Week - 3 Quiz Solutions

1. What is the primary benefit of a LazyList in kotlin?
   Ans: It optimizes memory usage by loading elements only when needed.

2. Which technique is MOST LIKELY used in a LazyList to ensure that elements are only created when they are actually accessed?
   Ans: Lazy initialization

3. Which of the following methods is the MOST appropriate way to write a string of text to an internal storage file within your app's private directory?
   Ans: openFileOutput(fileName, mode).write(data.toByteArray())

4. Which approach is MOST appropriate for writing a string named "data" to an internal storage file named "my_file.txt" within your app's private directory?
   Ans: Opening an output stream with openFileOutput, converting the string to a byte array, and writing it to the stream

5. Which method is typically used to access an element by its index within a List or array?
   Ans: get(index)

The code for the task is also given in the Week - 3 folder in the GitHub repository.

The function ListItem works as the template for the LazyList element. It helps define how each item of the list should look like. Here we have created a rectangular box with a green border and white background with three elements in it namely a checkbox, a text for name of the food item and a text box for the price placed in a horizontal fashion. For every item added to this list, it will follow the same template in terms of its looks but the data differs.

```
@Composable
fun ListItem(item: FoodItem, isChecked: Boolean, onCheckedChange: (Boolean) -> Unit) {
    Surface(
        color = Color.White,
        border = BorderStroke(3.dp, Color(116, 201, 49)),
        modifier = Modifier.padding(vertical = 10.dp, horizontal = 20.dp)
    ) {
        Row(verticalAlignment = Alignment.CenterVertically) {
            Checkbox(
                checked = isChecked,
                colors = CheckboxDefaults.colors(checkedColor = Color.Black),
                onCheckedChange = onCheckedChange
            )
            Column(
                modifier = Modifier
                    .padding(horizontal = 10.dp, vertical = 15.dp)
                    .fillMaxWidth()
            ) {
                Row(
                    horizontalArrangement = Arrangement.SpaceBetween,
                    modifier = Modifier.fillMaxWidth()
                ) {
                    Column(modifier = Modifier.padding(start = 5.dp)) {
                        Text(
                            text = item.name,
                            color = Color.Black,
                            style = MaterialTheme.typography.headlineMedium.copy(
                                fontFamily = Klee, fontSize = 25.sp
                            )
                        )
                    }
                    Column {
                        Text(
                            modifier = Modifier.padding(end = 10.dp),
                            text = item.price.toString(),
                            color = Color.Black,
                            style = MaterialTheme.typography.headlineMedium.copy(
                                fontWeight = FontWeight.ExtraBold, fontFamily = Klee, fontSize = 27.sp
                            )
                        )
                    }
                }
            }
        }
    }
}
```

This is the place where we handle the lazycolumn. It basically sets up the lazycolumn and mentions the template that needs to be used which in this case is List Item. Another functionality here is that of the checkbox which changes state based on the user action.

```
@Composable
fun RecyclerView(names: List<FoodItem>, selectedItems: Set<FoodItem>, onItemCheckedChange: (FoodItem, Boolean) -> Unit) {
    LazyColumn(modifier = Modifier.padding(vertical = 4.dp), verticalArrangement = Arrangement.Top) {
        items(items = names) { item ->
            ListItem(
                item = item,
                isChecked = selectedItems.contains(item),
                onCheckedChange = { isChecked ->
                    onItemCheckedChange(item, isChecked)
                }
            )
        }
    }
}
```

The FoodScreen is where we make a call to the RecyclerView() function and pass the arguments like name, price etc.

```
@Composable
fun FoodScreen(foodItems: List<FoodItem>, onAddItem: (FoodItem) -> Unit) {
    val mContext = LocalContext.current
    var selectedItems by remember { mutableStateOf(setOf<FoodItem>()) }
    val totalPrice by remember { derivedStateOf { selectedItems.sumOf { it.price } } }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(color = Color.White),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.Start,
    ) {
        Text(
            text = "Total Price: $totalPrice",
            modifier = Modifier.padding(top = 30.dp, bottom = 10.dp, start = 25.dp),
            fontSize = 20.sp,
            color = Color.Black
        )
        RecyclerView(
            names = foodItems,
            selectedItems = selectedItems,
            onItemCheckedChange = { item, isChecked ->
                selectedItems = if (isChecked) {
                    selectedItems + item
                } else {
                    selectedItems - item
                }
            }
        )
    }
```

With this, the food section of the app is fully functional and we can proceed to finish the app in week - 4.