Week - 4 Quiz Solutions

1. **What does the remember function do in the given Kotlin snippet?**
   Ans: It saves the state of a variable across recompositions.

2. **In the Slider component, what does the steps parameter define?**
   Ans: The number of discrete intervals the slider can take.

3. **What does the onValueChange parameter in the Slider component do?**
   Ans: It is a callback that gets invoked when the slider's value changes.

4. **What is the role of the Modifier in Jetpack Compose?**
   Ans: It is used to modify existing composables.

5. **How can you trigger a recomposition in Jetpack Compose?**
   Ans: By changing the state of a mutable StateOf variable.

The code for the task is also given in the Week -4 folder in the GitHub repository.

In this week, we'll code the final part of the app that is the fun-section of the app. It is shown in below the code snippet.

The **FunScreen** composable includes a **randomAngle** function to generate random angles and a **color_bg** function to change the background color based on a code.

```
Kuru07 +1 *
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun FunScreen() {
    var sliderPosition by remember { mutableFloatStateOf( value: 3F) }
    var switchClick by remember { mutableFloatStateOf( value: 0F) }
    var color_choose by remember { mutableStateOf(Color.White) }
    var targetAngle by remember { mutableStateOf( value: 0f) }
    var cod by remember { mutableIntStateOf( value: 1) }


    fun randomAngle(): Float {
        return (0 ≤ .. ≤ 360).random().toFloat()
    }

    @Composable
    fun color_bg(int: Int) {
        when (int) {
            1 -> color_choose = colorResource(R.color.blue)
            2 -> color_choose = colorResource(R.color.red)
            3 -> color_choose = colorResource(R.color.green)
            4 -> color_choose = colorResource(R.color.orange)
            5 -> color_choose = colorResource(R.color.purle)
            6 -> color_choose = colorResource(R.color.yellow)
            7 -> color_choose = colorResource(R.color.pink)
            8 -> color_choose = colorResource(R.color.gold)

        }
    }
```

The below snippet is used to display a text prompt and a spinner wheel based on the **switchClick** value. The spinner wheel's segments and point align is done based on the **sliderPosition**.

```
Column(
    modifier = Modifier.(...),
    verticalArrangement = Arrangement.Center,
    horizontalAlignment = Alignment.CenterHorizontally
) { this: ColumnScope
    Text(
        modifier = Modifier.(...),
        text = "EACH PERSON SHOULD \nPICK A COLOUR",
        fontWeight = FontWeight.Bold,
        fontSize = 20.sp,
        textAlign = TextAlign.Center,
        color = Color.Black
    )
    Box(modifier = Modifier.padding(0.dp)){ this: BoxScope
        if (switchClick == 0F){
            when (sliderPosition) {
                3F ->{
                    Box{ this: BoxScope
                        Image(
                            painter = painterResource(R.drawable.wheel_3),
                            contentDescription = null
                        )
                        Box(contentAlignment = Alignment.Center,
                            modifier = Modifier.align(Alignment.Center)){ this: BoxScope
                            Image(
                                painter = painterResource(id = R.drawable.point),
                                modifier = Modifier.align(Alignment.Center),
                                contentDescription = null
                            )
                        }
                    }

                }

                4F -> {...}

                5F -> {...}

                6F -> {...}

                7F -> {...}
```

If `**switchClick**` is greater than 0, the displayed spinner wheel is selected based on the `**sliderPosition**`. The `**cod**` value determines the rotation angle of the point image within the wheel. Different rotations are applied based on the value of `**cod**` to align the point with specific segments of the wheel.

```
if (switchClick>0f) {
    when (sliderPosition) {
        3F -> Box{ this: BoxScope
            Image(
                painter = painterResource(R.drawable.wheel_3),
                contentDescription = null
            )
            Box(contentAlignment = Alignment.Center,
                modifier = Modifier.align(Alignment.Center)) { this: BoxScope
                if (cod == 1)
                    Image(
                        painter = painterResource(id = R.drawable.point),
                        modifier = Modifier
                            .align(Alignment.Center)
                            .rotate( degrees: 130F),
                        contentDescription = null,
                    )
                if (cod == 2)
                    Image(
                        painter = painterResource(id = R.drawable.point),
                        modifier = Modifier
                            .align(Alignment.Center)
                            .rotate( degrees: 230F),
                        contentDescription = null,
                    )
                if (cod == 3)
                    Image(
                        painter = painterResource(id = R.drawable.point),
                        modifier = Modifier
                            .align(Alignment.Center)
                            .rotate( degrees: 40f),
                        contentDescription = null,
                    )
            }
        }

        4F -> Box{...}

        5F -> Box{...}

        6F -> Box{...}
```

The `**Slider**` component lets users select a value within a range, adjusting `**sliderPosition**`. The `**Button**` labeled "**SPIN**" triggers actions such as generating a random angle and updating a counter when clicked. If the `**switchClick**` value exceeds 0, it invokes the `**PayBill()**` based on the slider's value and updates the background color using `**color_bg()**`.

```
        Spacer(modifier = Modifier.padding(13.dp))
        Slider(
            value = sliderPosition,
            onValueChange = { it: Float
                sliderPosition = it.roundToInt().toFloat()
                switchClick = 0F
                color_choose = Color.White
            },
            colors = SliderDefaults.colors(
                thumbColor = Color.LightGray,
                activeTrackColor = colorResource(R.color.darkslider),
                inactiveTrackColor = colorResource(R.color.blueslider)
            ),
            thumb = { it: SliderPositions
                Image(painterResource(id = R.drawable.pin), contentDescription: "contentDescription")
            },
            steps = 4,
            valueRange = 3F ≤ .. ≤ 8F,
            modifier = Modifier
                .fillMaxWidth()
        )
        Spacer(modifier = Modifier.padding(13.dp))
        Text(text = sliderPosition.toInt().toString() + " Persons", fontSize = 20.sp, fontWeight =

        Spacer(modifier = Modifier.padding(13.dp))

        Button(
            onClick = {
                targetAngle = randomAngle()
                switchClick++ },
            colors = ButtonDefaults.buttonColors(containerColor = Color.Black) )
        { this: RowScope
            Text(text = "SPIN", fontWeight = FontWeight.Bold, color = Color.White)
        }
        Spacer(modifier = Modifier.padding(13.dp))
        if (switchClick > 0) {
            cod=PayBill(numberOfPeople = sliderPosition.toInt(), reset = false)
            color_bg(int = cod)
        }

    }

}
```

The **PayBill** function is used to select a random discrete number between the selected range of numbers and the colourChoose function is invoked in the after the number is randomized. The **colourChoose** function displays the output text by assigning each number with a specific output.

```
Kuru07
@Composable
fun PayBill(numberOfPeople: Int, reset: Boolean = false) : Int {
    var chosenNumber by remember { mutableIntStateOf((1 ≤ .. ≤ numberOfPeople).random()) }
    if (reset) {
        chosenNumber = (1 ≤ .. ≤ numberOfPeople).random()
    }
    Column { this: ColumnScope
        colourChoose(chosenNumber)
    }
    return chosenNumber
}

Kuru07
@Composable
fun colourChoose(num: Int) {
    when (num) {
        1 -> Text(text = "Blue must pay the Bill", color = Color.White, fontWeight = FontWeight.Bold , fontSize = 20.sp)
        2 -> Text(text = "Red must pay the Bill", color = Color.White, fontWeight = FontWeight.Bold, fontSize = 20.sp)
        3 -> Text(text = "Green must pay the Bill", color = Color.Black, fontWeight = FontWeight.Bold, fontSize = 20.sp)
        4 -> Text(text = "Orange must pay the Bill", color = Color.White, fontWeight = FontWeight.Bold, fontSize = 20.sp)
        5 -> Text(text = "Purple must pay the Bill", color = Color.White, fontWeight = FontWeight.Bold, fontSize = 20.sp)
        6 -> Text(text = "Yellow  must pay the Bill", color = Color.Black, fontWeight = FontWeight.Bold, fontSize = 20.sp)
        7 -> Text(text = "Pink must pay the Bill", color = Color.Black, fontWeight = FontWeight.Bold, fontSize = 20.sp)
        8 -> Text(text = "Gold must pay the Bill", color = Color.Black, fontWeight = FontWeight.Bold, fontSize = 20.sp)
    }
}
```