Week - 2 Quiz Solutions

1. Which of the following statements about SharedPreferences in Android is NOT true?
   Ans: SharedPreferences are the best choice for storing large amounts of complex data (like images or videos).

2. In Jetpack Compose, what is the primary function of the Scaffold composable?
   Ans: It provides a foundation for building user interfaces with common elements like content area, snackbars, and floating action buttons.

3. In Jetpack Compose, which annotation is used to mark a composable function for previewing within Android Studio?
   Ans: @Preview

4. Which method is typically called to save the state of the UI when a configuration change occurs ?
   Ans: onSave

5. In Kotlin, which of the following methods is used to write bytes of data to a file?
   Ans: openFile(mode: String) for writing

The code for the task is also given in the Week -2 folder in the GitHub repository.

In the previous week, we added the floating action button, but we didn't add any functionality to it. The functionality can be added as shown below the code snippet.

```
this: ColumnScope
androidx.compose.material.FloatingActionButton(
    onClick = {
        mContext.startActivity(Intent(mContext, AddItem::class.java))
    },
    backgroundColor = Color( red: 116,  green: 201,  blue: 49),
    elevation = FloatingActionButtonDefaults.elevation(
        defaultElevation = 20.dp
    )
) {
    Icon(Icons.Filled.Add,  contentDescription: "Floating action button.")
}
```

The snippet of the code given for the task has been completed here. For saving food items to a file, we have defined a "**Food**" data class and a function called "**writeFoodItemToFile**". The function writes the "**Food**" object's name and price to a file in the app's files directory. This implementation should be added to your project to enable file storage functionality.

```
data class Food(
    val name: String,
    val canteen: String,
    val price: String,
    val type: FoodType,
    val cuisine: String
)

p3rp1ex3d
fun writeFoodItemToFile(context: Context, food: Food, fileName: String) {
    val file = File(context.filesDir, fileName)
    val foodItemString = "${food.name},${food.price.toString()}\n"
    FileOutputStream(file, append: true).bufferedWriter().use { writer ->
        writer.write(foodItemString)
    }
}
```

The given code defines an **"enum class"** called **"FoodType"** with three values: *BEVERAGE, DESSERT,* and *MAIN_COURSE*. It also includes a **"@Composable"** function named **"AddFoodScreen"** that provides a user interface for adding food items. This UI consists of several input fields (name, canteen, price, type, and cuisine) and a save button.

```
enum class FoodType {
    BEVERAGE,
    DESSERT,
    MAIN_COURSE
}
```

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddFoodScreen(
    onSave: (Food) -> Unit
) {
    var name by remember { mutableStateOf( value: "") }
    var canteen by remember { mutableStateOf( value: "") }
    var price by remember { mutableStateOf( value: "") }
    var type by remember { mutableStateOf(FoodType.MAIN_COURSE) }
    var cuisine by remember { mutableStateOf( value: "") }
    var expanded by remember { mutableStateOf( value: false) }
    val context = LocalContext.current
```

The dropdown menu in **AddFoodScreen** lets users select a food type from options like *BEVERAGE, DESSERT,* and *MAIN_COURSE.* It uses **ExposedDropdownMenuBox** with an **OutlinedTextField** to display the selected type, and **ExposedDropdownMenu** to show and update the selection when expanded.

```
ExposedDropdownMenuBox(
    expanded = expanded,
    onExpandedChange = { expanded = !expanded }
) { this: ExposedDropdownMenuBoxScope
    OutlinedTextField(
        value = type.name,
        onValueChange = {},
        readOnly = true,
        label = { Text( text: "Type") },
        trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded) },
        modifier = Modifier.menuAnchor(),
        colors = outlineColor
    )
    ExposedDropdownMenu(
        expanded = expanded,
        onDismissRequest = { expanded = false }
    ) { this: ColumnScope
        FoodType.values().forEach { selectedType ->
            DropdownMenuItem(
                text = { Text(selectedType.name) },
                onClick = {
                    type = selectedType
                    expanded = false
                }
            )
        }
    }
}
```

When the button is clicked, the input data is validated and a **“Food”** object is created. This object is then saved to a file using the **“writeFoodItemToFile”** function, and the user is redirected to the **“MainActivity”**.

```
Button(
    onClick = {
        if (name.isNotBlank() && canteen.isNotBlank() && price.isNotBlank() && cuisine.isNotBlank()) {
            val foodItem = Food(
                name = name,
                canteen = canteen,
                price = price,
                type = type,
                cuisine = cuisine
            )
            writeFoodItemToFile(context, foodItem,  fileName: "food.txt")
            onSave(foodItem)
            context.startActivity(Intent(context, MainActivity::class.java))
        }
    },
    colors = ButtonDefaults.buttonColors(
        containerColor = Color( red: 116,  green: 201,  blue: 49),
        contentColor = Color.White
    )
) {  this: RowScope
    Text( text: "Save")
}
```

It is important to note that this isn't the only way to build the app. There are many ways in which you can choose to proceed and this is just one of them.