onDispose callback of your Composable function](https://code.tutsplus.com/android-sensors-in-depth-proximity-and-gyroscope--cms-28084t)[2](https://stackoverflow.com/questions/66834234/access-sensors-from-within-composables-in-jetpack-compose).

## CODE:

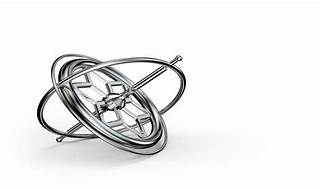
package com.example.exercise7  
  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.Surface  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.tooling.preview.Preview  
import com.example.exercise7.ui.theme.Exercise7Theme  
import android.content.Context  
import android.hardware.Sensor  
import android.hardware.SensorEvent  
import android.hardware.SensorEventListener  
import android.hardware.SensorManager  
import androidx.compose.foundation.background  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.Row  
import androidx.compose.foundation.layout.Spacer  
import androidx.compose.foundation.layout.padding  
import androidx.compose.foundation.layout.width  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.Refresh  
import androidx.compose.material3.Icon  
import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.remember  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.graphics.vector.ImageVector  
import androidx.compose.ui.platform.LocalContext  
import androidx.compose.ui.unit.dp  
  
class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContent {  
 Exercise7Theme {  
 // A surface container using the 'background' color from the theme  
 Surface(  
 modifier = Modifier.fillMaxSize(),  
 color = MaterialTheme.colorScheme.background  
 ) {  
 Greeting()  
 }  
 }  
 }  
 }  
}  
  
@Composable  
fun Greeting() {  
 GyroSensorValue()  
}  
  
@Preview(showBackground = true)  
@Composable  
fun GreetingPreview() {  
 Exercise7Theme {  
 Greeting()  
 }  
}  
  
  
@Composable  
fun GyroSensorValue() {  
 val context = LocalContext.current  
 val sensorManager = context.getSystemService(Context.SENSOR\_SERVICE) as SensorManager  
 val gyroSensor = sensorManager.getDefaultSensor(Sensor.TYPE\_GYROSCOPE)  
  
 val gyroValues = remember { mutableStateOf(Triple(0f, 0f, 0f)) }  
  
 val listener = object : SensorEventListener {  
 override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}  
  
 override fun onSensorChanged(event: SensorEvent?) {  
 event?.let {  
 gyroValues.value = Triple(it.values[0], it.values[1], it.values[2])  
 }  
 }  
 }  
  
 sensorManager.registerListener(listener, gyroSensor, SensorManager.SENSOR\_DELAY\_NORMAL)  
  
 Column(modifier = Modifier.background(Color.LightGray).padding(16.dp)) {  
 Text(text = "Gyro Sensor Values", color = Color.Black)  
 IconRow("X: ${gyroValues.value.first}", Icons.Default.Refresh)  
 IconRow("Y: ${gyroValues.value.second}", Icons.Default.Refresh)  
 IconRow("Z: ${gyroValues.value.third}", Icons.Default.Refresh)  
 }  
}  
  
@Composable  
fun IconRow(text: String, icon: ImageVector) {  
 Row(verticalAlignment = Alignment.CenterVertically) {  
 Icon(imageVector = icon, contentDescription = null)  
 Spacer(modifier = Modifier.width(8.dp))  
 Text(text = text, color = Color.Black)  
 }  
}

## OUTPUT:



# VIVA TOPICS:

1. **Android:** Android is a software package and Linux-based operating system for mobile devices such as tablet computers and smartphones.
2. **Kotlin:** Kotlin is a modern, trending programming language released in 2016 by JetBrains. It's compatible with Java and is used for various applications, especially Android apps.
3. **Framework**: A framework is a structure that you can build software on. It serves as a foundation, so you're not starting entirely from scratch. Frameworks are typically associated with a specific programming language and are suited to different types of tasks.
4. **IDE (Integrated Development Environment):** An IDE is software that combines commonly used developer tools into a compact GUI (graphical user interface) application. It includes tools like a code editor, code compiler, and code debugger with an integrated terminal.
5. **Jetpack Compose:** Jetpack Compose is Android’s recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android.
6. **Android Studio:** Android Studio is the official Integrated Development Environment (IDE) for Android app development. It's based on the powerful code editor and developer tools from IntelliJ IDEA.
7. **Android Emulator**: An Android Emulator is a tool that creates virtual Android devices (with software and hardware) on your computer. It's used for executing, debugging, and testing android applications.
8. **MaterialTheme**: This composable applies a theme to all composables within its scope. It sets the colors, typography, and shapes that will be used by the child composables.
9. **Scaffold**: This composable implements the basic Material Design visual layout structure. It provides slots for a top app bar, a floating action button, and content. It also takes care of drawing the background and managing the top inset on the screen.
10. **TopAppBar**: This composable creates a top app bar with a title. The top app bar contains navigation icons, titles, actions, and other elements important to your app.
11. **FloatingActionButton**: This composable creates a floating action button - a circular material button that lifts and displays an ink reaction on press. It is often used for a promoted action.
12. **BottomAppBar**: This composable creates a bottom app bar - a container placed at the bottom of the screen that contains primary actions and navigation controls.
13. **IconButton**: This composable creates an icon button - a clickable icon that triggers an event when clicked.
14. **Box, Row, Column**: These are layout composables used to arrange their children in specific patterns. Box places its children on top of each other, Row places its children horizontally, and Column places its children vertically.
15. **Text**: This composable displays text on the screen. It’s one of the most basic composables.
16. **Spacer**: This composable provides an empty space in your layout that can be used for padding or separating other composables.
17. **ModalNavigationDrawer**: This is a Material Design component that provides a slide-in menu for navigation within your app. It appears over the top of other content within a screen. You can use the drawerContent slot to provide a ModalDrawerSheet and provide the drawer's contents. The ModalNavigationDrawer accepts a number of additional drawer parameters, such as gesturesEnabled to toggle whether or not the drawer responds to drags.
18. **NavigationDrawerItem:** This is a composable function that creates an item in the navigation drawer. Each NavigationDrawerItem represents a destination in your app that users can navigate to. It typically includes a label and an icon, and an onClick event to handle navigation.
19. **NavController:** This is the central API for the Navigation component in Jetpack Compose. It manages app navigation within a NavHost. The NavController keeps track of the back stack of composables that make up the screens in your app and the state of each screen. You can create a NavController by using the rememberNavController() method in your composable.
20. **NavHost**: This is a composable that hosts a navigation graph. It takes in a NavController and a start destination. The start destination is the route of the first composable that should be displayed. The NavHost links the NavController with a navigation graph that specifies the composable destinations that you should be able to navigate between.
21. composable Function: This function is used to define a destination in your navigation graph. It takes in a route and a trailing lambda where you define what UI should be displayed at this destination4. Each composable destination in your navigation graph is associated with a route.
22. **AnimatedVisibility:** This composable animates visibility changes of its content, either fading it in and expanding its height/width from 0 when becoming visible or fading it out and reducing its height/width to 0 when becoming invisible.
23. WebView is a key component in Android that allows developers to display webpages and other web content within an app. It’s an extension of Android’s View class that lets you display web pages as a part of your activity layout



1. [Gyro sensors, or gyroscopes, measure angular velocity, which is the change in the rotational angle of the object per unit of time](https://www.bing.com/aclk?ld=e83VIqelo_nWVfqVOb8EdIzDVUCUxFB3teQEorBV989PGwpwhV9oZrAeX8qoUEg_7qWtBOo4WnnaTPEAZd24tz8ub1EYVHrcP2jrZ7COcnKUyuy-FE13uoz8lzKYxY-hB9bwSuQ-i-WkU72EyBqxcn_lKAmHRzkDHy3xFZHRb0Pq4-CwSQ&u=aHR0cHMlM2ElMmYlMmZ3d3cuYW1hem9uLmluJTJmcyUyZiUzZmllJTNkVVRGOCUyNmtleXdvcmRzJTNkZ3lybyUyYnNlbnNvciUyYm1vYmlsZSUyNmluZGV4JTNkYXBzJTI2dGFnJTNkbXNuZGVza3N0ZGluLTIxJTI2cmVmJTNkcGRfc2xfdTFoM3V4c2ZiX2UlMjZhZGdycGlkJTNkMTMyNjAxMjYyOTMyNjYwNCUyNmh2YWRpZCUzZDgyODc2MDU5NjE4NDQzJTI2aHZuZXR3JTNkbyUyNmh2cW10JTNkZSUyNmh2Ym10JTNkYmUlMjZodmRldiUzZGMlMjZodmxvY2ludCUzZCUyNmh2bG9jcGh5JTNkMTQzOTU2JTI2aHZ0YXJnaWQlM2Rrd2QtODI4NzY2NzYwNjkzMzIlM2Fsb2MtOTAlMjZoeWRhZGNyJTNkMjY3NTRfMjc5OTk5NQ&rlid=69b62cf289fa1904c40b49aabe1cd8e6)[1](https://bing.com/search?q=gyro+sensors+in+mobile).
2. [Gyro sensors help devices know which way is up and sense movement and orientation in three-dimensional space](https://www.bing.com/aclk?ld=e83VIqelo_nWVfqVOb8EdIzDVUCUxFB3teQEorBV989PGwpwhV9oZrAeX8qoUEg_7qWtBOo4WnnaTPEAZd24tz8ub1EYVHrcP2jrZ7COcnKUyuy-FE13uoz8lzKYxY-hB9bwSuQ-i-WkU72EyBqxcn_lKAmHRzkDHy3xFZHRb0Pq4-CwSQ&u=aHR0cHMlM2ElMmYlMmZ3d3cuYW1hem9uLmluJTJmcyUyZiUzZmllJTNkVVRGOCUyNmtleXdvcmRzJTNkZ3lybyUyYnNlbnNvciUyYm1vYmlsZSUyNmluZGV4JTNkYXBzJTI2dGFnJTNkbXNuZGVza3N0ZGluLTIxJTI2cmVmJTNkcGRfc2xfdTFoM3V4c2ZiX2UlMjZhZGdycGlkJTNkMTMyNjAxMjYyOTMyNjYwNCUyNmh2YWRpZCUzZDgyODc2MDU5NjE4NDQzJTI2aHZuZXR3JTNkbyUyNmh2cW10JTNkZSUyNmh2Ym10JTNkYmUlMjZodmRldiUzZGMlMjZodmxvY2ludCUzZCUyNmh2bG9jcGh5JTNkMTQzOTU2JTI2aHZ0YXJnaWQlM2Rrd2QtODI4NzY2NzYwNjkzMzIlM2Fsb2MtOTAlMjZoeWRhZGNyJTNkMjY3NTRfMjc5OTk5NQ&rlid=69b62cf289fa1904c40b49aabe1cd8e6)[1](https://bing.com/search?q=gyro+sensors+in+mobile).
3. [Gyro sensors are used for applications such as immersive gaming, augmented reality, navigation, and camera stabilization1](https://bing.com/search?q=gyro+sensors+in+mobile)[2](https://www.techaheadcorp.com/knowledge-center/how-gyroscope-sensor-work-in-smartphone/).
4. Gyro sensors work by using microscopic crystal structures that become stressed due to rotational forces. [The sensor then interprets the voltage coming from the crystals to figure out how fast the phone is rotating and which direction it is pointing in](https://www.bing.com/aclk?ld=e83VIqelo_nWVfqVOb8EdIzDVUCUxFB3teQEorBV989PGwpwhV9oZrAeX8qoUEg_7qWtBOo4WnnaTPEAZd24tz8ub1EYVHrcP2jrZ7COcnKUyuy-FE13uoz8lzKYxY-hB9bwSuQ-i-WkU72EyBqxcn_lKAmHRzkDHy3xFZHRb0Pq4-CwSQ&u=aHR0cHMlM2ElMmYlMmZ3d3cuYW1hem9uLmluJTJmcyUyZiUzZmllJTNkVVRGOCUyNmtleXdvcmRzJTNkZ3lybyUyYnNlbnNvciUyYm1vYmlsZSUyNmluZGV4JTNkYXBzJTI2dGFnJTNkbXNuZGVza3N0ZGluLTIxJTI2cmVmJTNkcGRfc2xfdTFoM3V4c2ZiX2UlMjZhZGdycGlkJTNkMTMyNjAxMjYyOTMyNjYwNCUyNmh2YWRpZCUzZDgyODc2MDU5NjE4NDQzJTI2aHZuZXR3JTNkbyUyNmh2cW10JTNkZSUyNmh2Ym10JTNkYmUlMjZodmRldiUzZGMlMjZodmxvY2ludCUzZCUyNmh2bG9jcGh5JTNkMTQzOTU2JTI2aHZ0YXJnaWQlM2Rrd2QtODI4NzY2NzYwNjkzMzIlM2Fsb2MtOTAlMjZoeWRhZGNyJTNkMjY3NTRfMjc5OTk5NQ&rlid=69b62cf289fa1904c40b49aabe1cd8e6)[1](https://bing.com/search?q=gyro+sensors+in+mobile).
5. [Gyro sensors are often combined with accelerometer sensors, which measure linear acceleration, to provide more accurate and robust motion sensing](https://www.bing.com/aclk?ld=e83VIqelo_nWVfqVOb8EdIzDVUCUxFB3teQEorBV989PGwpwhV9oZrAeX8qoUEg_7qWtBOo4WnnaTPEAZd24tz8ub1EYVHrcP2jrZ7COcnKUyuy-FE13uoz8lzKYxY-hB9bwSuQ-i-WkU72EyBqxcn_lKAmHRzkDHy3xFZHRb0Pq4-CwSQ&u=aHR0cHMlM2ElMmYlMmZ3d3cuYW1hem9uLmluJTJmcyUyZiUzZmllJTNkVVRGOCUyNmtleXdvcmRzJTNkZ3lybyUyYnNlbnNvciUyYm1vYmlsZSUyNmluZGV4JTNkYXBzJTI2dGFnJTNkbXNuZGVza3N0ZGluLTIxJTI2cmVmJTNkcGRfc2xfdTFoM3V4c2ZiX2UlMjZhZGdycGlkJTNkMTMyNjAxMjYyOTMyNjYwNCUyNmh2YWRpZCUzZDgyODc2MDU5NjE4NDQzJTI2aHZuZXR3JTNkbyUyNmh2cW10JTNkZSUyNmh2Ym10JTNkYmUlMjZodmRldiUzZGMlMjZodmxvY2ludCUzZCUyNmh2bG9jcGh5JTNkMTQzOTU2JTI2aHZ0YXJnaWQlM2Rrd2QtODI4NzY2NzYwNjkzMzIlM2Fsb2MtOTAlMjZoeWRhZGNyJTNkMjY3NTRfMjc5OTk5NQ&rlid=69b62cf289fa1904c40b49aabe1cd8e6)[1](https://bing.com/search?q=gyro+sensors+in+mobile)[2](https://www.techaheadcorp.com/knowledge-center/how-gyroscope-sensor-work-in-smartphone/).
6. [Gyro sensors are based on the invention of mechanical gyroscopes by Leon Foucault in 1852, but they have become smaller and cheaper due to advancements in micro-electro-mechanical systems (MEMS) technology](https://www.bing.com/aclk?ld=e83VIqelo_nWVfqVOb8EdIzDVUCUxFB3teQEorBV989PGwpwhV9oZrAeX8qoUEg_7qWtBOo4WnnaTPEAZd24tz8ub1EYVHrcP2jrZ7COcnKUyuy-FE13uoz8lzKYxY-hB9bwSuQ-i-WkU72EyBqxcn_lKAmHRzkDHy3xFZHRb0Pq4-CwSQ&u=aHR0cHMlM2ElMmYlMmZ3d3cuYW1hem9uLmluJTJmcyUyZiUzZmllJTNkVVRGOCUyNmtleXdvcmRzJTNkZ3lybyUyYnNlbnNvciUyYm1vYmlsZSUyNmluZGV4JTNkYXBzJTI2dGFnJTNkbXNuZGVza3N0ZGluLTIxJTI2cmVmJTNkcGRfc2xfdTFoM3V4c2ZiX2UlMjZhZGdycGlkJTNkMTMyNjAxMjYyOTMyNjYwNCUyNmh2YWRpZCUzZDgyODc2MDU5NjE4NDQzJTI2aHZuZXR3JTNkbyUyNmh2cW10JTNkZSUyNmh2Ym10JTNkYmUlMjZodmRldiUzZGMlMjZodmxvY2ludCUzZCUyNmh2bG9jcGh5JTNkMTQzOTU2JTI2aHZ0YXJnaWQlM2Rrd2QtODI4NzY2NzYwNjkzMzIlM2Fsb2MtOTAlMjZoeWRhZGNyJTNkMjY3NTRfMjc5OTk5NQ&rlid=69b62cf289fa1904c40b49aabe1cd8e6)[1](https://bing.com/search?q=gyro+sensors+in+mobile)[2](https://www.techaheadcorp.com/knowledge-center/how-gyroscope-sensor-work-in-smartphone/).

Made with ❤️ by [KarthiDreamr](https://github.com/KarthiDreamr/)