Mobile Application Development

Week4　First Android App by Jetpack Compose

Yi SUN
Kobe Institute of Computing

What is Jetpack Compose

- Jetpack Compose is Android's modern toolkit for building native user interfaces using a declarative approach.
- Instead of defining your UI in XML files like in traditional Android development, with Jetpack Compose you write UI code directly in Kotlin.
- This makes it easier to create dynamic, responsive, and intuitive interfaces. Here's a beginner-friendly breakdown:

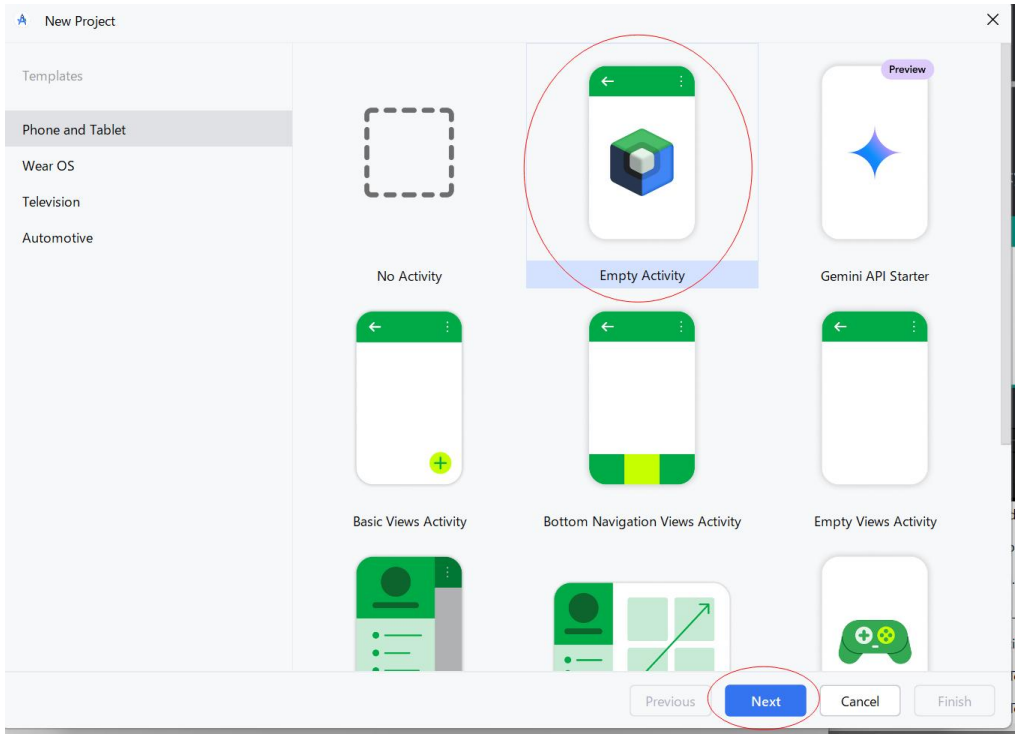
The keys of Jetpack Compose

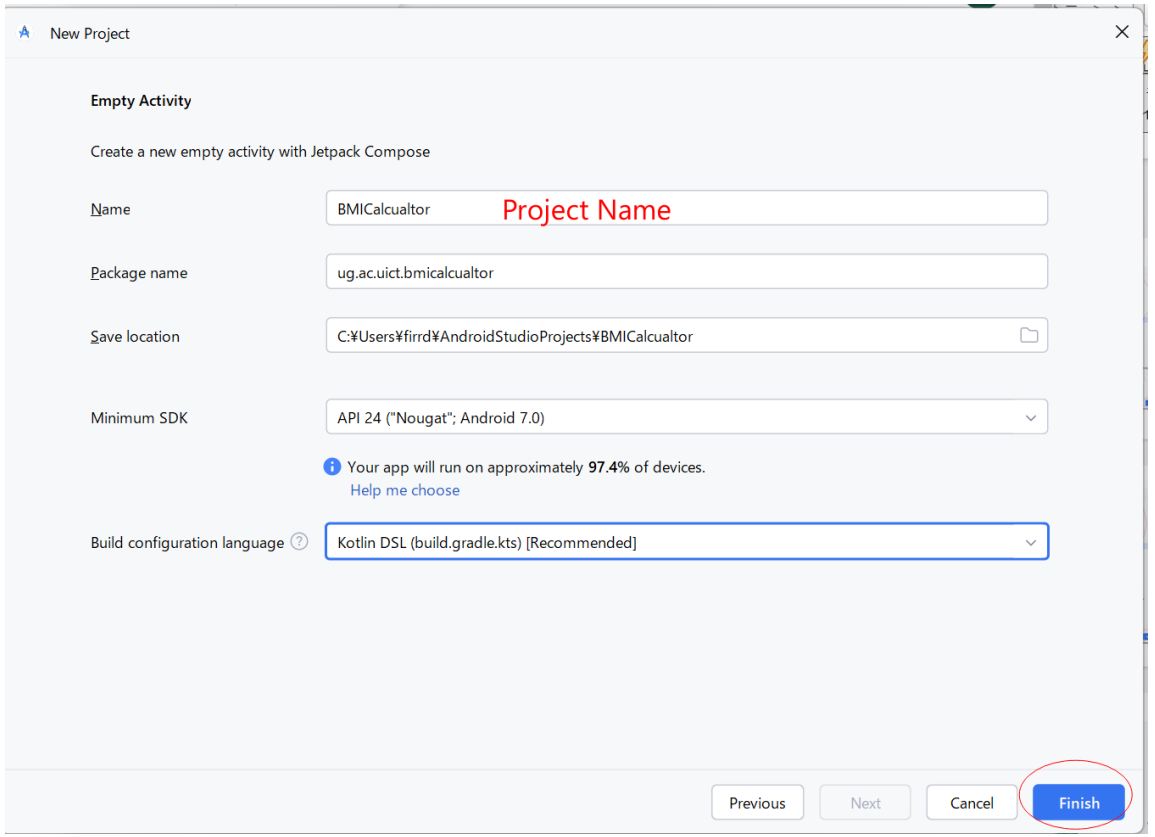
- Declarative UI
  - Instead of describing how to update the UI (imperative), you simply describe what the UI should look like based on its state. When the state changes, Compose automatically updates the parts of the UI that need to change.

- Component-driven
  - The UI is built from small, reusable functions called composables. These functions are annotated with @Composable and represent individual parts of your UI.

The first Android App by Jetpack compose

1. Open AndroidStudio
2. Select `File > New > NewProject`
3. Choose `Empty Activity`
4. Name: BMICalculator





@Composable

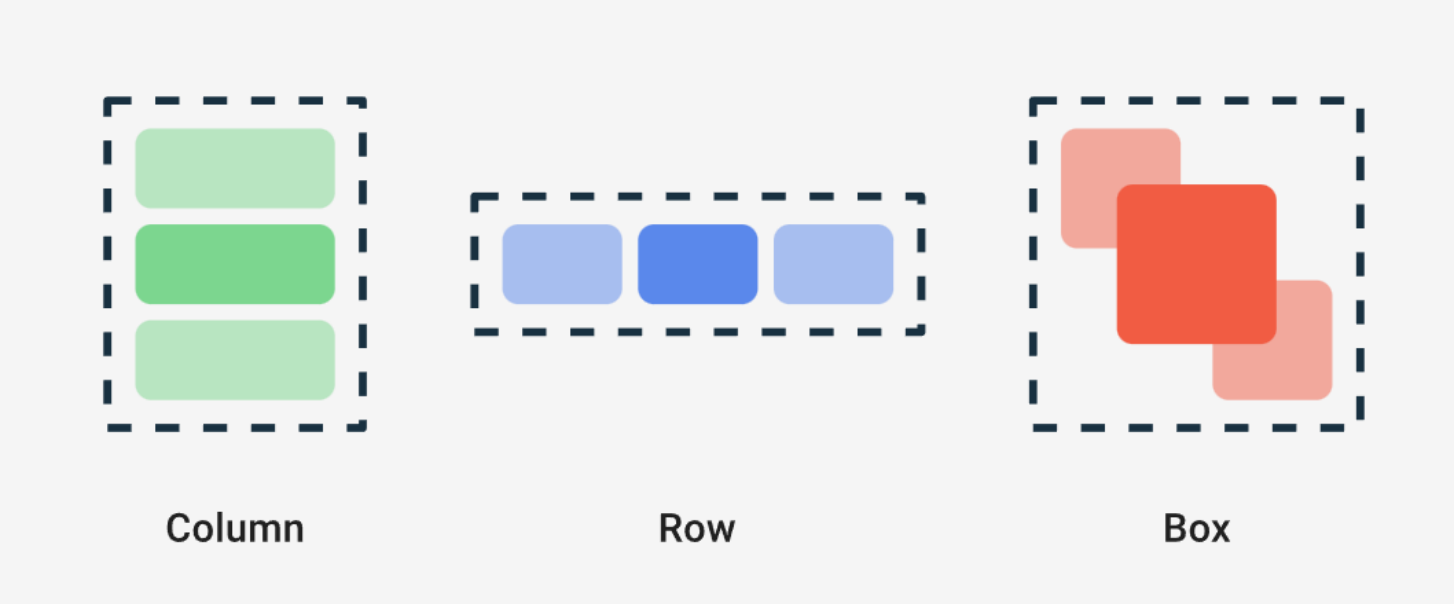
- **@Composable is an Annotation:**
  - It Enables Recomposition:
    Changes in the underlying data trigger a re-execution of the composable function to update the UI.
- **It Supports Modularity:**
  - You can build complex UIs by composing small, reusable functions, each responsible for a part of the screen.
- **It Simplifies UI Development:**
  - By leveraging Kotlin's language features and Compose's runtime optimizations, you write less boilerplate code and focus on describing what the UI should look like.

Material Design and scaffold structure

- Material 3 is the latest version of Google's open-source design system. Design and build beautiful, usable products with Material 3.
- https://m3.material.io/
- https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary.html
- In Material Design, a scaffold is a fundamental structure that provides a standardized platform for complex user interfaces. It holds together different parts of the UI, such as app bars and floating action buttons, giving apps a coherent look and feel.
- https://developer.android.com/develop/ui/compose/components/scaffold

Building Layouts by @Composable

- Compose offers a variety of layout components (also called “layout composables”) to arrange your UI elements.
  - **Column**: Stacks children vertically.
  - **Row**: Places children horizontally.
  - **Box**: Overlays children on top of each other.

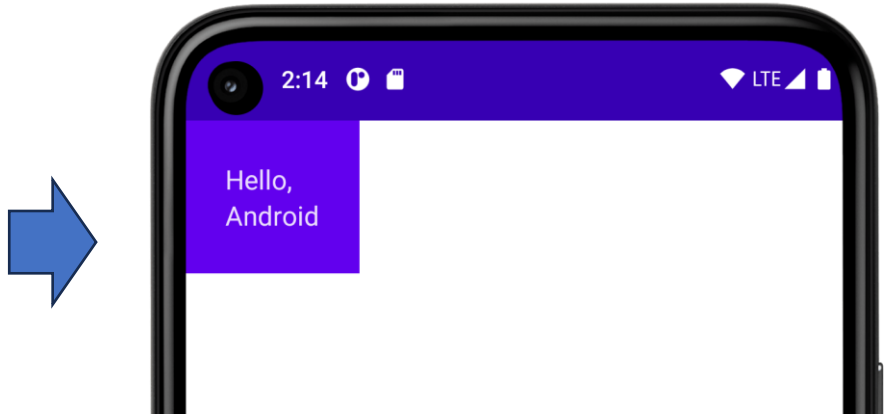


https://developer.android.com/develop/ui/compose/layouts/basics

Compose modifiers

- Modifiers allow you to decorate or augment a composable. Modifiers let you do these sorts of things:
  - Change the composable's size, layout, behavior, and appearance
  - Add information, like accessibility labels
  - Process user input
  - Add high-level interactions, like making an element clickable, scrollable, draggable, or zoomable
- Modifiers are standard Kotlin objects. Create a modifier by calling one of the Modifier class functions:

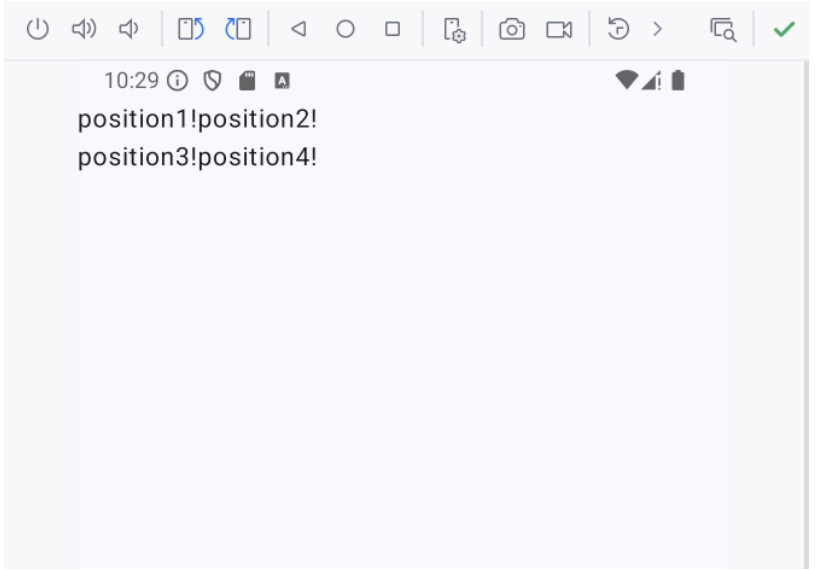
```
@Composable
private fun Greeting(name: String) {
    Column(modifier = Modifier.padding(24.dp)) {
        Text(text = "Hello,")
        Text(text = name)
    }
}
```



https://developer.android.com/develop/ui/compose/modifiers

A sample of Column and Row

```
Scaffold( modifier = Modifier.fillMaxSize() ) { innerPadding ->
    Column(modifier = Modifier.padding(innerPadding)) {
        Row{
            Greeting("position1")
            Greeting("position2")
        }
        Row{
            Greeting("position3")
            Greeting("position4")
        }
    }
}
```

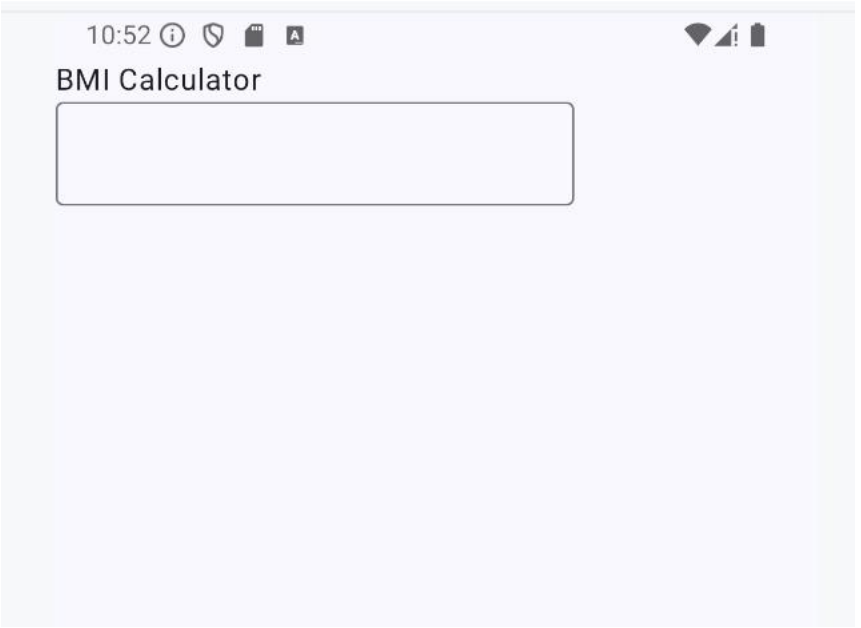


Text and TextField

```
Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
    BMICalc(
        modifier = Modifier.padding(innerPadding)
    )
}
```

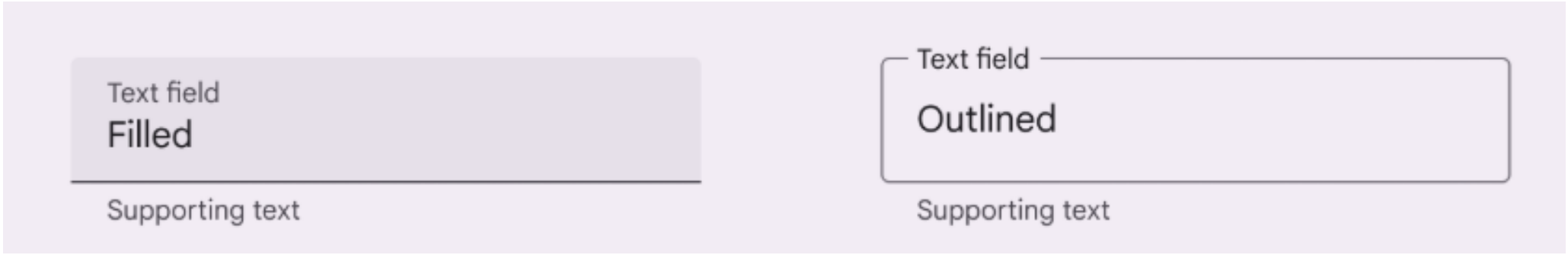
```
@Composable
fun BMICalc(modifier: Modifier = Modifier ){
    Column(modifier = modifier) {
        Text("BMI Calculator")
        OutlinedTextField(value = "", onValueChange = {})
    }

}
```



The type of TextField

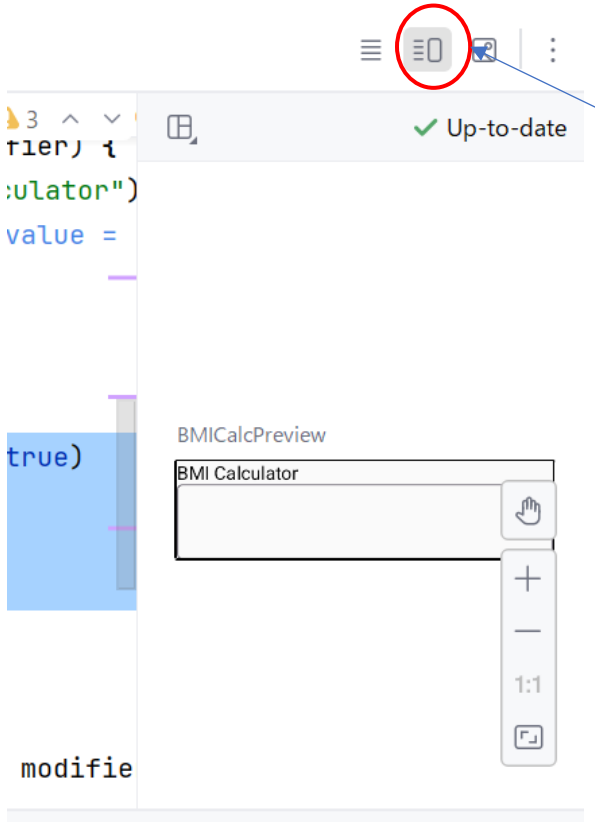
- Filled
  - Filled text fields have more visual emphasis than outlined text fields. They're often used in dialogs and short forms where their style draws more attention.
- Outlined
  - Outlined text fields have less visual emphasis than filled text fields. They're often used in long forms where their reduced emphasis helps simplify the layout.



Preview the design

- Android Studio offers some features to extend composable previews. You can change their container design, interact with them, or deploy them directly to an emulator or device.

```
@Preview(showBackground = true)
@Composable
fun BMICalcPreview(){
    BMICalc()
}
```

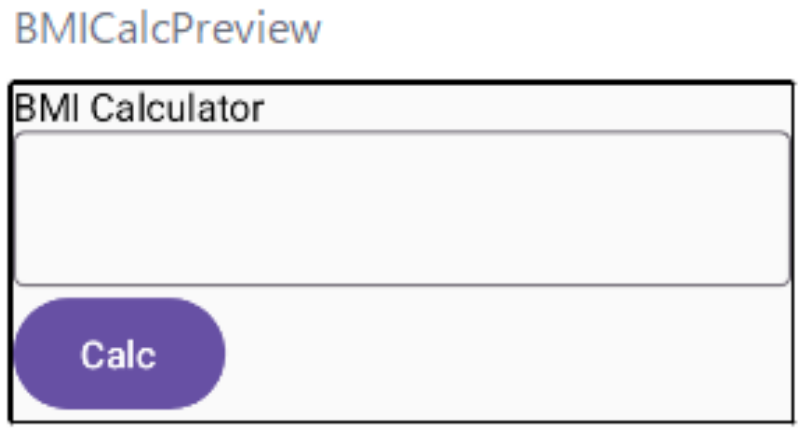


split the code window and preview window

Button

```
Column(modifier = modifier) {
    Text("BMI Calculator")
    OutlinedTextField(value = "", onValueChange = {})

    Row{
        Button(onClick = {}) {
            Text("Calc")
        }
    }
}
```



https://developer.android.com/develop/ui/compose/components/button

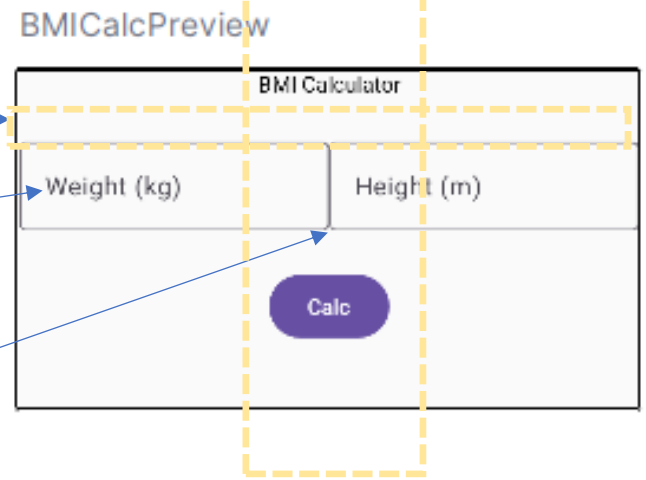
Design the App

```
fun BMICalc(modifier: Modifier = Modifier ){
    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text("BMI Calculator")
        Spacer(modifier = Modifier.height(20.dp))
        Row{
            OutlinedTextField(
                value = " ",
                label = { Text("Weight (kg)") },
                modifier = Modifier.weight(1f))
)
            OutlinedTextField(
                value = " ",
                label = { Text("Height (m)") },
                modifier = Modifier.weight(1f))
                )
        }
```

Center

Spacer

Label

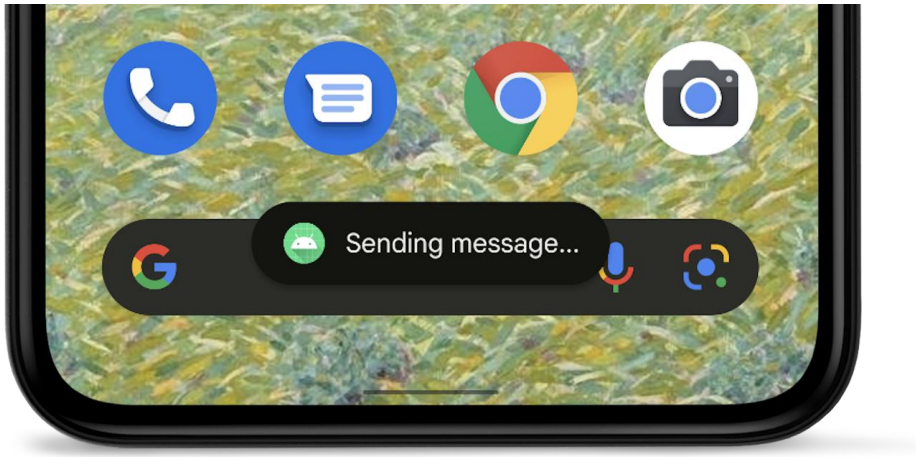


Each input field is given a modifier of Modifier.weight(1f), which ensures that they each take up an equal portion of the available horizontal space.

https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/Arrangement

Toasts

- A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.



https://developer.android.com/guide/topics/ui/notifiers/toasts

Add a toast for Button onClick event

```
Row{
    val context = LocalContext.current
    Button(onClick = {
        Toast.makeText(context,
            "Calcuated the BMI",
            Toast.LENGTH_LONG).show()
    }) {
        Text("Calc")
    }
}
```

LocalContext: Provides the current Android context needed for operations like displaying a toast.

Toast: A simple pop-up message used to provide feedback to the user.
Parameters:
context: The Android context obtained earlier via LocalContext.current.
"Calcuated the BMI": The message that will be displayed in the toast.
Toast.LENGTH_LONG: Specifies that the toast should be displayed for a long duration.
show(): Calling this method actually displays the toast on the screen.

State Management

- Uses **remember** and **mutableStateOf** to handle user inputs and the computed result, updating the UI reactively.

```
fun BMICalc(modifier: Modifier = Modifier ){
    var weightInput by remember { mutableStateOf("") }
    var heightInput by remember { mutableStateOf("") }
    var result by remember { mutableStateOf("") }
    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    )
```

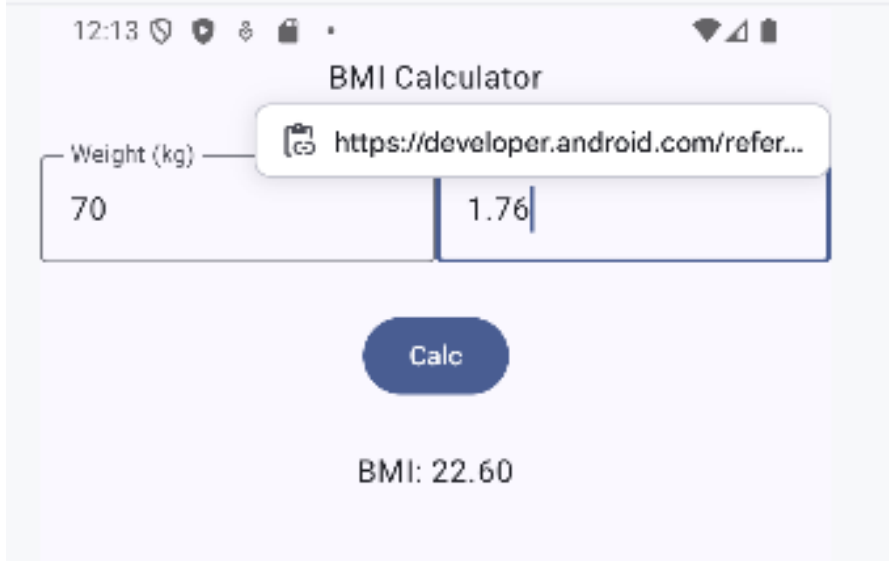
Update the TextField source code

```
Row{
    OutlinedTextField(
        value = weightInput,
        onValueChange = { weightInput = it },
        label = { Text("Weight (kg)") },
        modifier = Modifier.weight(1f))
    OutlinedTextField(
        value = heightInput,
        onValueChange = { heightInput = it },
        label = { Text("Height (m)") },
        modifier = Modifier.weight(1f))
}
```

Add the Button action to calculate BMI

```
Row{
    val context = LocalContext.current
    Button(onClick = {
        val weight = weightInput.toFloatOrNull()
        val height = heightInput.toFloatOrNull()
        if (weight == null || height == null || height <= 0f) {
            result = "Please input the values"
        } else {
            val bmi = weight / (height * height)
            result = "BMI: %.2f".format(bmi)
        }
    }) {
        Text("Calc")
    }
}
Spacer(modifier = Modifier.height(24.dp))
Text(
    text = result
)
```

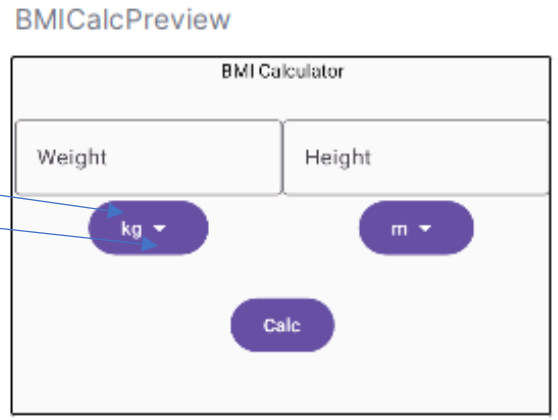
Show the result at screen



Use Box to design the Drop-Down Menu

```
Row {
    Box(
        modifier = Modifier.weight(1f),
        contentAlignment = Alignment.Center
    ){
        Button(onClick={}) {
            Text("kg")
            Icon(Icons.Default.ArrowDropDown, contentDescription = "Arrow Down")
        }
    }
    Box(
        modifier = Modifier.weight(1f),
        contentAlignment = Alignment.Center
    ){
        Button(onClick={}) {
            Text("m")
            Icon(Icons.Default.ArrowDropDown, contentDescription = "Arrow Down")
        }
    }
}
```

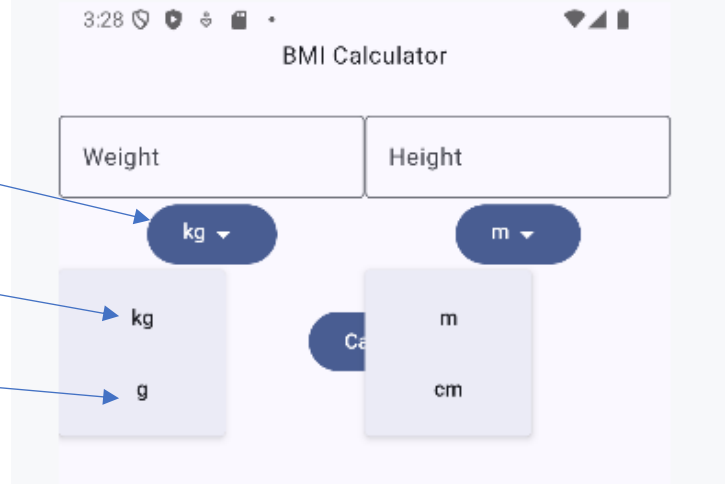
Make it come to center



https://developer.android.com/reference/kotlin/androidx/compose/material/icons/Icons.Filled

Add the DropMenu Item and expand action

```
DropdownMenu(
    expanded = true,
    onDismissRequest = {},
) {
    DropdownMenuItem(
        text = { Text("kg", textAlign = TextAlign.Center, modifier = Modifier.fillMaxWidth()) },
        onClick = {}
    )
    DropdownMenuItem(
        text = { Text("g", textAlign = TextAlign.Center, modifier = Modifier.fillMaxWidth()) },
        onClick = {}
    )
}
```



Make the Drop Menu can be select (1)

Setp 1. Prepare the variables

```
// State variables to control the dropdown menus and store button widths.
var weightDropdownExpanded by remember { mutableStateOf(false) }
var heightDropdownExpanded by remember { mutableStateOf(false) }

// Unit selection state variables
var selectedWeightUnit by remember { mutableStateOf("kg") }
var selectedHeightUnit by remember { mutableStateOf("m") }
```

Setp 2. Use the states to change the Click and text status.

```
Button(
    onClick = { weightDropdownExpanded = !weightDropdownExpanded }
) {
    Row {
        Text(selectedWeightUnit)
        Icon(Icons.Default.ArrowDropDown, contentDescription = "Arrow Down")
    }
}
```

Make the Drop Menu can be select (2)

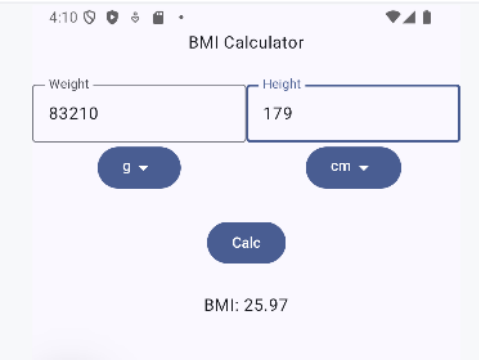
Setp3. Change the Unit Selection States

```
DropdownMenu(
    expanded = weightDropdownExpanded,
    onDismissRequest = { weightDropdownExpanded = false},
) {
    DropdownMenuItem(
        text = { Text("kg", textAlign = TextAlign.Center, modifier = Modifier.fillMaxWidth()) },
        onClick = {
            selectedWeightUnit = "kg"
            weightDropdownExpanded = false
        }
    )
    DropdownMenuItem(
        text = { Text("g", textAlign = TextAlign.Center, modifier = Modifier.fillMaxWidth()) },
        onClick = {
            selectedWeightUnit = "g"
            weightDropdownExpanded = false
```

the onClick callbacks now update the corresponding selected unit.

BMI Calculation Adjustments

```
Button(onClick = {
    val weightVal = weightInput.toFloatOrNull()
    val heightVal = heightInput.toFloatOrNull()
    if (weightVal == null || heightVal == null || heightVal <= 0f) {
        result = "Please input valid values"
    } else {
        // Convert weight to kilograms if necessary
        val weightInKg = if (selectedWeightUnit == "g") weightVal / 1000f else weightVal
        // Convert height to meters if necessary
        val heightInM = if (selectedHeightUnit == "cm") heightVal / 100f else heightVal
        if (heightInM <= 0f) {
            result = "Height must be greater than 0"
        } else {
            val bmi = weightInKg / (heightInM * heightInM)
            result = "BMI: %.2f".format(bmi)
        }
    }
}) {
    Text("Calc")
}
```



Week 4 Assignment

- About the assignment of this week. Please practice the teaching content covered this week and submit your work after making appropriate modifications or improvements to the code in the course materials.
- (For example, changing the font size, modifying the type of text field, altering the shape of the button, or applying the code to solve other computational problems.)