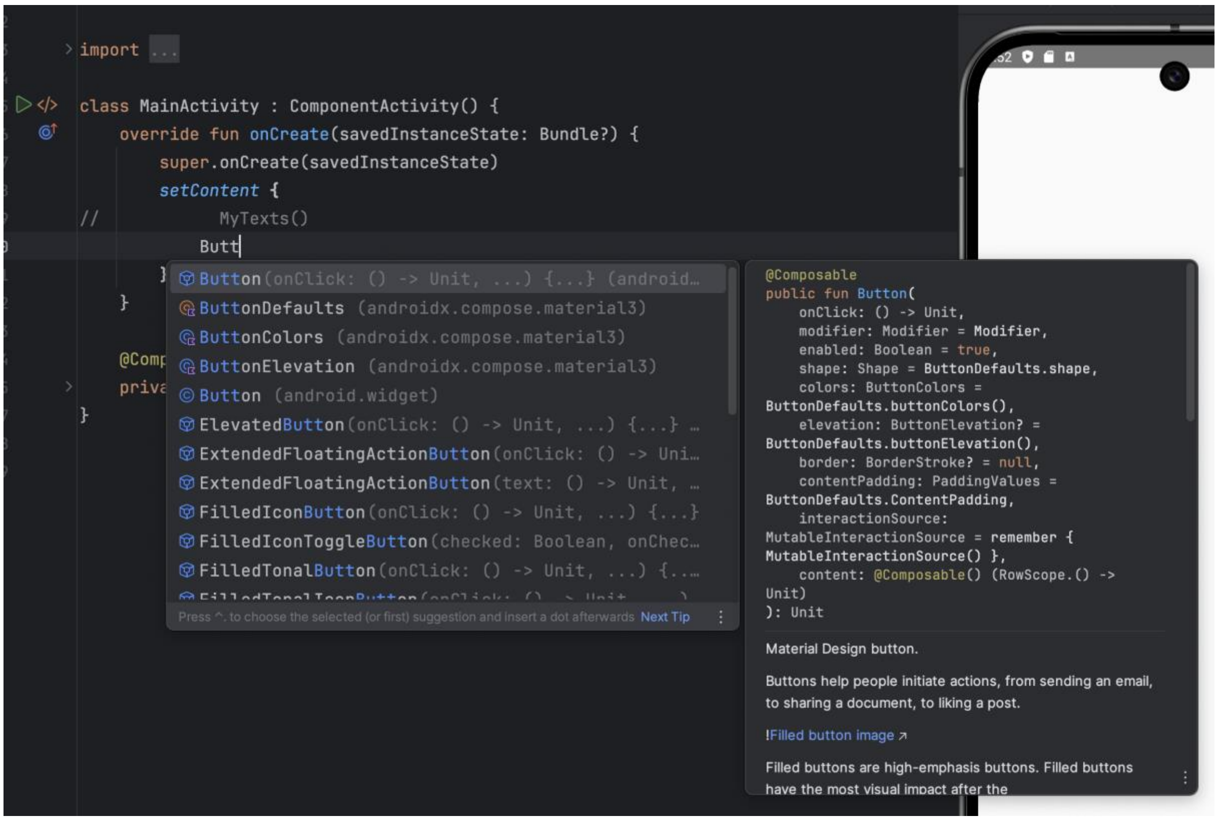
Android App Development – Laboratory 1B

- The next UI element we are going to use, which will allow us to perform "actions," is the Button. So, let's go to our setContent function, let's start typing Button, and Studio will show us the available options.

- 

```
> import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//          MyTexts()
            Butt
        }
    }

    @Comp
    priva
}
```

```
Button(onClick: () -> Unit, ...) {...} (android…
ButtonDefaults (androidx.compose.material3)
ButtonColors (androidx.compose.material3)
ButtonElevation (androidx.compose.material3)
Button (android.widget)
ElevatedButton(onClick: () -> Unit, ...) {...} …
ExtendedFloatingActionButton(onClick: () -> Uni…
ExtendedFloatingActionButton(text: () -> Unit, …
FilledIconButton(onClick: () -> Unit, ...) {...}
FilledIconToggleButton(checked: Boolean, onChec…
FilledTonalButton(onClick: () -> Unit, ...) {...…
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards Next Tip
```

```
@Composable
public fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors =
ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? =
ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues =
ButtonDefaults.ContentPadding,
    interactionSource:
MutableInteractionSource = remember {
MutableInteractionSource() },
    content: @Composable() (RowScope.() ->
Unit)
): Unit
```

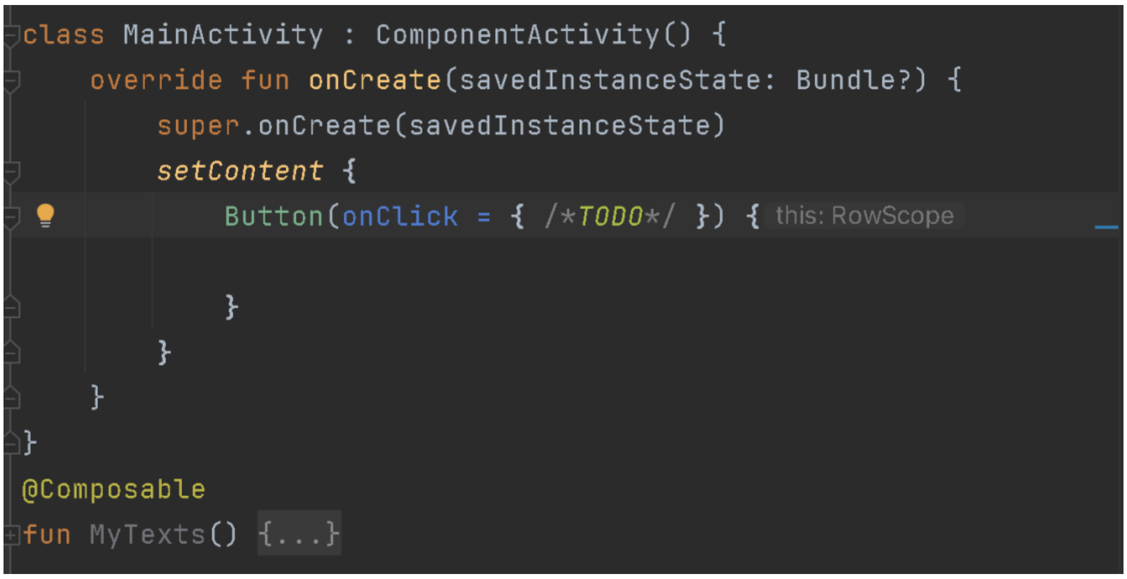
Material Design button.

Buttons help people initiate actions, from sending an email, to sharing a document, to liking a post.

!Filled button image ↗

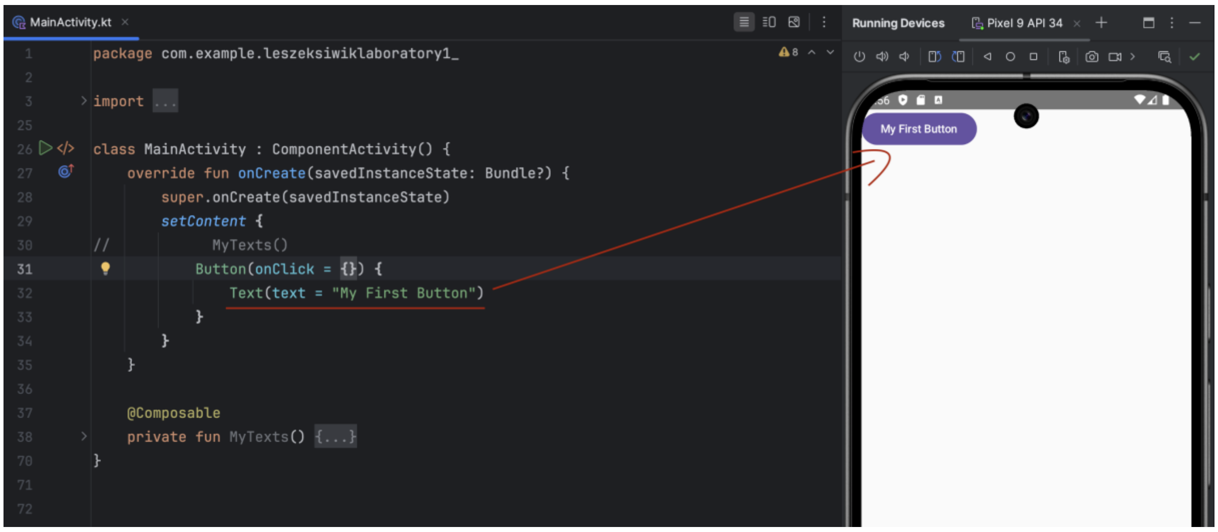
Filled buttons are high-emphasis buttons. Filled buttons have the most visual impact after the

- We're interested in the first option, so we press Enter and start working with our button.

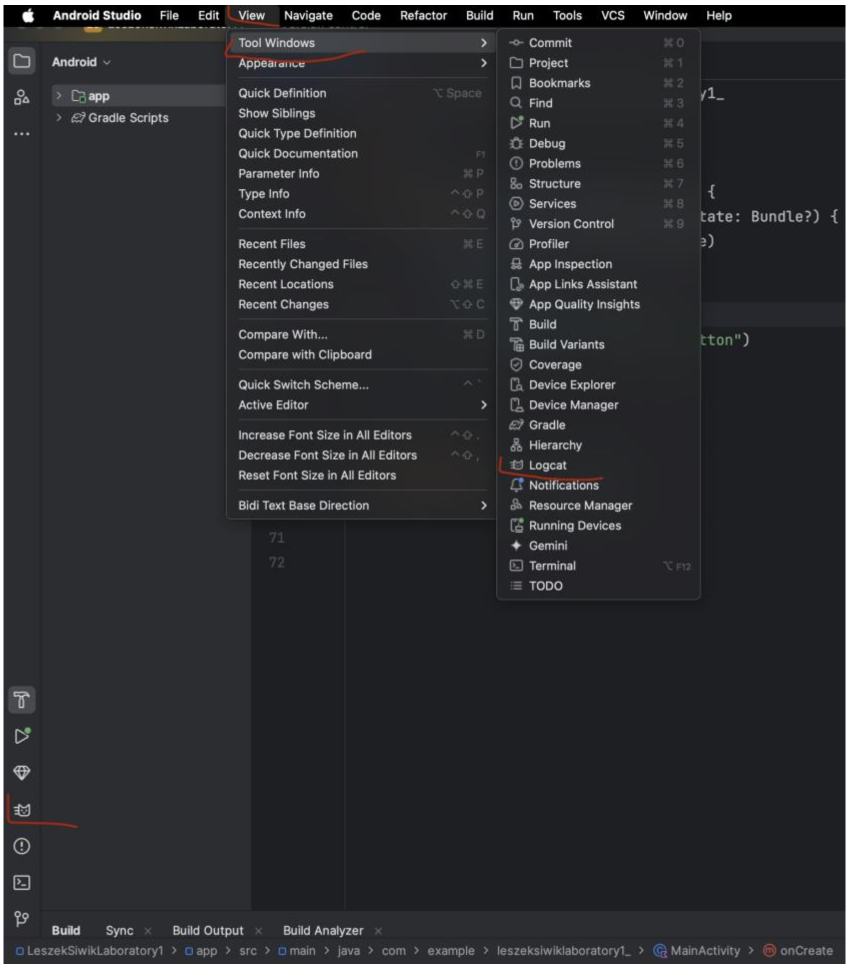
- 

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Button(onClick = { /*TODO*/ }) { this: RowScope

            }
        }
    }
}
@Composable
fun MyTexts() {...}
```

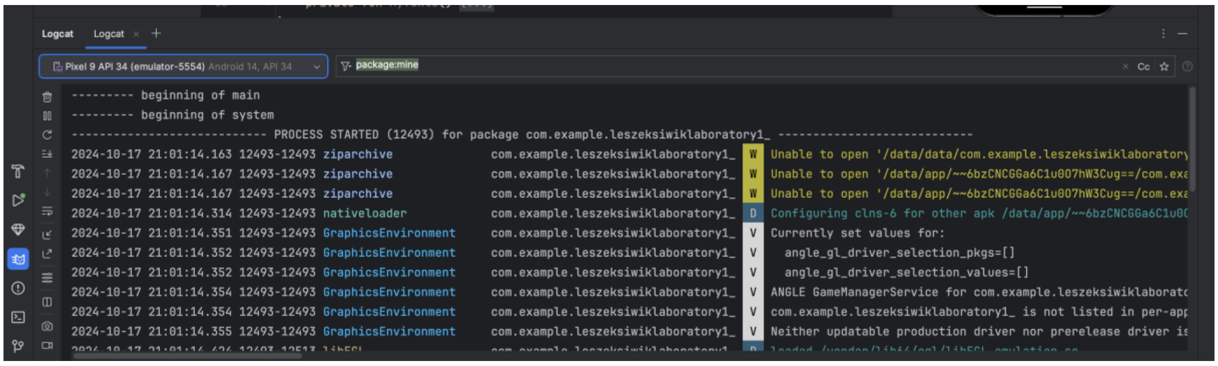
- Let's start by adding some text on the button (a familiar Text element) so we know what this button is for. So, let's begin by adding the text "My first button" and see what we get. We should expect something like this

- 

```
package com.example.leszeksiwiklaboratory1_

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//              MyTexts()
            Button(onClick = {}) {
                Text(text = "My First Button")
            }
        }
    }

    @Composable
    private fun MyTexts() {...}
}
```

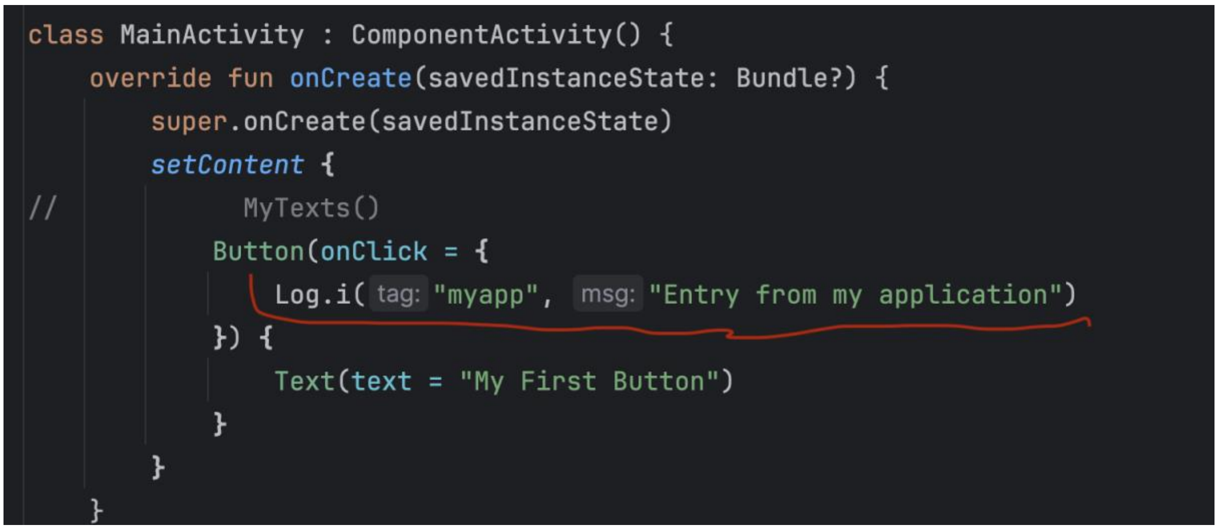
- The button is, of course, clickable, but currently, it doesn't do anything (or rather, the application doesn't do anything when it's clicked because as you can see, the content of the onClick attribute is empty and contains only a comment; this is still to be implemented). So let's try to perform some action.
- Let's start by trying to save some message/content in the logs of our device/emulator. The first important thing is how to access the logs of the device and be able to view them. Well, in Android Studio, we have the LogCat tool precisely for this purpose. It is available through one of the tabs in the bottom part of Android Studio:
- 

- Let's open it, and after a moment needed to establish a connection with the device/emulator and initialize the whole mechanism/tool, we should see logs from our emulator.

- 

```
--------- beginning of main
--------- beginning of system
---------------------------- PROCESS STARTED (12493) for package com.example.leszeksiwiklaboratory1_ ----------------------------
2024-10-17 21:01:14.163 12493-12493 ziparchive              com.example.leszeksiwiklaboratory1_  W  Unable to open '/data/data/com.example.leszeksiwiklaboratory
2024-10-17 21:01:14.167 12493-12493 ziparchive              com.example.leszeksiwiklaboratory1_  W  Unable to open '/data/app/~~6bzCNCGGa6C1u0O7hW3Cug==/com.exa
2024-10-17 21:01:14.167 12493-12493 ziparchive              com.example.leszeksiwiklaboratory1_  W  Unable to open '/data/app/~~6bzCNCGGa6C1u0O7hW3Cug==/com.exa
2024-10-17 21:01:14.314 12493-12493 nativeloader            com.example.leszeksiwiklaboratory1_  D  Configuring clns-6 for other apk /data/app/~~6bzCNCGGa6C1u0
2024-10-17 21:01:14.351 12493-12493 GraphicsEnvironment     com.example.leszeksiwiklaboratory1_  V  Currently set values for:
2024-10-17 21:01:14.352 12493-12493 GraphicsEnvironment     com.example.leszeksiwiklaboratory1_  V    angle_gl_driver_selection_pkgs=[]
2024-10-17 21:01:14.352 12493-12493 GraphicsEnvironment     com.example.leszeksiwiklaboratory1_  V    angle_gl_driver_selection_values=[]
2024-10-17 21:01:14.354 12493-12493 GraphicsEnvironment     com.example.leszeksiwiklaboratory1_  V  ANGLE GameManagerService for com.example.leszeksiwiklaborato
2024-10-17 21:01:14.354 12493-12493 GraphicsEnvironment     com.example.leszeksiwiklaboratory1_  V  com.example.leszeksiwiklaboratory1_ is not listed in per-app
2024-10-17 21:01:14.355 12493-12493 GraphicsEnvironment     com.example.leszeksiwiklaboratory1_  V  Neither updatable production driver nor prerelease driver is
```

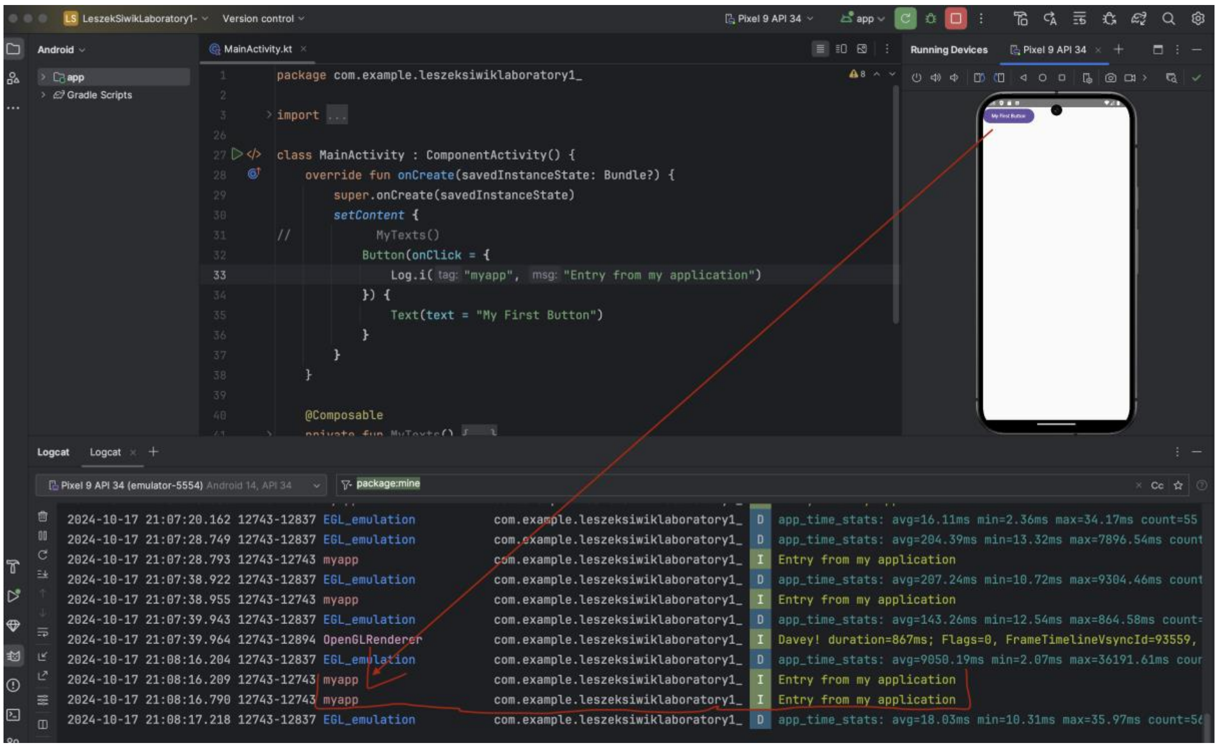
- Now when we know where to find any entries in the logs, let's try to write something there. So, we go back to the definition of our button, go to the definition of the onClick parameter, and let's add there the following code:

- 

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//            MyTexts()
            Button(onClick = {
                Log.i( tag: "myapp",  msg: "Entry from my application")
            }) {
                Text(text = "My First Button")
            }
        }
    }
```

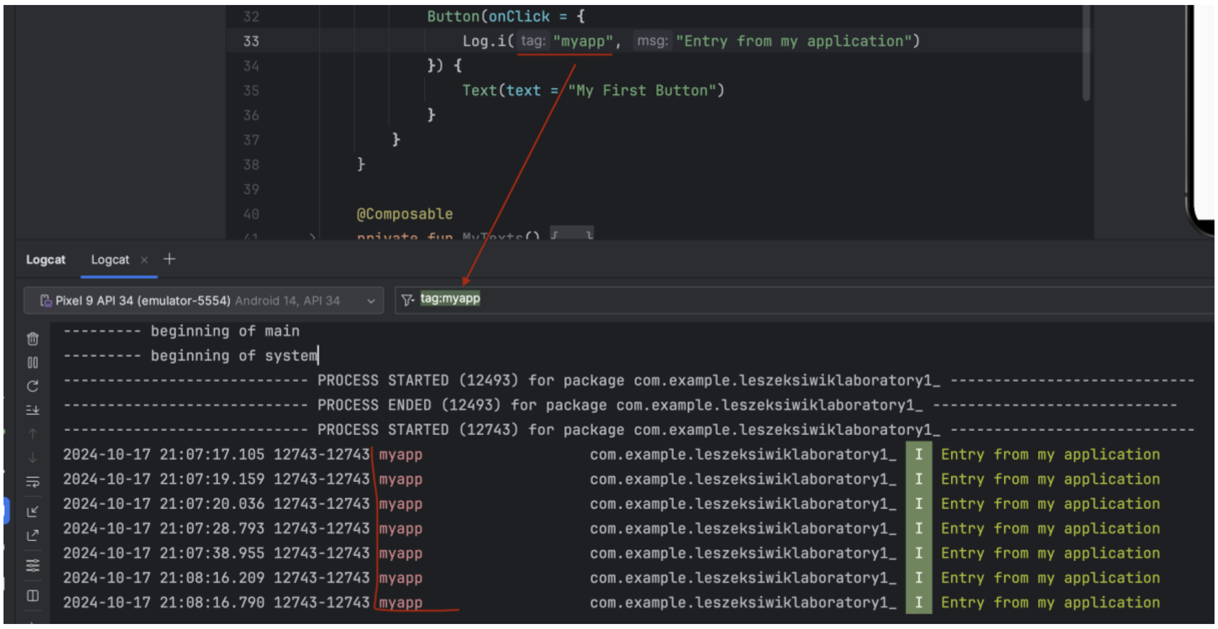
- This piece of code should add an entry "Entry from my application" in the logs tagged with "myapp" (which makes it easier to search for the logs we're interested in using LogCat).

- So let's rebuild our application and see how it works:

- 

```kotlin
package com.example.leszeksiwiklaboratory1_

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//              MyTexts()
            Button(onClick = {
                Log.i( tag: "myapp",  msg: "Entry from my application")
            }) {
                Text(text = "My First Button")
            }
        }
    }

    @Composable
```

```
2024-10-17 21:07:20.162 12743-12837 EGL_emulation           com.example.leszeksiwiklaboratory1_  D  app_time_stats: avg=16.11ms min=2.36ms max=34.17ms count=55
2024-10-17 21:07:28.749 12743-12837 EGL_emulation           com.example.leszeksiwiklaboratory1_  D  app_time_stats: avg=204.39ms min=13.32ms max=7896.54ms count
2024-10-17 21:07:28.793 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:07:38.922 12743-12837 EGL_emulation           com.example.leszeksiwiklaboratory1_  D  app_time_stats: avg=207.24ms min=10.72ms max=9304.46ms count
2024-10-17 21:07:38.955 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:07:39.943 12743-12837 EGL_emulation           com.example.leszeksiwiklaboratory1_  D  app_time_stats: avg=143.26ms min=12.54ms max=864.58ms count=
2024-10-17 21:07:39.964 12743-12894 OpenGLRenderer          com.example.leszeksiwiklaboratory1_  I  Davey! duration=867ms; Flags=0, FrameTimelineVsyncId=93559,
2024-10-17 21:08:16.204 12743-12837 EGL_emulation           com.example.leszeksiwiklaboratory1_  D  app_time_stats: avg=9050.19ms min=2.07ms max=36191.61ms cour
2024-10-17 21:08:16.209 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:08:16.790 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:08:17.218 12743-12837 EGL_emulation           com.example.leszeksiwiklaboratory1_  D  app_time_stats: avg=18.03ms min=10.31ms max=35.97ms count=5
```

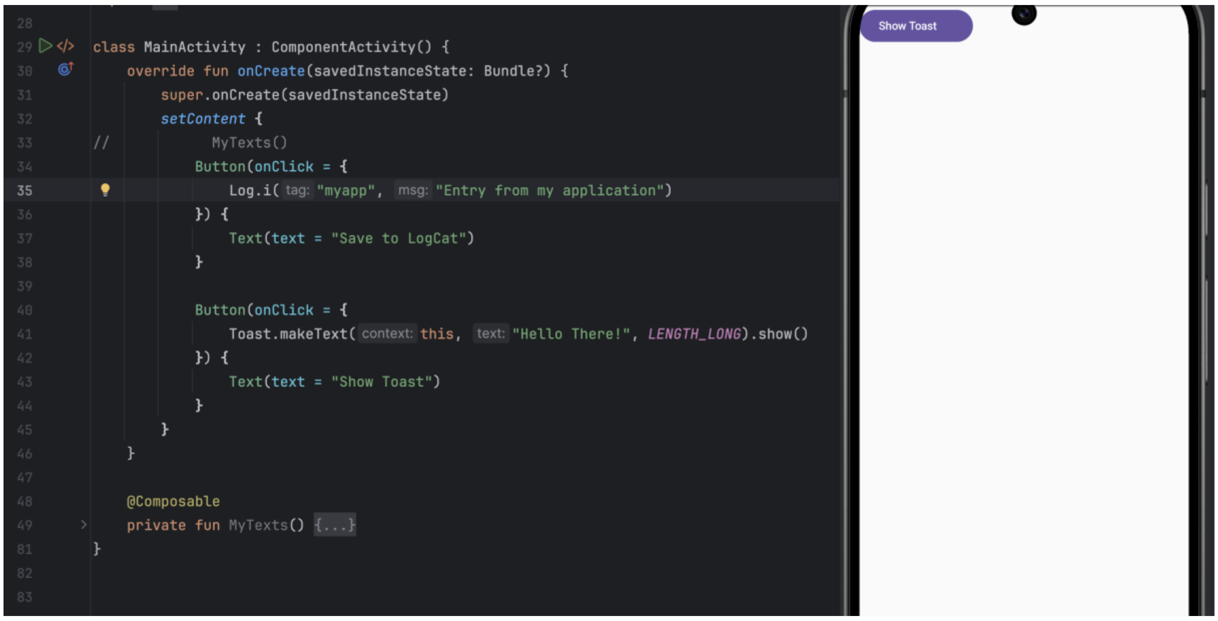
- And it seems that "something" is working because after each click on the button, a new entry appears in the logs.
- For our convenience, we can filter the LogCat output by our tag:
- 

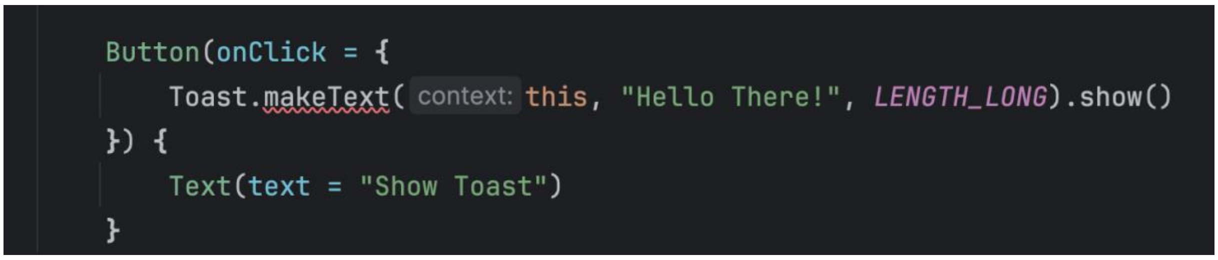
```kotlin
            Button(onClick = {
                Log.i( tag: "myapp",  msg: "Entry from my application")
            }) {
                Text(text = "My First Button")
            }
        }
    }

    @Composable
```

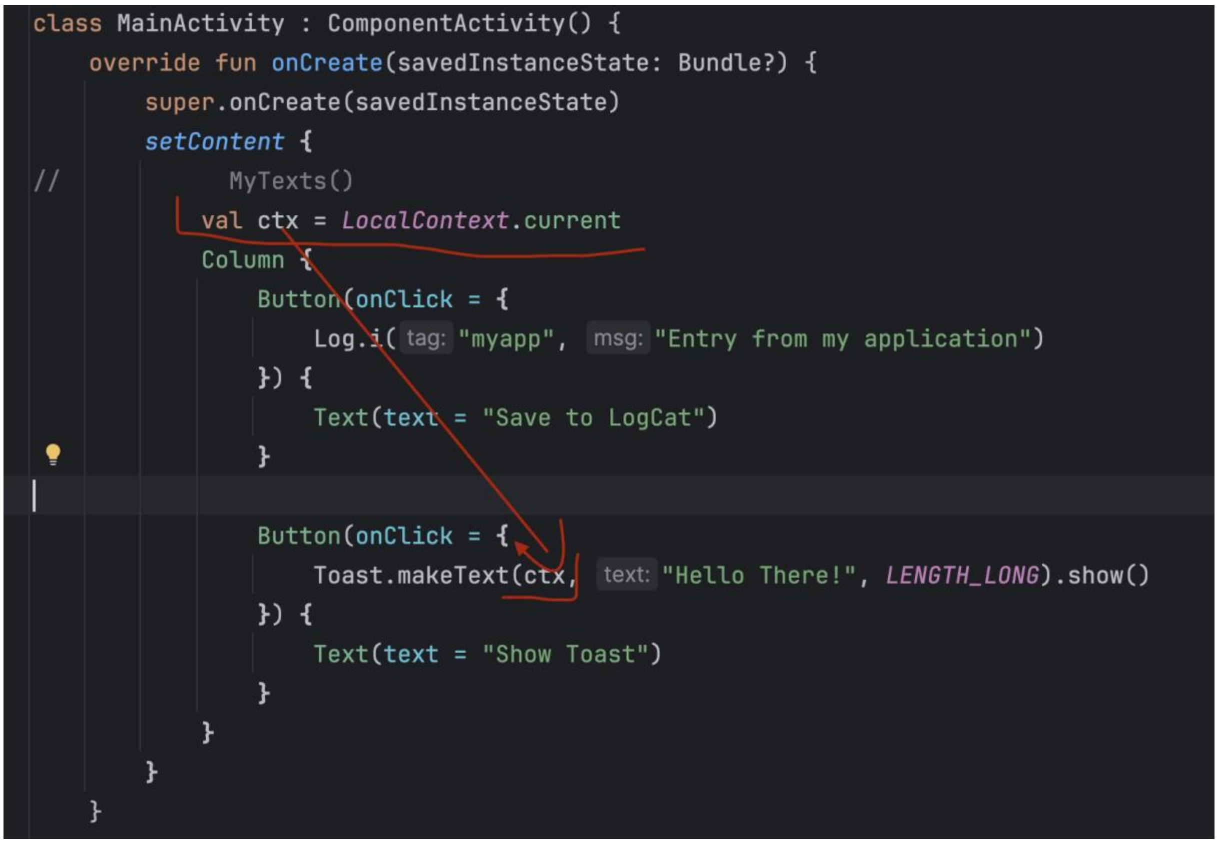
```
--------- beginning of main
--------- beginning of system
--------------------------- PROCESS STARTED (12493) for package com.example.leszeksiwiklaboratory1_ ---------------------------
--------------------------- PROCESS ENDED (12493) for package com.example.leszeksiwiklaboratory1_ ---------------------------
--------------------------- PROCESS STARTED (12743) for package com.example.leszeksiwiklaboratory1_ ---------------------------
2024-10-17 21:07:17.105 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:07:19.159 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:07:20.036 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:07:28.793 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:07:38.955 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:08:16.209 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
2024-10-17 21:08:16.790 12743-12743 myapp                   com.example.leszeksiwiklaboratory1_  I  Entry from my application
```

- Ok, so we know how to log messages. Let's try displaying something in the application itself now. But before we proceed, let's change the text on our first button to "Save to LogCat" so it's clear what it does and which one is which. Now, let's add a second button, set the text on ot to "Show Toast", and as for onClick attribute, let's add the following piece of code:

```kotlin
Toast.makeText( context: this,  text: "Hello There!", LENGTH_LONG).show()
```

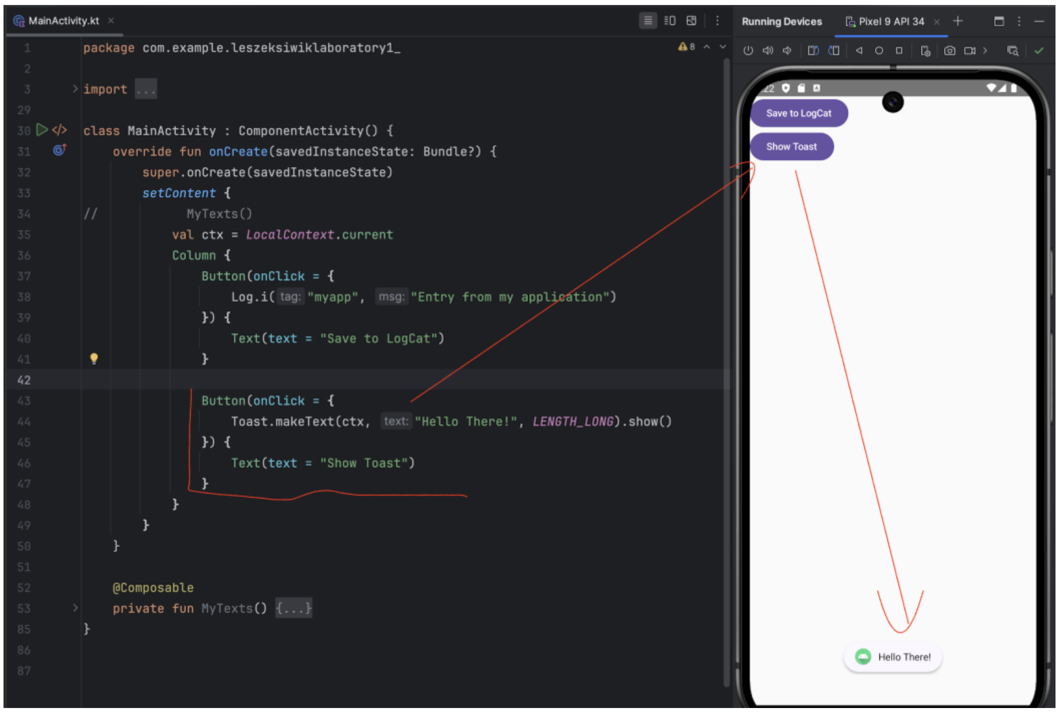
- And let's check out what we get
- 
```kotlin
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//              MyTexts()
            Button(onClick = {
                Log.i( tag: "myapp",  msg: "Entry from my application")
            }) {
                Text(text = "Save to LogCat")
            }

            Button(onClick = {
                Toast.makeText( context: this,  text: "Hello There!", LENGTH_LONG).show()
            }) {
                Text(text = "Show Toast")
            }
        }
    }

    @Composable
    private fun MyTexts() {...}
}
```

- Well, it is not really what we expected since looks like the buttons have overlapped. But we already know how to deal with this. So let's wrap them in a column and check. Hmm, after moving the buttons inside the column, Android Studio started underlining the makeText function for me.
- 
```kotlin
Button(onClick = {
    Toast.makeText( context: this, "Hello There!", LENGTH_LONG).show()
}) {
    Text(text = "Show Toast")
}
```
- That's because this function requires the first argument to be the activity in the context of which the toast will be displayed. And while before wrapping the buttons in a column, we could provide this argument as "this," after putting them in a column, "this" no longer refers to the activity context but to the context of the column. And a Toast cannot be displayed in the context of a column. So let's remember to store the activity context somewhere before starting the column, like this:



```kotlin
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//              MyTexts()
            val ctx = LocalContext.current
            Column {
                Button(onClick = {
                    Log.i( tag: "myapp",  msg: "Entry from my application")
                }) {
                    Text(text = "Save to LogCat")
                }

                Button(onClick = {
                    Toast.makeText(ctx,  text: "Hello There!", LENGTH_LONG).show()
                }) {
                    Text(text = "Show Toast")
                }
            }
        }
    }
```

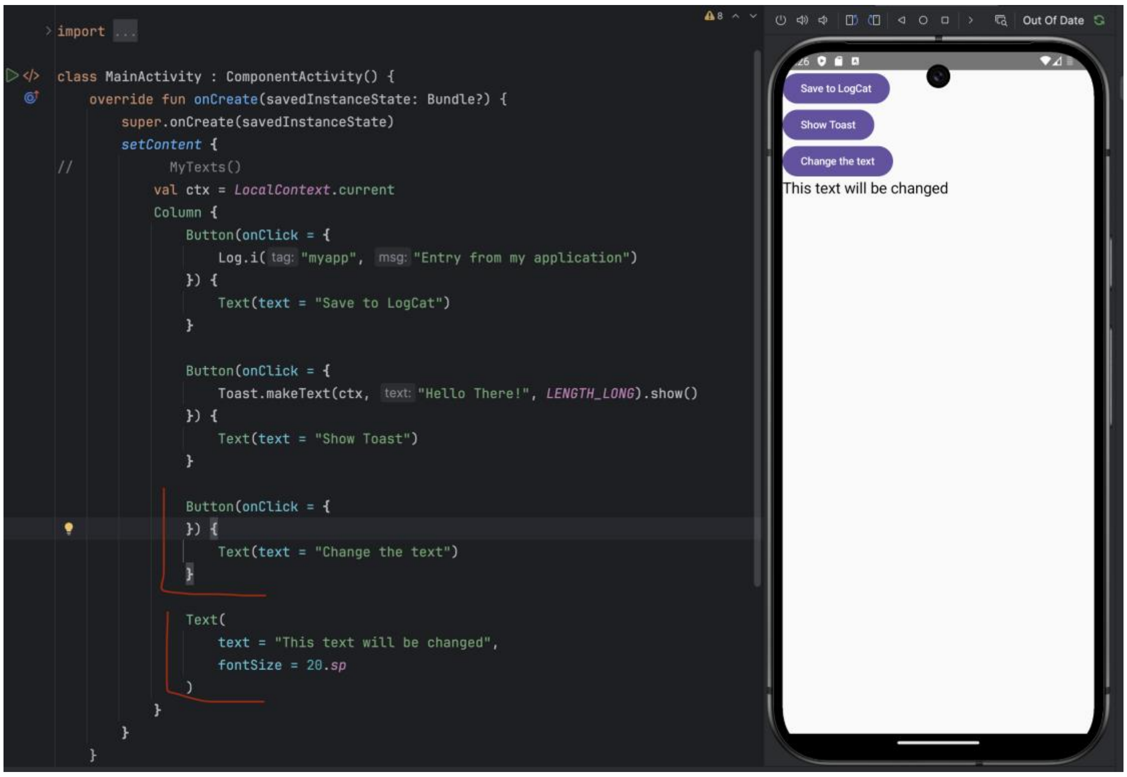
- And now, let’s use it in our makeText function. It should fix the problem, so let’x check what we get:

- 

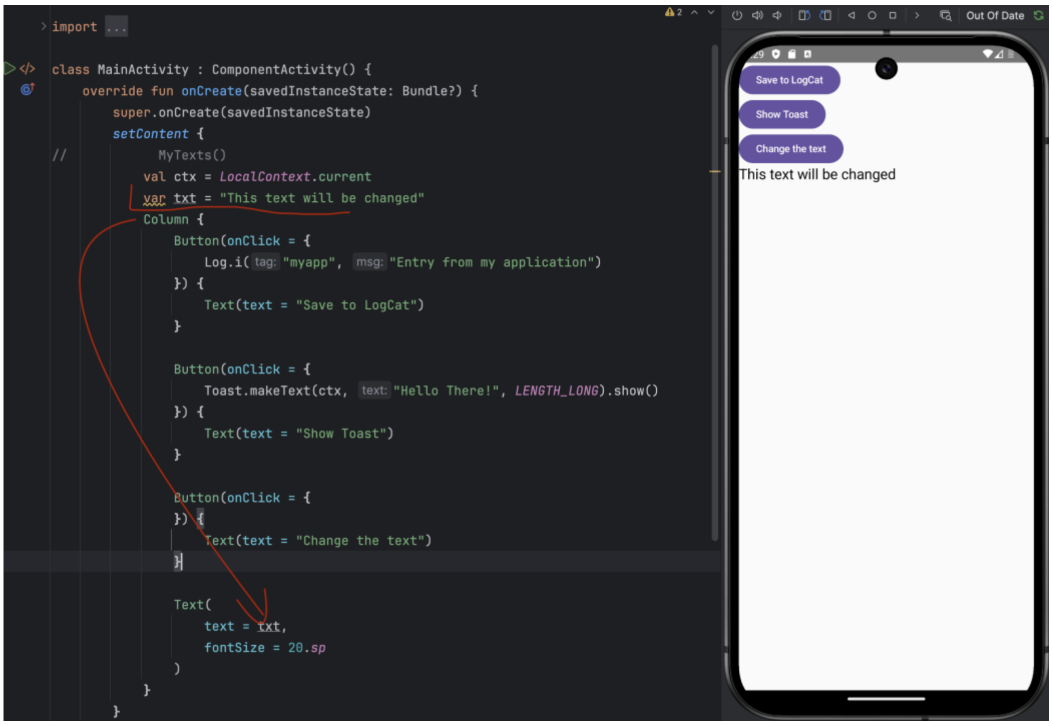
MainActivity.kt

```kotlin
package com.example.leszeksiwiklaboratory1_

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//              MyTexts()
            val ctx = LocalContext.current
            Column {
                Button(onClick = {
                    Log.i( tag: "myapp",  msg: "Entry from my application")
                }) {
                    Text(text = "Save to LogCat")
                }

                Button(onClick = {
                    Toast.makeText(ctx,  text: "Hello There!", LENGTH_LONG).show()
                }) {
                    Text(text = "Show Toast")
                }
            }
        }
    }

    @Composable
    private fun MyTexts() {...}
}
```

- Great., So let's continue but this time with trying to "write" something directly on the application UI element(s).
- So let's add another button with the text "Change the text" and a Text element with the text "This text will be changed".

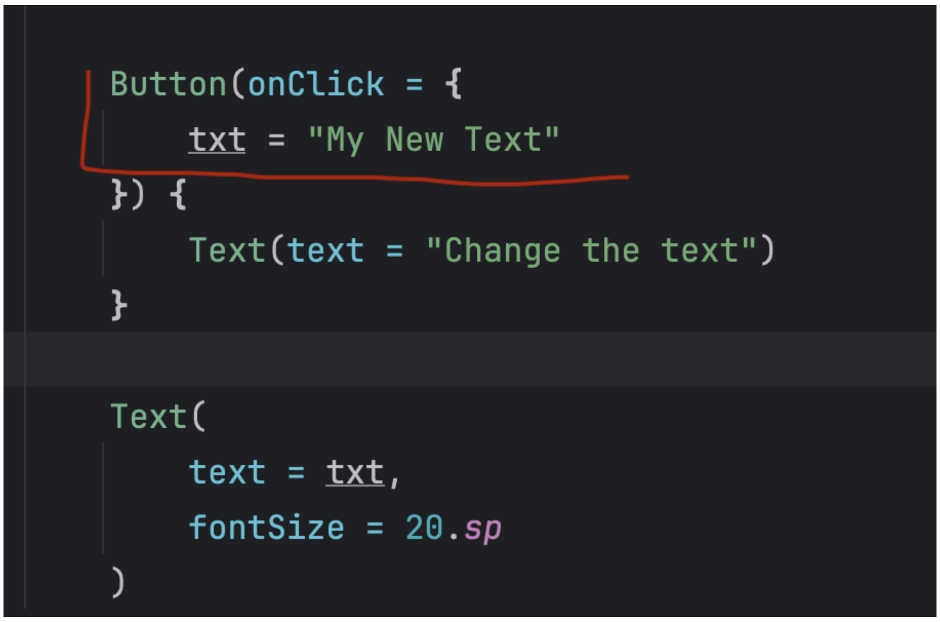
- 

```kotlin
import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//            MyTexts()
            val ctx = LocalContext.current
            Column {
                Button(onClick = {
                    Log.i( tag: "myapp",  msg: "Entry from my application")
                }) {
                    Text(text = "Save to LogCat")
                }

                Button(onClick = {
                    Toast.makeText(ctx,  text: "Hello There!", LENGTH_LONG).show()
                }) {
                    Text(text = "Show Toast")
                }

                Button(onClick = {
                }) {
                    Text(text = "Change the text")
                }

                Text(
                    text = "This text will be changed",
                    fontSize = 20.sp
                )
            }
        }
    }
```

- And as you can guess, after clicking our new button, we would like to change the displayed text/inscription.
- The first step would be to define a variable to store the content of our text:
- 

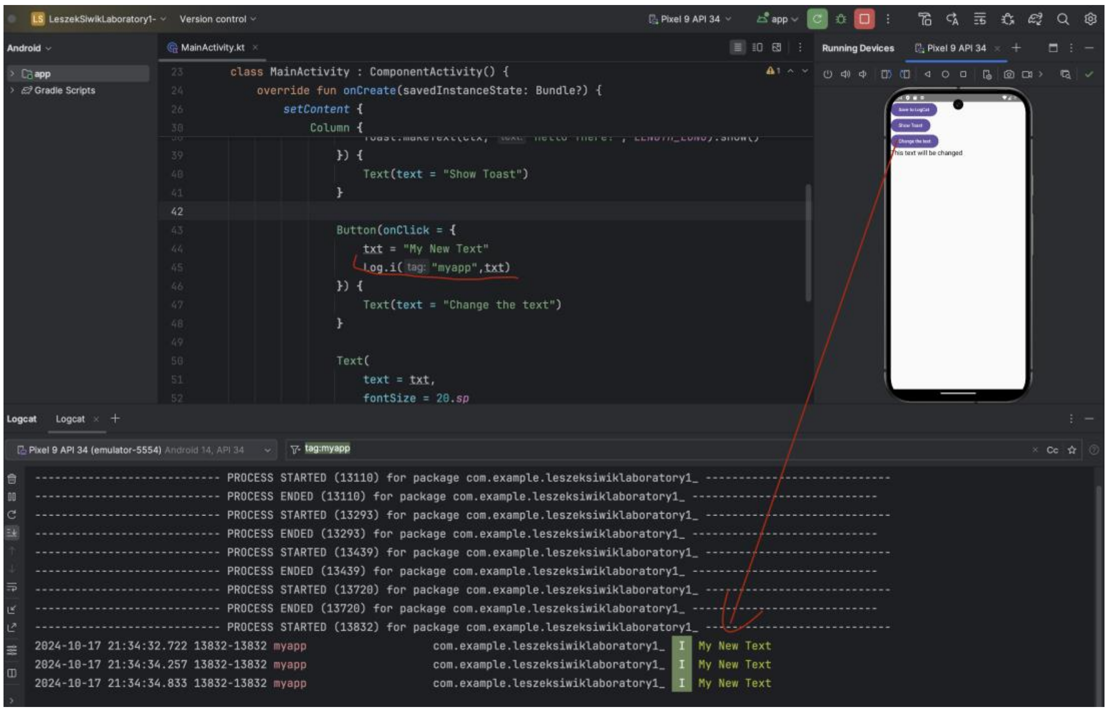
```kotlin
import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
//            MyTexts()
            val ctx = LocalContext.current
            var txt = "This text will be changed"
            Column {
                Button(onClick = {
                    Log.i( tag: "myapp",  msg: "Entry from my application")
                }) {
                    Text(text = "Save to LogCat")
                }

                Button(onClick = {
                    Toast.makeText(ctx,  text: "Hello There!", LENGTH_LONG).show()
                }) {
                    Text(text = "Show Toast")
                }

                Button(onClick = {
                }) {
                    Text(text = "Change the text")
                }

                Text(
                    text = txt,
                    fontSize = 20.sp
                )
            }
        }
```

- And, theoretically, nothing could be easier now; just as to change the value of our variable "txt" when the button is clicked.

- 

```
Button(onClick = {
    txt = "My New Text"
}) {
    Text(text = "Change the text")
}

Text(
    text = txt,
    fontSize = 20.sp
)
```

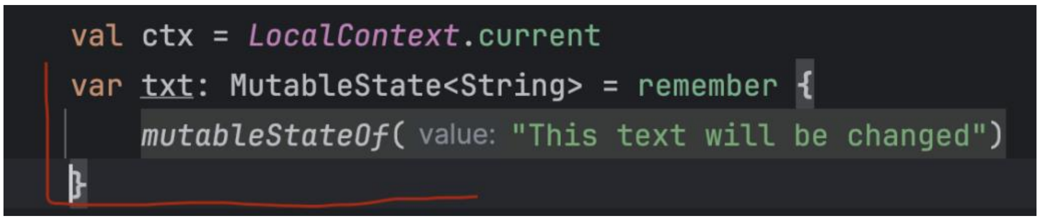
- And everything looks good... but doesn't work. I mean, no matter how many times we click our new button, the text content remains the same. So, the question is, "What went wrong?" Let's start by adding to our onClick (after updating the value of the txt variable) logging it to the device logs, and let's see if this action is actually being executed and if the content of this variable is being updated, and see what's going on there:

- 

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        setContent {
            Column {
                }) {
                    Text(text = "Show Toast")
                }

                Button(onClick = {
                    txt = "My New Text"
                    Log.i( tag: "myapp",txt)
                }) {
                    Text(text = "Change the text")
                }

                Text(
                    text = txt,
                    fontSize = 20.sp
```

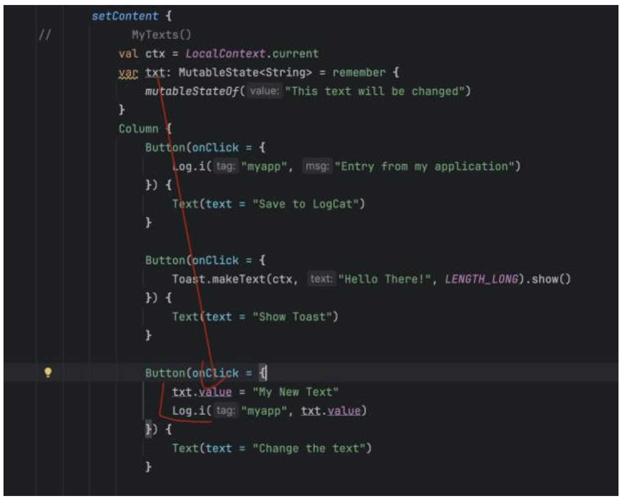
```
Pixel 9 API 34 (emulator-5554) Android 14, API 34      tag:myapp

---------------------------- PROCESS STARTED (13110) for package com.example.leszeksiwiklaboratory1_ ----------------------------
---------------------------- PROCESS ENDED (13110) for package com.example.leszeksiwiklaboratory1_ ----------------------------
---------------------------- PROCESS STARTED (13293) for package com.example.leszeksiwiklaboratory1_ ----------------------------
---------------------------- PROCESS ENDED (13293) for package com.example.leszeksiwiklaboratory1_ ----------------------------
---------------------------- PROCESS STARTED (13439) for package com.example.leszeksiwiklaboratory1_ ----------------------------
---------------------------- PROCESS ENDED (13439) for package com.example.leszeksiwiklaboratory1_ ----------------------------
---------------------------- PROCESS STARTED (13720) for package com.example.leszeksiwiklaboratory1_ ----------------------------
---------------------------- PROCESS ENDED (13720) for package com.example.leszeksiwiklaboratory1_ ----------------------------
---------------------------- PROCESS STARTED (13832) for package com.example.leszeksiwiklaboratory1_ ----------------------------
2024-10-17 21:34:32.722 13832-13832 myapp                com.example.leszeksiwiklaboratory1_  I  My New Text
2024-10-17 21:34:34.257 13832-13832 myapp                com.example.leszeksiwiklaboratory1_  I  My New Text
2024-10-17 21:34:34.833 13832-13832 myapp                com.example.leszeksiwiklaboratory1_  I  My New Text
```

- So, looks like the action is being executed, and the content of the variable is being updated, but in the application itself, the text remains unchanged. This is because the content of the txt variable is indeed updated, but our Text element on the screen was drawn with the "old" content of this variable, and later, no one (and nothing) forces it to be refreshed. To force this, the variable holding a specific attribute of the UI element (in this case, the text attribute of the Text element) must be declared in a "special" way, as a variable holding (mutable) state of our application/interface, which means as a MutableState<>. In our case, it will be:

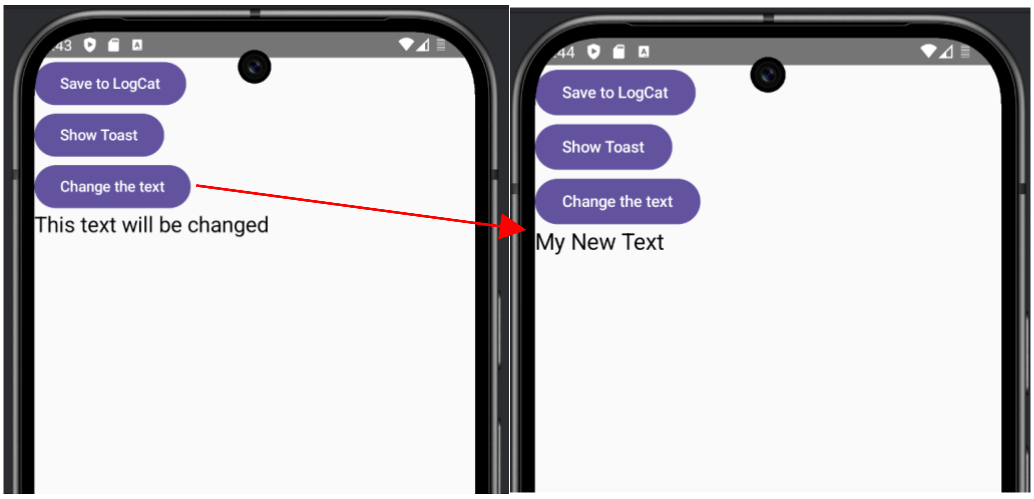
- 

```
val ctx = LocalContext.current
var txt: MutableState<String> = remember {
    mutableStateOf( value: "This text will be changed")
}
```

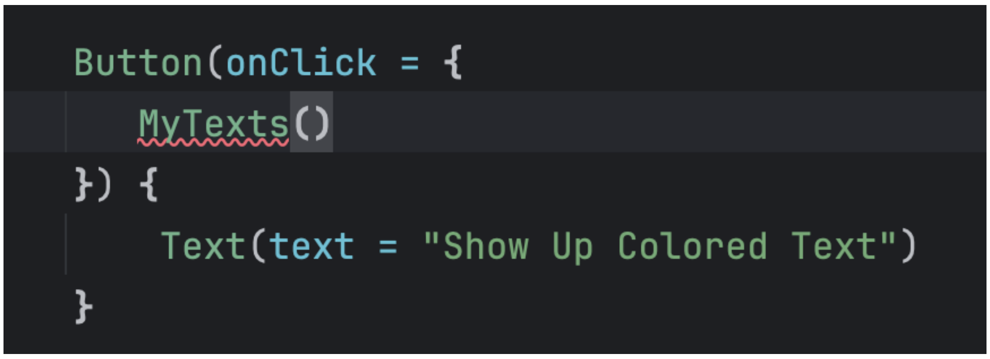
- And now, when modifying / accessing our txt we need to use its property .value. as below:
-



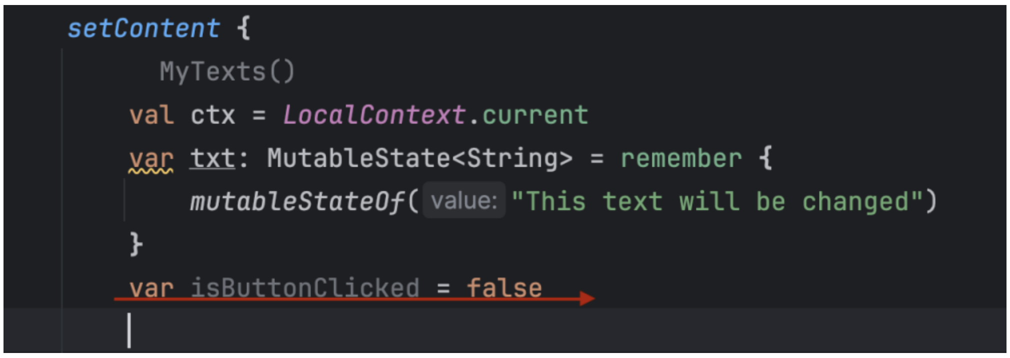
```
setContent {
//      MyTexts()
    val ctx = LocalContext.current
    var txt: MutableState<String> = remember {
        mutableStateOf( value: "This text will be changed")
    }
    Column {
        Button(onClick = {
            Log.i( tag: "myapp",  msg: "Entry from my application")
        }) {
            Text(text = "Save to LogCat")
        }

        Button(onClick = {
            Toast.makeText(ctx,  text: "Hello There!", LENGTH_LONG).show()
        }) {
            Text(text = "Show Toast")
        }

        Button(onClick = {
            txt.value = "My New Text"
            Log.i( tag: "myapp", txt.value)
        }) {
            Text(text = "Change the text")
        }
```

- And now it should work as expected:
- 

- The example is worth remembering because we need to proceed in the same way every time we want to define an attribute of a UI element through a variable and be able to modify it, forcing the refresh of the state of that element/attribute. So let's try to move on. Let's add a new button with the text "Display colored texts," and when this button is clicked, we would like to display several colored texts on the screen as described.

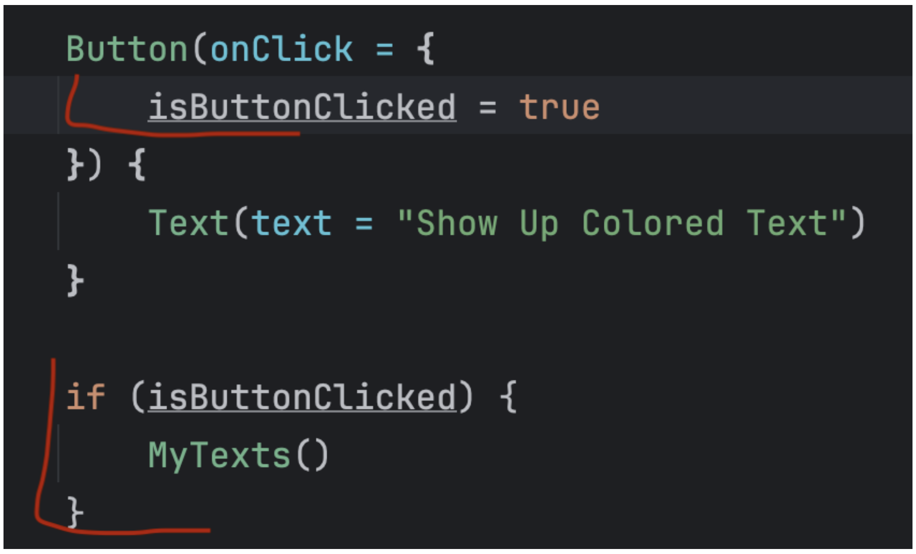
- And now, since the function MyTexts we implemented at the beginning of the session does exactly what we want (displays colored texts), we will use it. The first idea would be simply to add a call of MyTexts function inside onClick handler of our new button, like this:
- 

```
Button(onClick = {
    MyTexts()
}) {
    Text(text = "Show Up Colored Text")
}
```

- Unfortunately, as you can see, it's not possible because Composable functions (those adding/using UI elements, like our MyTexts function) can only be called from the other Composable functions (or from the setContent function), and onClick doesn't meet this condition.
- So, let's call our MyTexts directly from the setContent function (as we did previously), but let's wrap this call with a condition like if(isButtonClicked).
- So, let's declare a variable isButtonClicked, which would initially be set to false:
- 

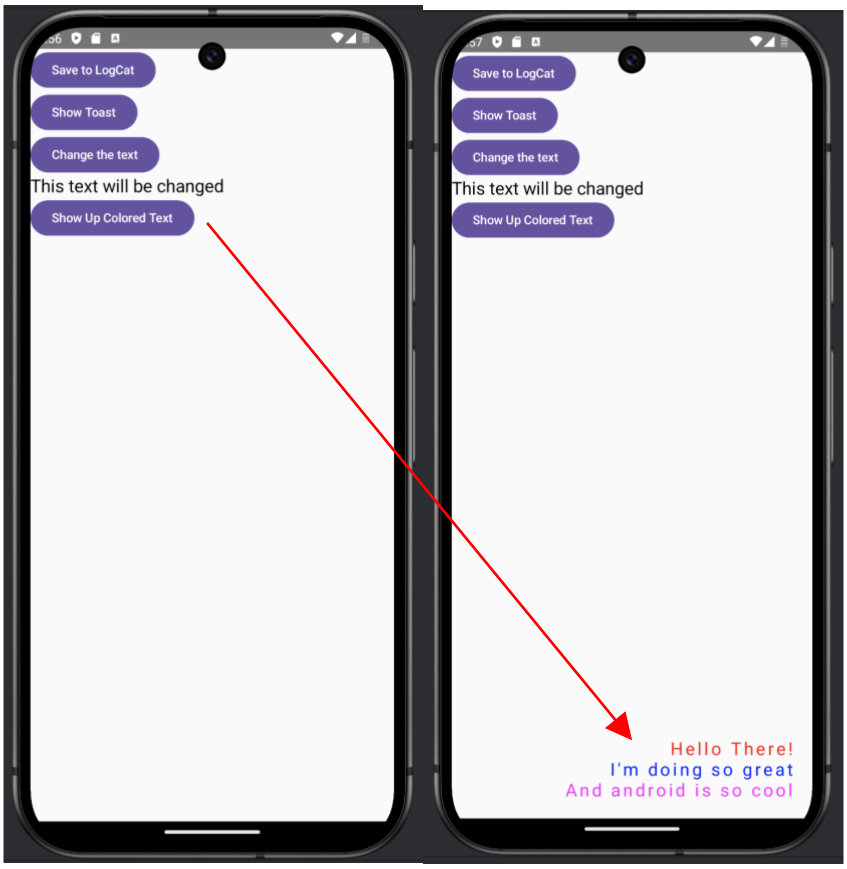
```
setContent {
        MyTexts()
    val ctx = LocalContext.current
    var txt: MutableState<String> = remember {
        mutableStateOf( value: "This text will be changed")
    }
    var isButtonClicked = false
    |
```

- And then let’s use it in our conditional calling of our MyTexts() function, and also let’s change the value of isButtonClicked from false to true whenever our new Button is actually clicked on

- 

```
Button(onClick = {
    isButtonClicked = true
}) {
    Text(text = "Show Up Colored Text")
}

if (isButtonClicked) {
    MyTexts()
}
```

-
- But again, everything seems okay, but it doesn't work because while we modify the value of the isButtonClicked variable, nothing forces to refresh the application/UI state. But we already know how to deal with this. So please make the necessary modifications and verify whether our colored texts will be displayed after clicking. So, the expected behaviour is as follows:

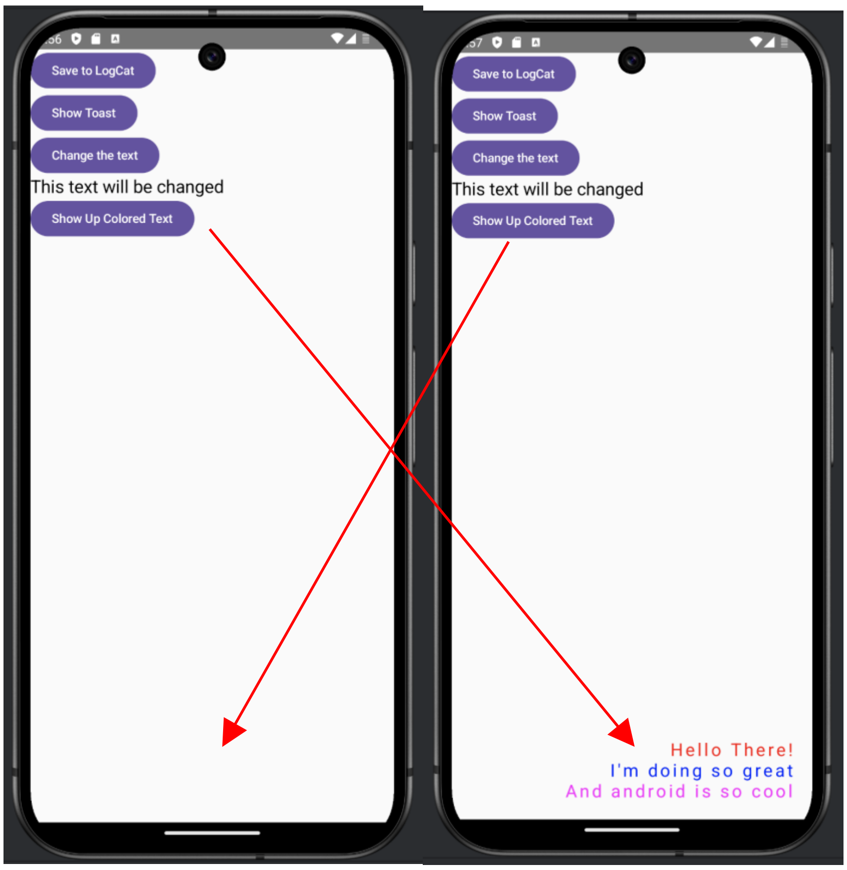


-
-

- Moving on, let's notice that if instead of setting the isButtonClicked value to true directly in the onClick handler, we toggle it between true and false or vice versa, meaning if we do the assignment there like this:

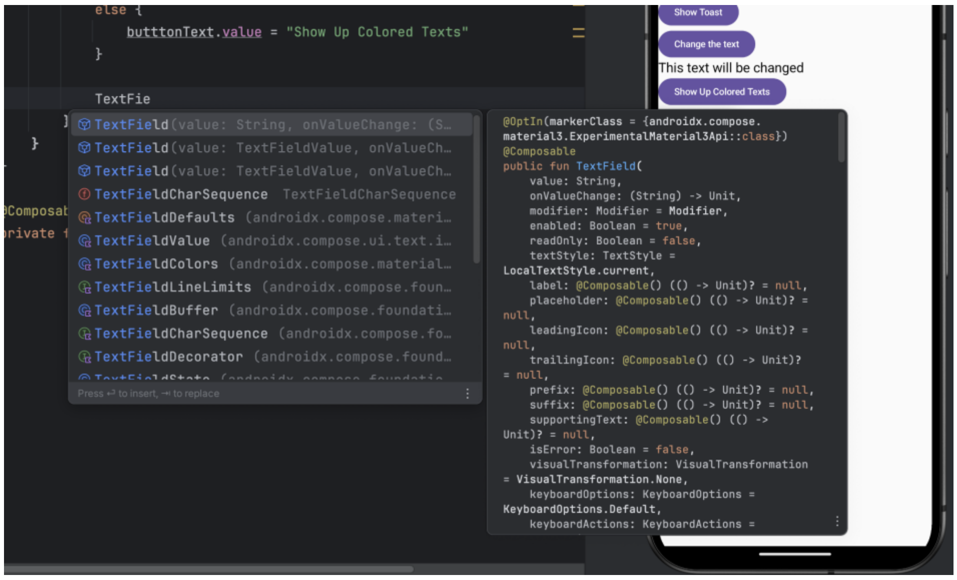
```
isButtonclicked.value = !isButtonclicked.value
```

- We'll get even better functionality allowing us to show up and hide our texts after each successive click. So, we'll get:



- So, just to make it even better, let's change the name of our variable isButtonClicked to something that better reflects the current functionality, for example, shouldTextsBePresented (my suggestion is it via refactoring options). Additionally, we should ensure that our button displays not always the text "Display colored texts" but depending on the situation, either what we have or something like "Hide colored texts". So, please try to make the necessary modifications.
- And one more tweak. We have already in our app the button, which when clicked, changes the text "This text will be changed" to "My new text." Please make the necessary modifications so that after each click on this button, the text changes accordingly from "My old text" to "My new text," and vice versa
- Ok. We've practiced displaying/modifying something on the screen, so let's see how to allow the user to input something in the application. ☺.

- The element we'll use for user input is TextField. So, let's go to our setContent, type TextField, and Studio will show us the available options.

- 

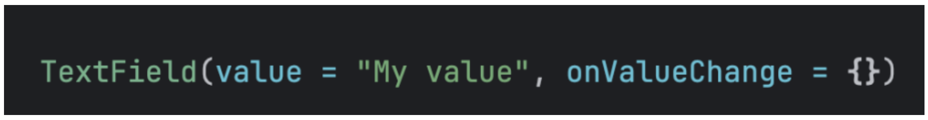
```
else {
    butttonText.value = "Show Up Colored Texts"
}

TextFie
```

```
TextField(value: String, onValueChange: (S…
TextField(value: TextFieldValue, onValueCh…
TextField(value: TextFieldValue, onValueCh…
TextFieldCharSequence  TextFieldCharSequence
TextFieldDefaults (androidx.compose.materi…
TextFieldValue (androidx.compose.ui.text.i…
TextFieldColors (androidx.compose.material…
TextFieldLineLimits (androidx.compose.foun…
TextFieldBuffer (androidx.compose.foundati…
TextFieldCharSequence (androidx.compose.fo…
TextFieldDecorator (androidx.compose.found…
```

Press ↩ to insert, ⇥ to replace

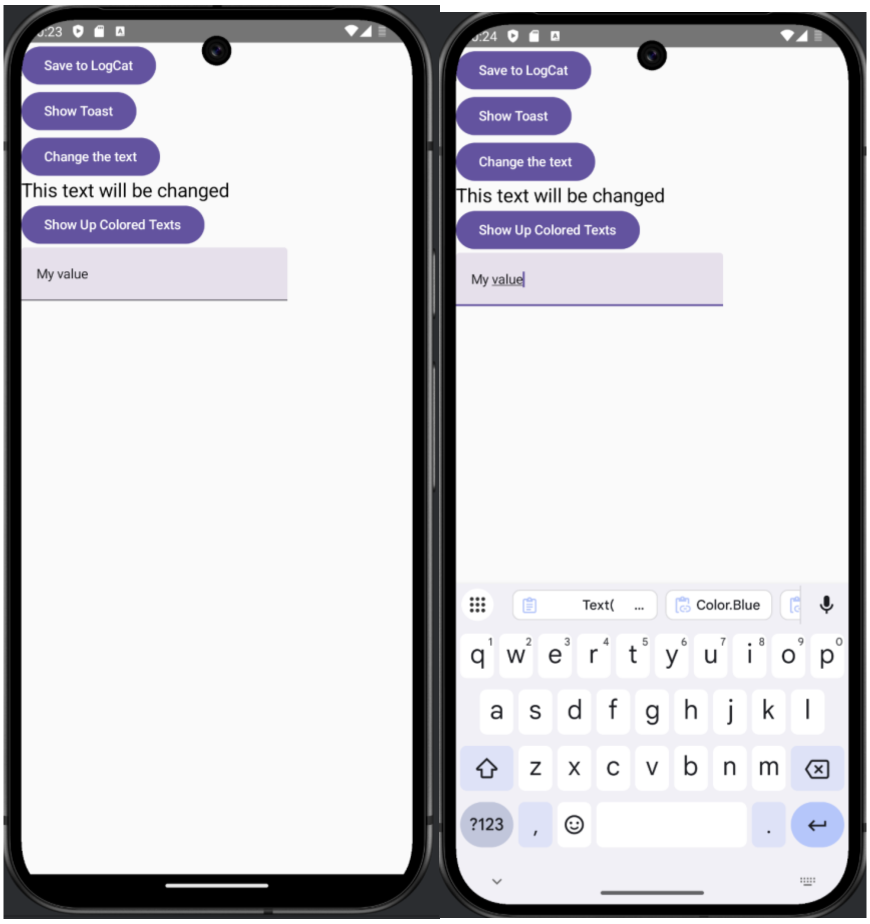
```
@OptIn(markerClass = {androidx.compose.
material3.ExperimentalMaterial3Api::class})
@Composable
public fun TextField(
    value: String,
    onValueChange: (String) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    readOnly: Boolean = false,
    textStyle: TextStyle =
LocalTextStyle.current,
    label: @Composable() (() -> Unit)? = null,
    placeholder: @Composable() (() -> Unit)? =
null,
    leadingIcon: @Composable() (() -> Unit)? =
null,
    trailingIcon: @Composable() (() -> Unit)?
= null,
    prefix: @Composable() (() -> Unit)? = null,
    suffix: @Composable() (() -> Unit)? = null,
    supportingText: @Composable() (() ->
Unit)? = null,
    isError: Boolean = false,
    visualTransformation: VisualTransformation
= VisualTransformation.None,
    keyboardOptions: KeyboardOptions =
KeyboardOptions.Default,
    keyboardActions: KeyboardActions =
```

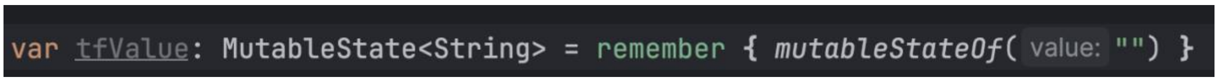
- We're interested in the first option, so let's press Enter and fill in value with "My Value" at the beginning, and for onValueChange lets keep the empty curly brackets,

- 

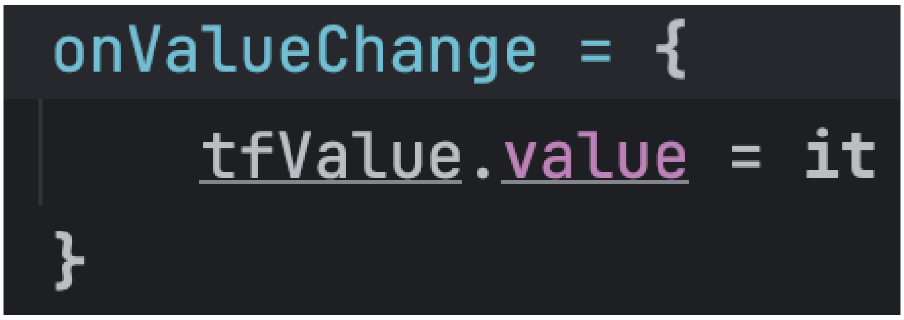
```
TextField(value = "My value", onValueChange = {})
```

- And we should get:

- 

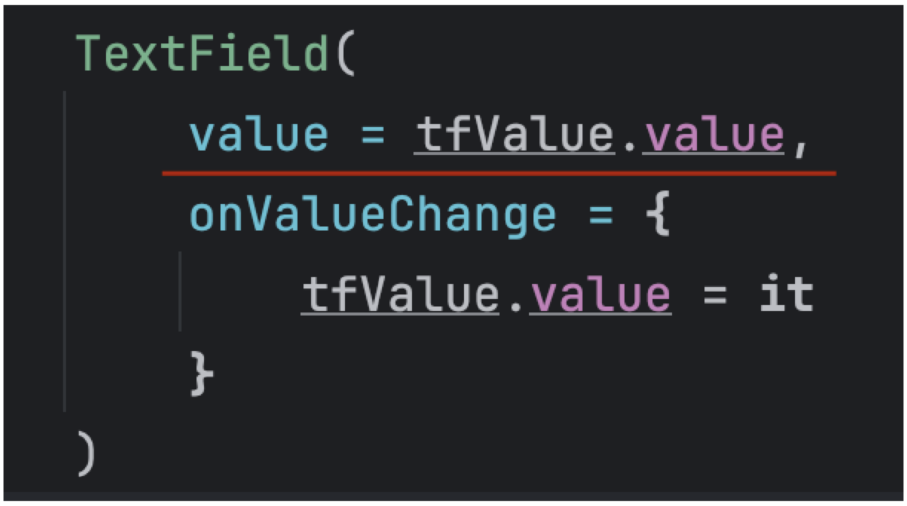
- So, we get a text field, theoretically allowing us to type something (and even the keyboard appears to facilitate this), but no matter what we type, the content of this field doesn't change.
- And there's no surprise because we explicitly set the value for this field to "My Value", and we never change it. To make our field behave as expected, firstly, let's define a variable to store the value of our text field. Generally speaking, we need a string variable, but because we want it to be associated with the application/UI state, let's define it as MutableState<String> and set its initial value to an empty string, like this:
- 

```
var tfValue: MutableState<String> = remember { mutableStateOf( value: "") }
```

- And now, in onValueChange, we may remember the current value of our text field content using “it” value / keyword like this:
- 

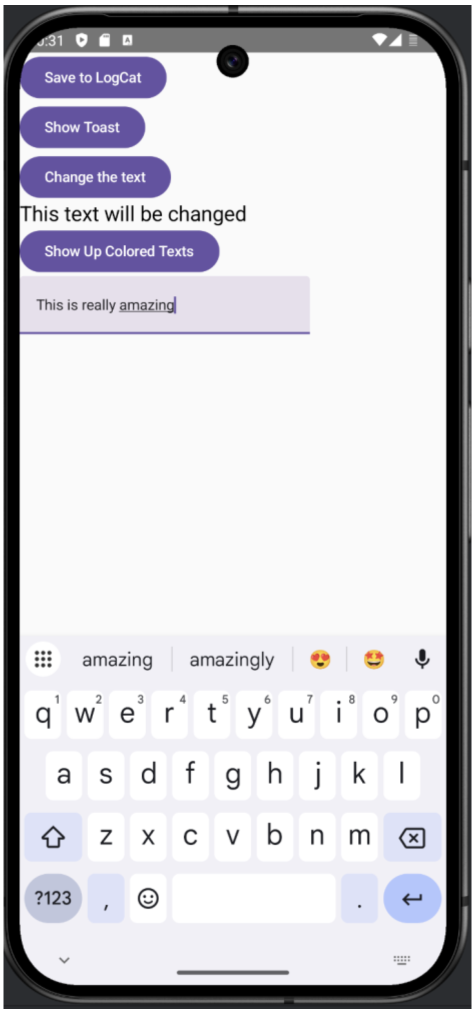
```
onValueChange = {
    tfValue.value = it
}
```

- And having this value we may use it to display the correct content in our TextField like this:



```
TextField(
    value = tfValue.value,
    onValueChange = {
        tfValue.value = it
    }
)
```

- And let's build and test if it works as expected



This text will be changed

This is really amazing

-
- So seems that now everything is fine
- When done, please upload to the UPEL the screenshot of your code and emulator. Thank you.