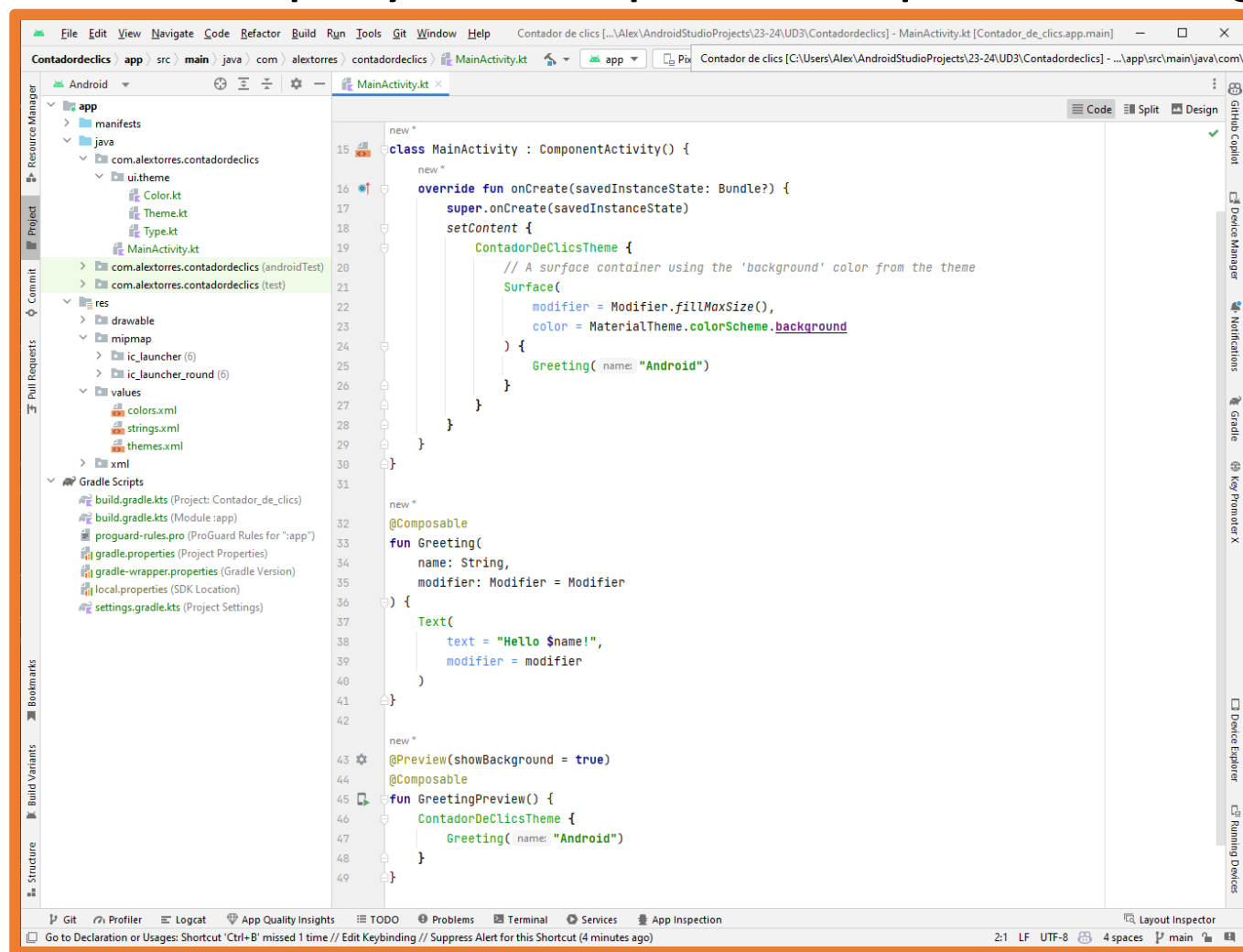


# UD3.2 – Introducción a Android

**2º CFGS**  
**Desarrollo de Aplicaciones Multiplataforma**  
**2023-24**

# 1.- Primera aplicación Android: Contador de clics

El contenido inicial de un proyecto Jetpack Compose es el siguiente:



# 1.- Primera aplicación Android: Contador de clics

Se pueden distinguir tres partes:

## Clase MainActivity:

- Extiende a `ComponentActivity` la cual es una Activity que permite componentes de Jetpack Compose.
- Esta función contiene el método `onCreate` que será el que se ejecute al iniciar la aplicación.
- Dentro de `onCreate` se carga el tema del proyecto y dentro de él se llama a un componente `Surface` que a su vez llama a la función `Greeting`.

## Función Greeting:

- Recibe un `String` y un `Modifier` y genera un componente `Text` de Jetpack Compose.
- Esta función es un componente de Jetpack Compose ya que está etiquetada con **@Composable**.

## Función GreetingPreview:

- Carga el tema del proyecto y dentro de él llama a la función `Greeting`.
- Esta función es un componente de Jetpack Compose ya que está etiquetada con **@Composable**.
- Esta función permite que se pueda previsualizar su contenido al estar etiquetada con **@Preview**.

```
class MainActivity : ComponentActivity() {
    new *
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ContadorDeClicsTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting(name: "Android")
                }
            }
        }
    }
}

new *
@Composable
fun Greeting(
    name: String,
    modifier: Modifier = Modifier
) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}

new *
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    ContadorDeClicsTheme {
        Greeting(name: "Android")
    }
}
```

# 1.- Primera aplicación Android: Contador de clics

En el código se pueden ver los siguientes componentes Jetpack Compose:

**Surface:** componente del sistema que utiliza Material Design que permite definir una elevación, un fondo...

**Text:** componente del sistema para mostrar texto

**ContadorDeClicksTheme:** componente propio que se crea con el proyecto y extiende al tema por defecto para Material Design. Está definido en el archivo `ui.theme/Theme.kt`.

**Greeting:** componente propio que extiende la funcionalidad del componente `Text`.

**GreetingPreview:** componente propio que sirve para previsualizar el componente `Greeting`.

# 1.- Primera aplicación Android: Contador de clics

## @Composable

Todos los componentes Jetpack Compose, ya sean del sistema o propios, son funciones que deben estar etiquetadas con **@Composable**.

Propios

```
@Composable  
fun Greeting(  
    name: String,  
    modifier: Modifier = Modifier  
) {
```

```
@Composable  
fun ContadorDeClicsTheme(  
    darkTheme: Boolean = isSystemInDarkTheme(),  
    // Dynamic color is available on Android 12+  
    dynamicColor: Boolean = true,  
    content: @Composable () -> Unit  
) {
```

Del sistema

```
@Composable  
@NonRestartableComposable  
fun Surface(  
    modifier: Modifier = Modifier,  
    shape: Shape = RectangleShape,  
    color: Color = MaterialTheme.co
```

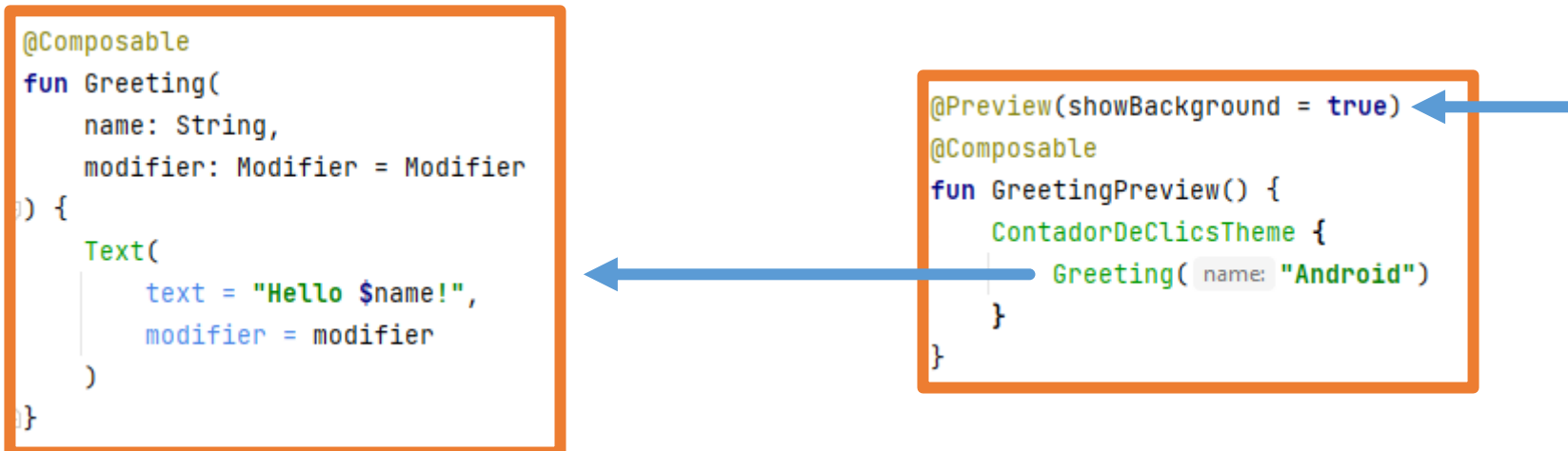
```
@Composable  
fun Text(  
    text: String,  
    modifier: Modifier = Modifier,  
    color: Color = Color.Unspecified,  
    fontSize: TextUnit = TextUnit.Unspecified,
```

# 1.- Primera aplicación Android: Contador de clics

## @Preview

Android Studio permite ver **una previsualización en tiempo real** de los componentes que se definan, para ello se debe etiquetar un componente con **@Preview** como ocurre con la función GreetingPreview.

No se pueden previsualizar componentes que reciben funciones, para solucionar esto se crean componentes que envuelvan a esos que reciben funciones.



# 1.- Primera aplicación Android: Contador de clics

## @Preview

Es muy importante que la previsualización muestre lo mismo que se mostrará en la ejecución de la aplicación por eso se puede realizar el siguiente cambio:

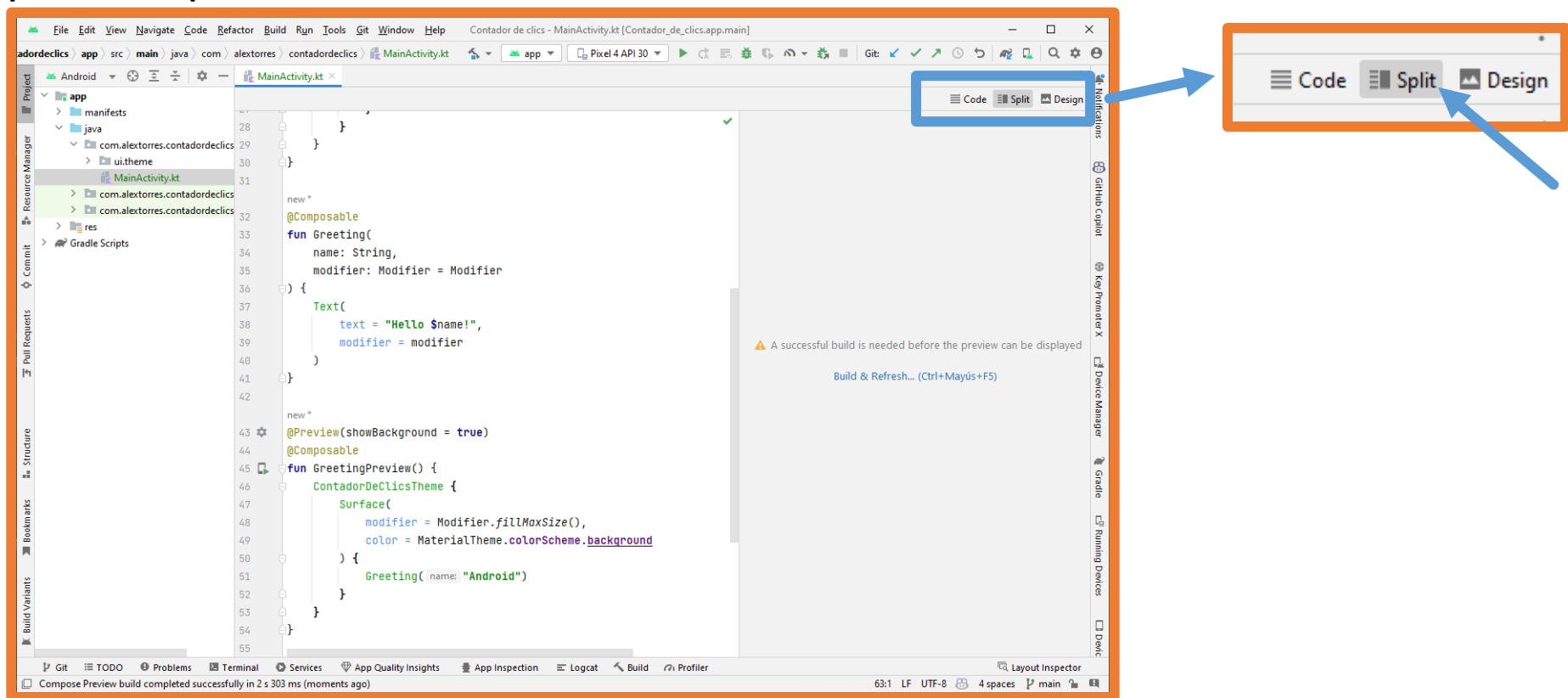
```
class MainActivity : ComponentActivity() {  
    new *  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            ContadorDeClicsTheme {  
                // A surface container using the 'background' color from the theme  
                Surface(  
                    modifier = Modifier.fillMaxSize(),  
                    color = MaterialTheme.colorScheme.background  
                ) {  
                    Greeting( name: "Android")  
                }  
            }  
        }  
    }  
}
```

```
@Preview(showBackground = true)  
@Composable  
fun GreetingPreview() {  
    ContadorDeClicsTheme {  
        Surface(  
            modifier = Modifier.fillMaxSize(),  
            color = MaterialTheme.colorScheme.background  
        ) {  
            Greeting( name: "Android")  
        }  
    }  
}
```

# 1.- Primera aplicación Android: Contador de clics

## @Preview

Para poder ver las previsualizaciones en Android Studio se debe seleccionar la opción **Split** en la parte superior derecha.

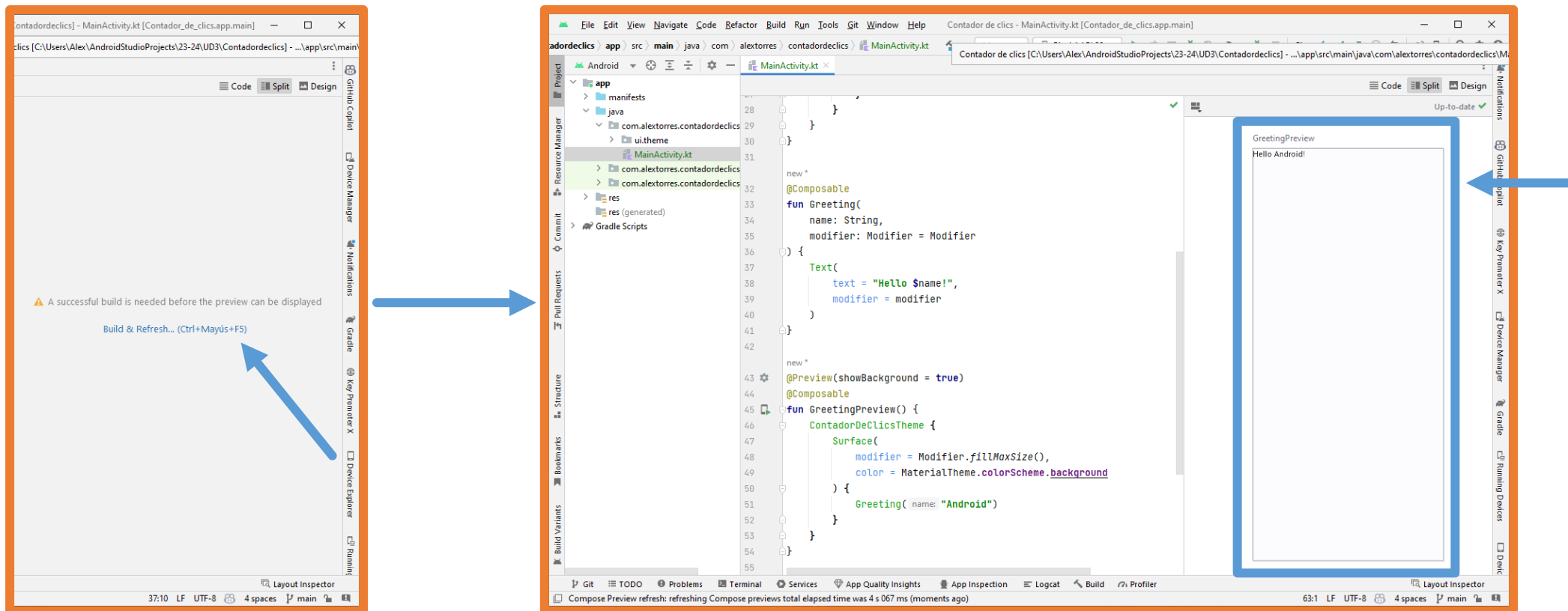




# 1.- Primera aplicación Android: Contador de clics

## @Preview

La primera vez que se quiere previsualizar un componente y cuando hay cambios grandes o errores en el build se deberá pulsar en **Build & Refresh...**



# 1.- Primera aplicación Android: Contador de clics

## @Preview – Opciones

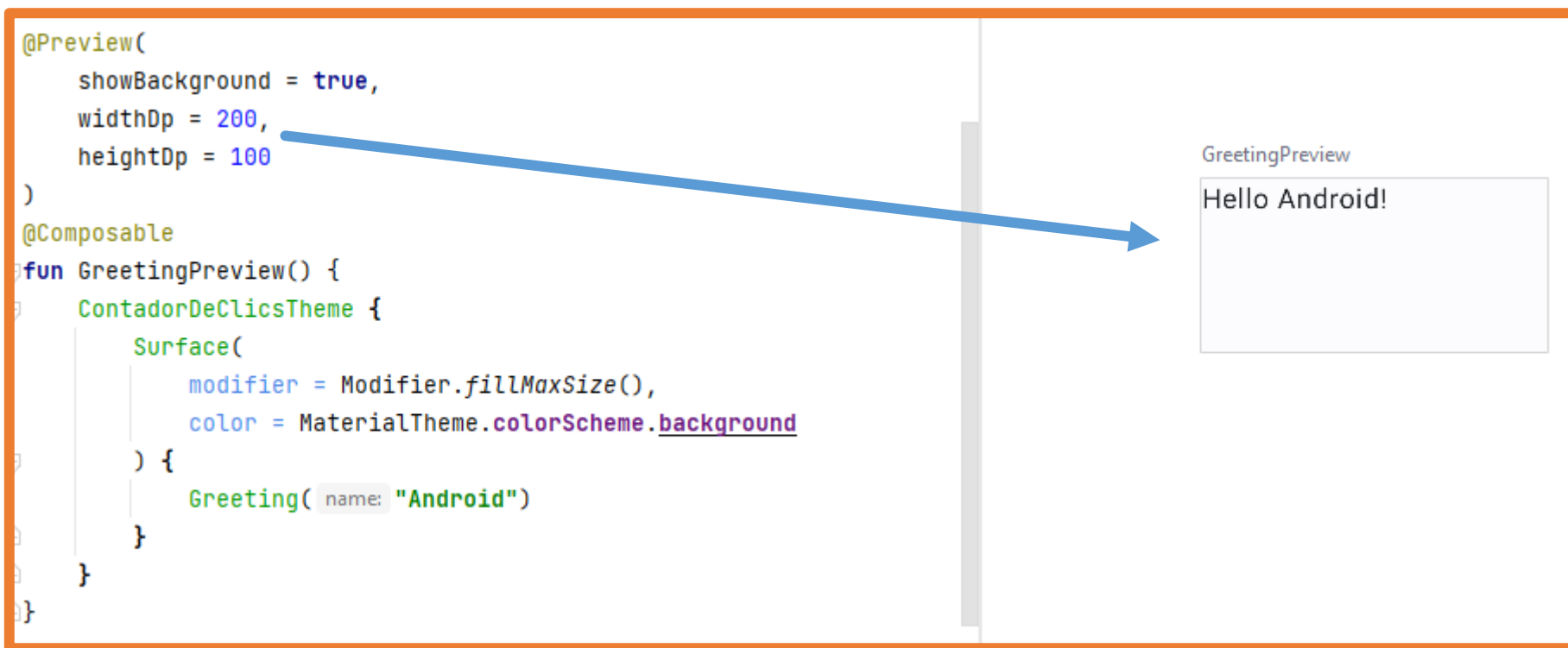
Pulsando la tecla CONTROL y haciendo clic sobre @Preview se pueden ver todas las opciones disponibles:

```
@annotation class Preview(  
    val name: String = "",  
    val group: String = "",  
    @IntRange(from = 1) val apiLevel: Int = -1,  
    // TODO(mount): Make this Dp when they are inline classes  
    val widthDp: Int = -1,  
    // TODO(mount): Make this Dp when they are inline classes  
    val heightDp: Int = -1,  
    val locale: String = "",  
    @FloatRange(from = 0.01) val fontScale: Float = 1f,  
    val showSystemUi: Boolean = false,  
    val showBackground: Boolean = false,  
    val backgroundColor: Long = 0,  
    @UiMode val uiMode: Int = 0,  
    @Device val device: String = Devices.DEFAULT,  
    @Wallpaper val wallpaper: Int = Wallpapers.NONE,  
)
```

# 1.- Primera aplicación Android: Contador de clics

## @Preview – Opciones

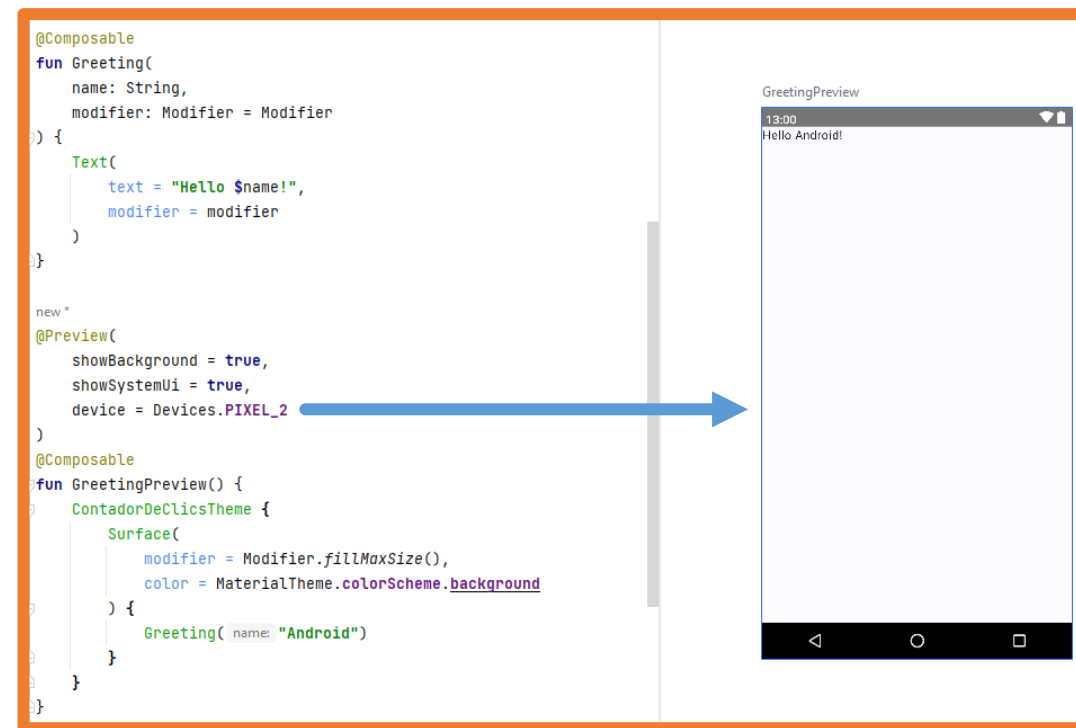
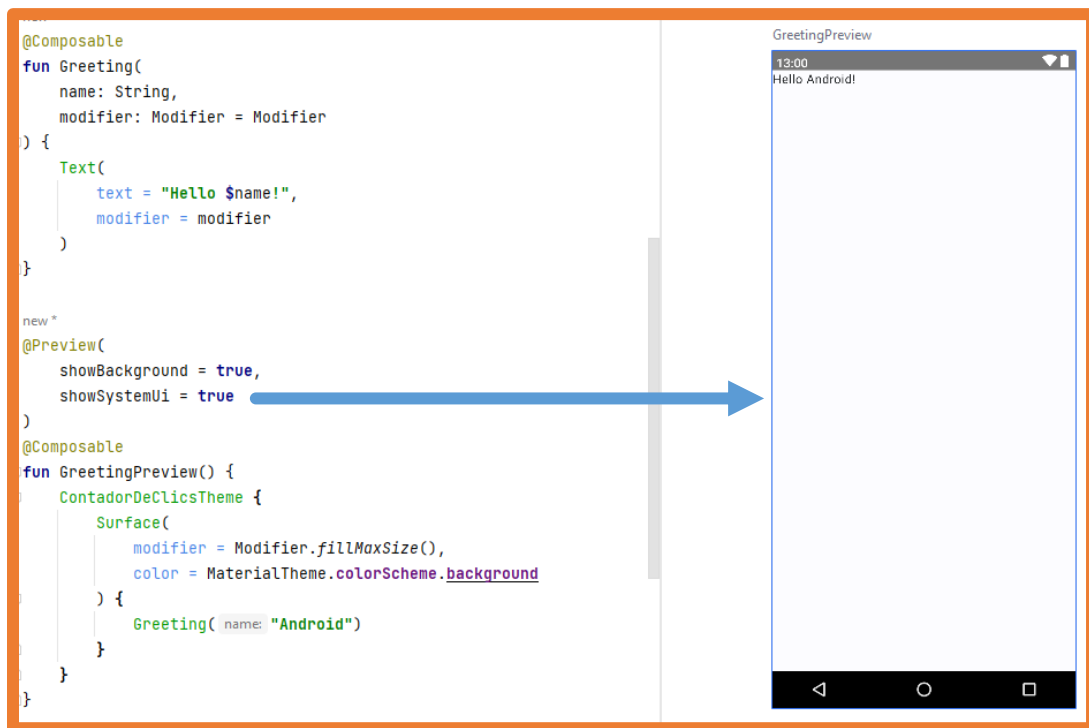
Por ejemplo, se puede indicar un tamaño a la previsualización:



# 1.- Primera aplicación Android: Contador de clics

## @Preview – Opciones

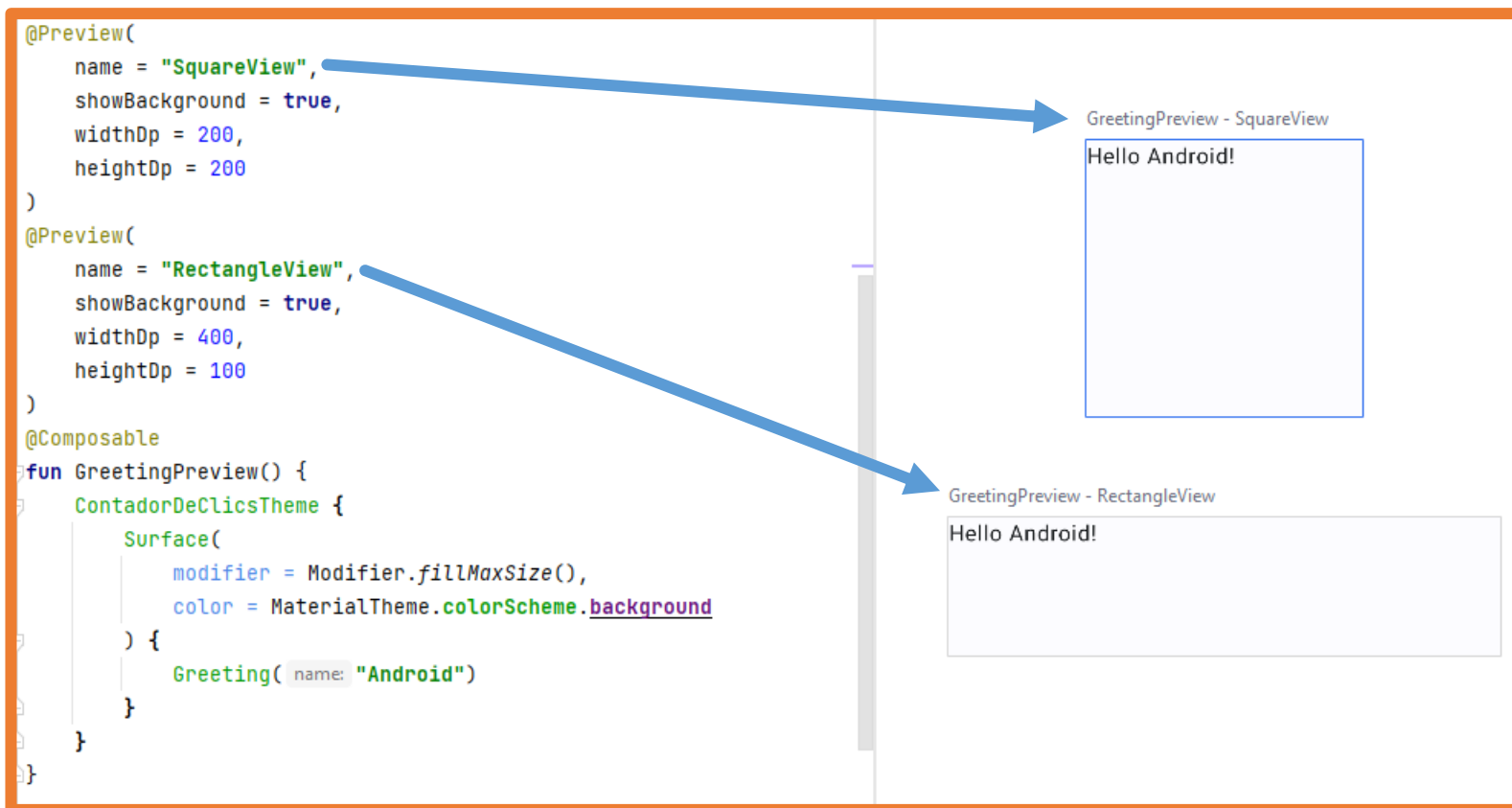
Se puede ver el componente dentro de la interfaz del sistema o incluso indicar un dispositivo concreto:



# 1.- Primera aplicación Android: Contador de clics

## @Preview – Opciones

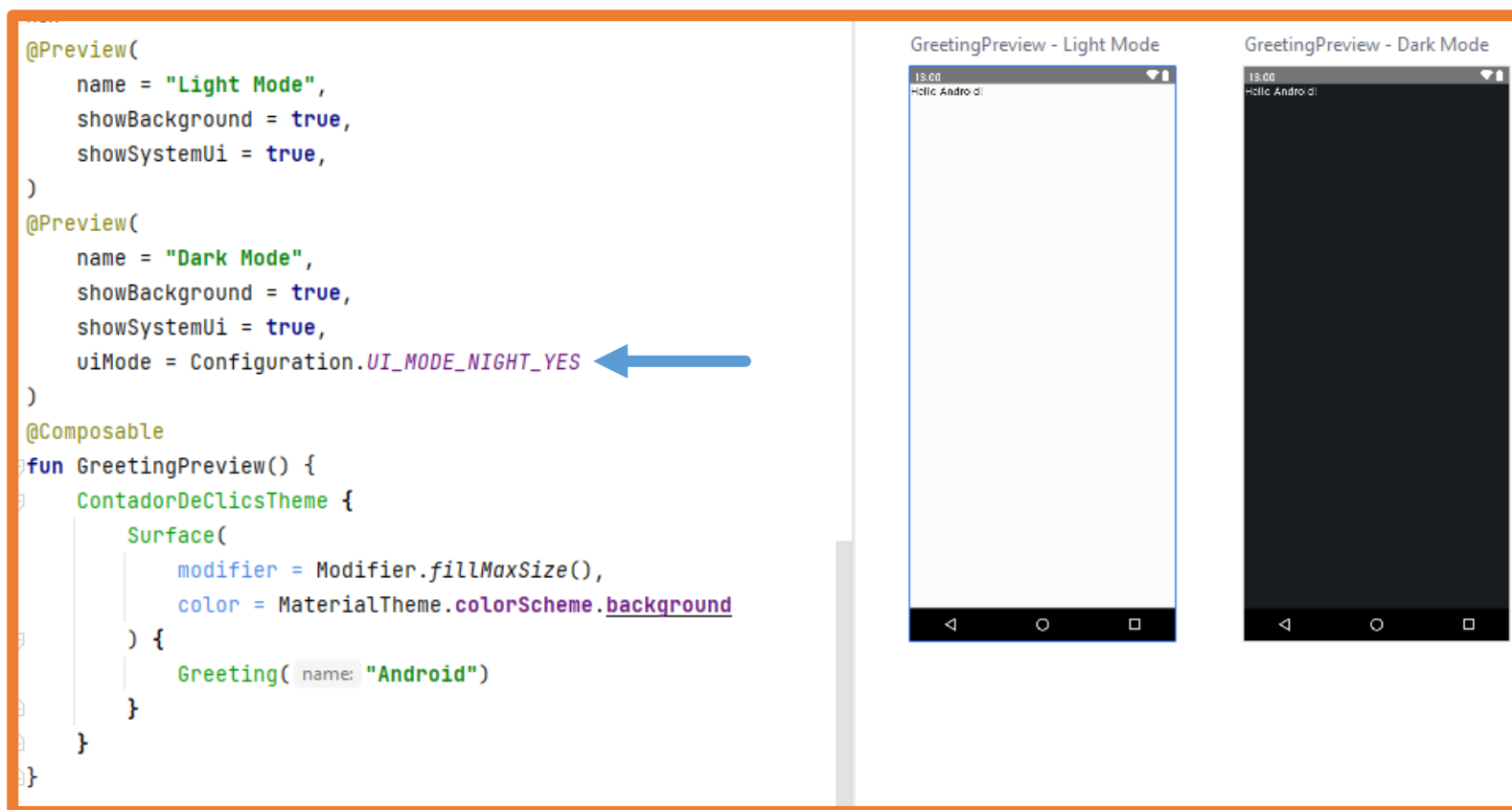
Se pueden crear varias previsualizaciones para un componente:



# 1.- Primera aplicación Android: Contador de clics

## @Preview – Opciones

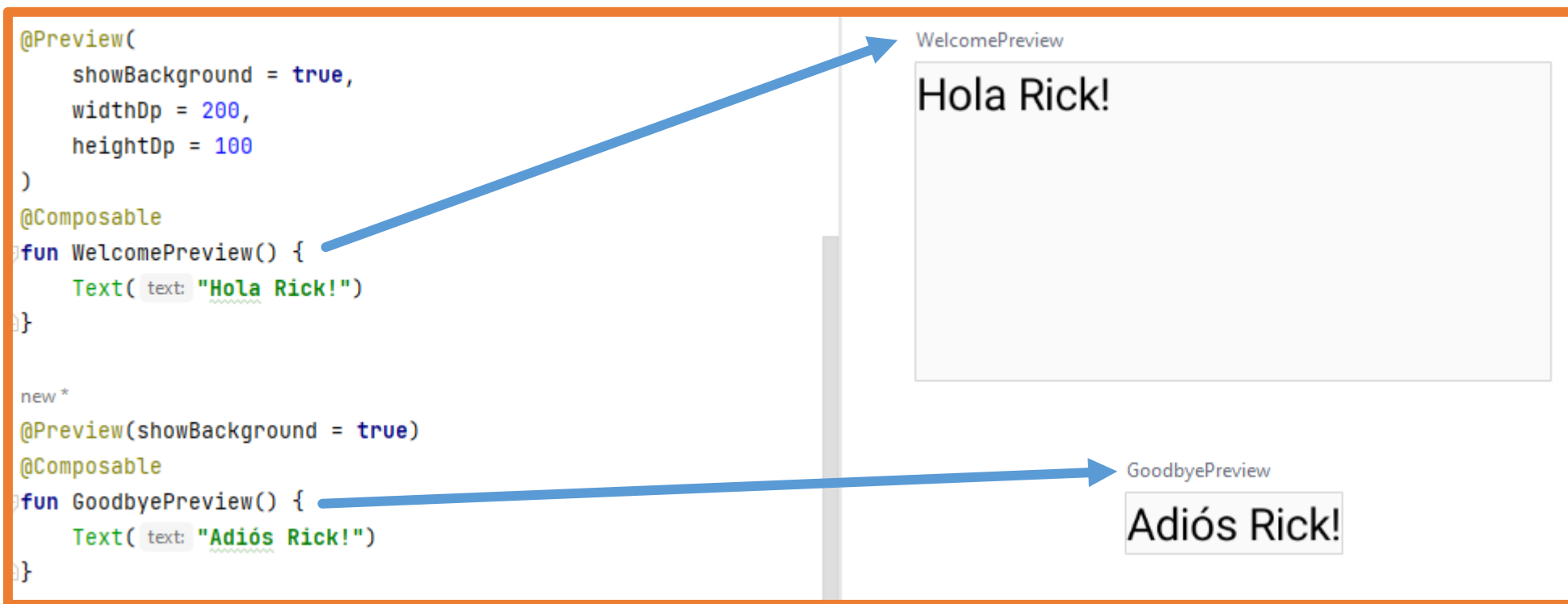
Esto tiene especial utilidad para mostrar los modos claro y oscuro.



# 1.- Primera aplicación Android: Contador de clics

## @Preview – Opciones

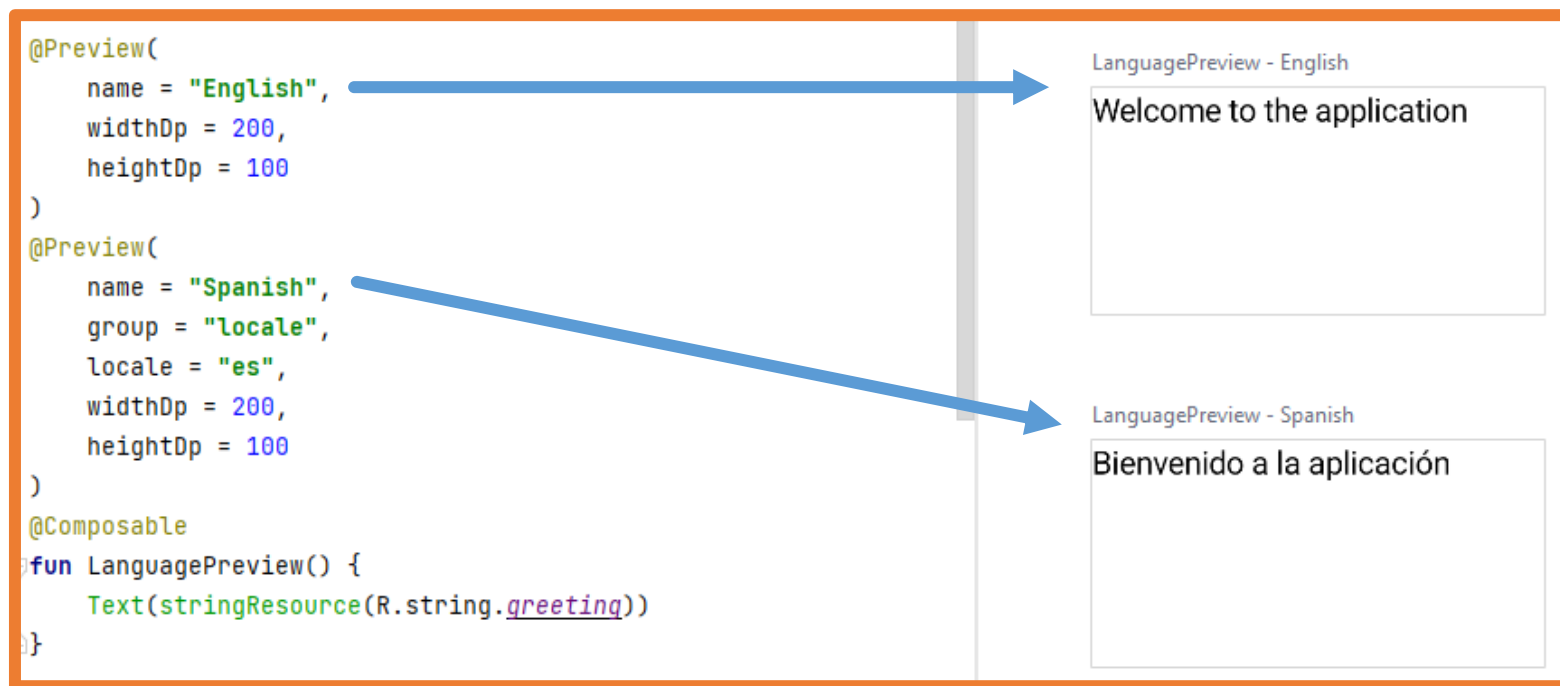
También pueden haber varios componentes con previsualización:



# 1.- Primera aplicación Android: Contador de clics

## @Preview – Opciones

Las previsualizaciones también permiten ver cómo quedan los componentes cuando se está desarrollando una aplicación multi idioma:

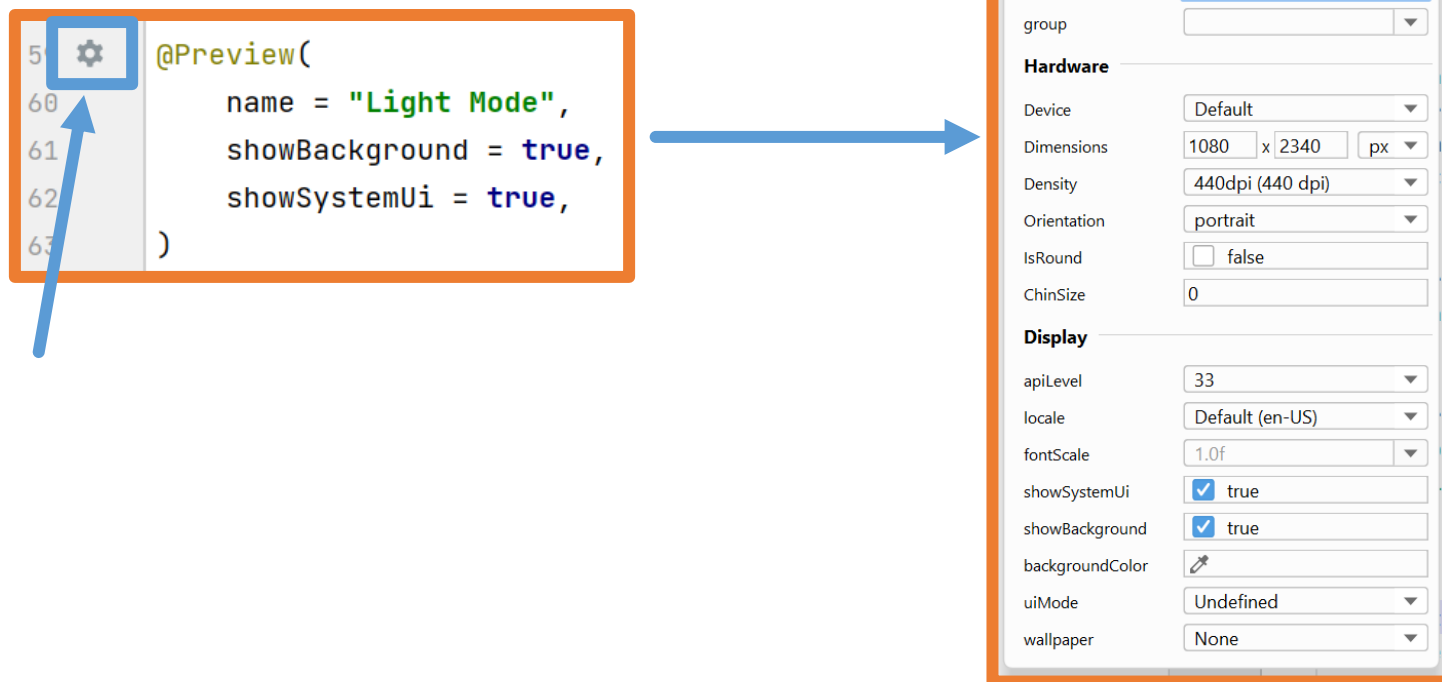




# 1.- Primera aplicación Android: Contador de clics

## @Preview – Opciones

Cuando se crea una previsualización Android Studio marca esa línea para poder acceder a todas las opciones disponibles desde un menú contextual:



# 1.- Primera aplicación Android: Contador de clics

## @Preview – Limitaciones

Las previsualizaciones tienen una serie de limitaciones:

- No pueden recibir parámetros.
- No tienen acceso a los archivos.
- No tienen acceso a la red (no cargarán datos de internet).
- Algunas API no funcionan completamente bien.

En la documentación oficial está toda la información sobre [@Preview](#).

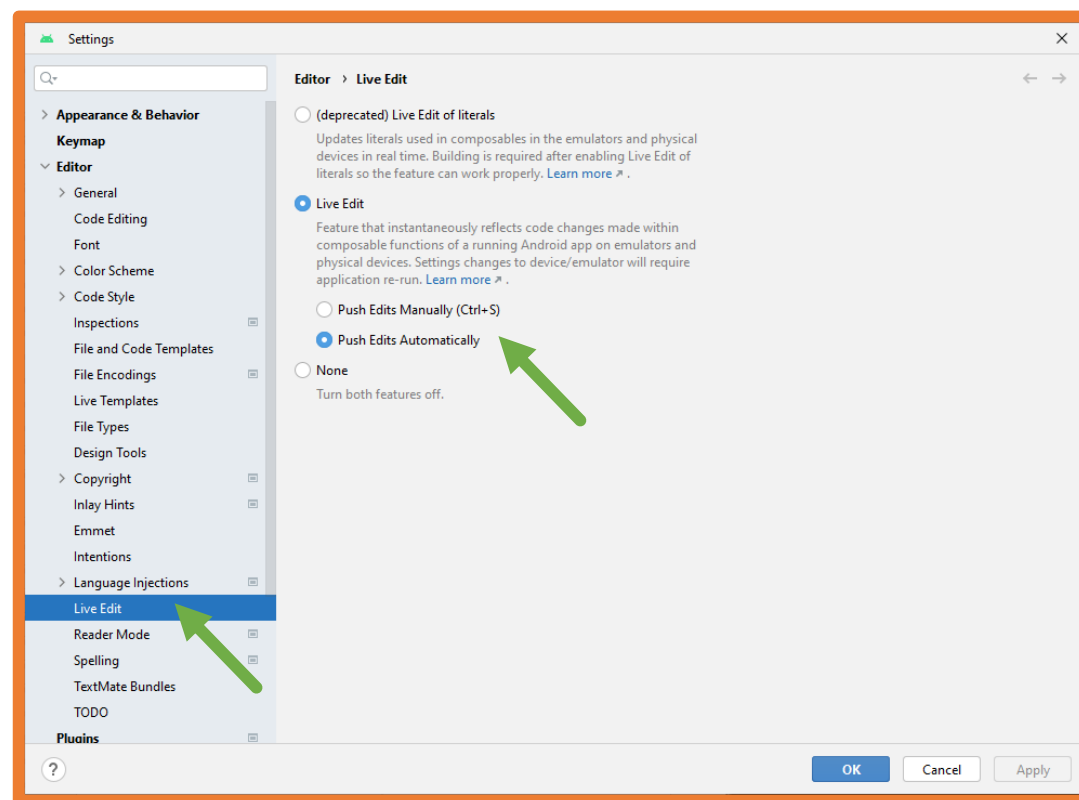
Las últimas versiones de Android Studio han mejorado la ejecución de Jetpack Compose y permiten visualizar los cambios en tiempo real → **Live Edit**.

# 1.- Primera aplicación Android: Contador de clics

## Live Edit

Si se quiere que los cambios en el código actualicen automáticamente la aplicación en el emulador, se debe configurar la opción **Live Edit** de Android Studio.

File → Settings (CONTROL+ALT+S)



# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación

La aplicación **Contador de clics** necesita un texto y un botón así que se va a modificar el código para que lo incluya.

Lo primero será **eliminar la función Greeting y todas sus llamadas**.

También se cambiará el nombre de la preview GreetingPreview por el nombre **ContentPreview**.

A continuación, se creará un componente Jetpack Compose llamado **Content**.

```
class MainActivity : ComponentActivity() {
    new *
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ContadorDeClicsTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                }
            }
        }
    }
}

new *
@Composable
fun Content() {
}

new *
@Preview(showBackground = true)
@Composable
fun ContentPreview() {
    ContadorDeClicsTheme {
        Surface(
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {
        }
    }
}
}
```

# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación

En el componente **Content** se añaden tanto el texto como el botón.

El texto se añade con un componente **Text** y el botón se añade con un componente **Button**.

Se puede observar que el componente **Button** recibe una función lambda como parámetro **onClick** y otra función lambda como contenido del propio botón.

El contenido del botón en este caso es un texto.

```
@Composable
fun Content() {
    Text(text: "Has hecho clic 0 veces")
    Button(onClick = {

    }) { this: RowScope
        Text(text: "¡PÚLSAME!")
    }
}
```


# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación


Como se estudió en la UD2, en Kotlin si el último parámetro es una función lambda, se puede extraer ese parámetro fuera de los paréntesis.

Aunque las dos maneras funcionan igual, en Jetpack Compose si el último parámetro es una función lambda se extrae fuera de los paréntesis.

```
@Composable
fun Content() {
    Text(text: "Has hecho clic 0 veces")
    Button(
        onClick = {
        },
        content = { this: RowScope
            Text(text: "¡PÚLSAME!")
        }
    )
}
```



```
@Composable
fun Content() {
    Text(text: "Has hecho clic 0 veces")
    Button(onClick = {
    }) { this: RowScope
        Text(text: "¡PÚLSAME!")
    }
}
```



# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación

Para poder previsualizar la interfaz gráfica y ver la interfaz gráfica en el emulador se deben añadir llamadas a la función **Content** tanto en **onCreate** como en **ContentPreview**.

```
class MainActivity : ComponentActivity() {  
    new *  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            ContadorDeClicsTheme {  
                Surface(  
                    modifier = Modifier.fillMaxSize(),  
                    color = MaterialTheme.colorScheme.background  
                ) {  
                    Content() ←  
                }  
            }  
        }  
    }  
}
```

```
@Preview(showBackground = true)  
@Composable  
fun ContentPreview() {  
    ContadorDeClicsTheme {  
        Surface(  
            modifier = Modifier.fillMaxSize(),  
            color = MaterialTheme.colorScheme.background  
        ) {  
            Content() ←  
        }  
    }  
}
```

# 1.- Primera aplicación Android: Contador de clics

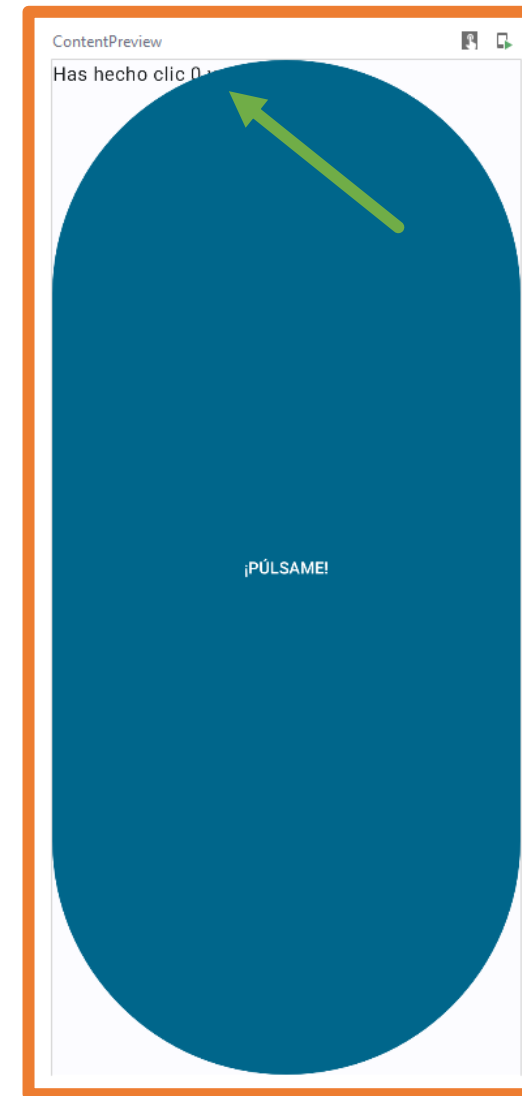
## Contenido de la aplicación

En la previsualización se puede observar que los dos componentes ocupan toda la pantalla (se nota más con el botón).

También se puede observar que los dos componentes se superponen, esto es debido a que no hay ningún componente de tipo **layout** en la interfaz.

Los componentes de tipo **layout** permiten organizar los componentes de la interfaz gráfica.

Para solucionar esto se va a utilizar el componente **Column** que permite organizar la interfaz en forma de columna.

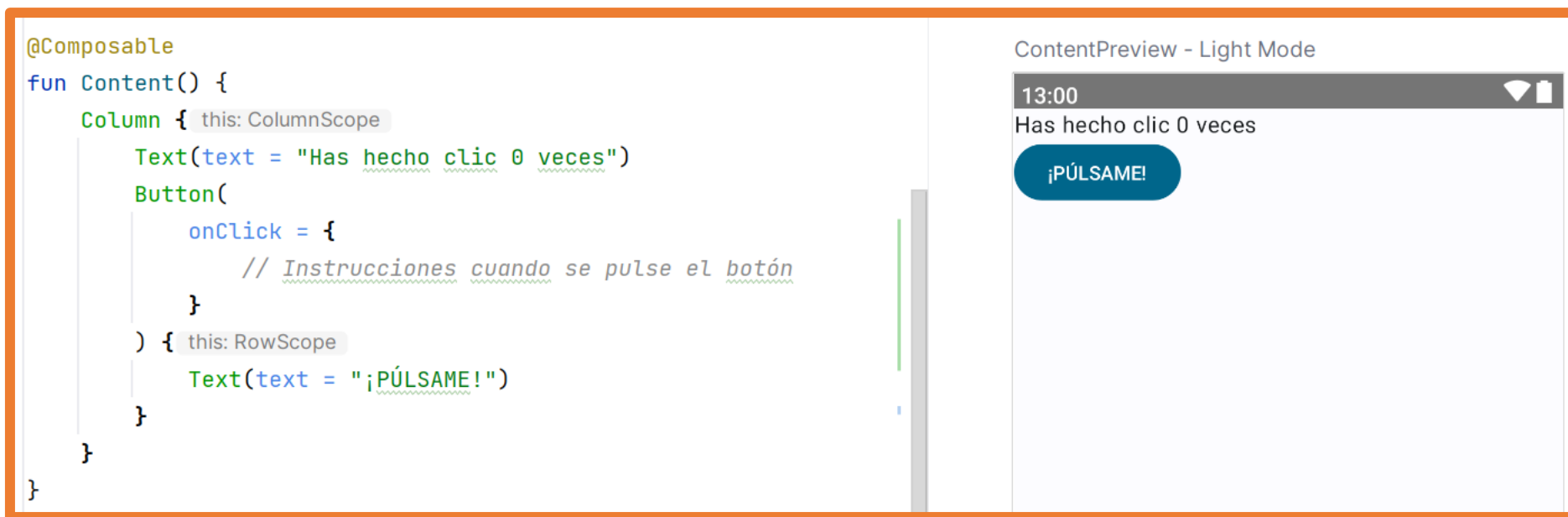




# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación

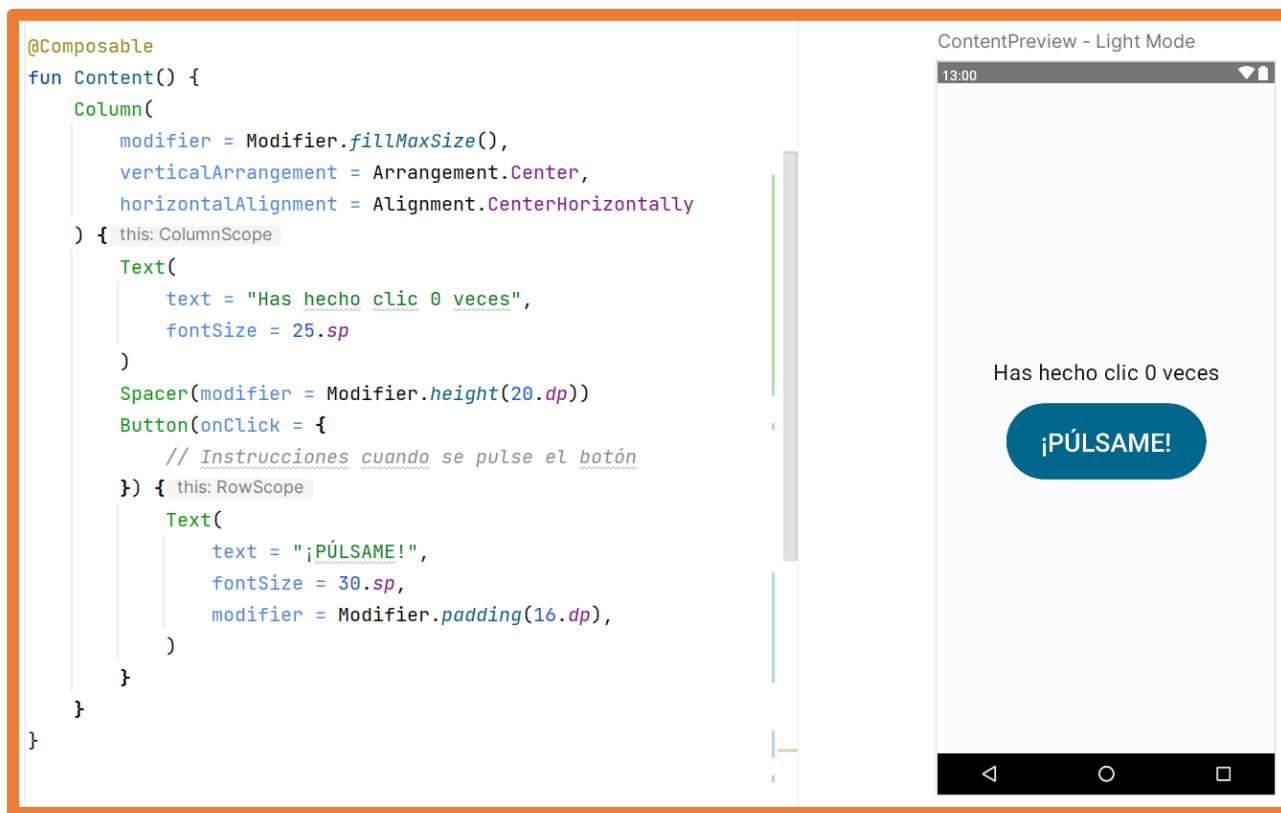
Se añade el componente **Column** y se introducen en él tanto el texto como el botón.



# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación

A continuación, se añaden algunas modificaciones para mejorar la interfaz.



# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación

Este código es **una muestra de las buenas prácticas** programando en Kotlin con Jetpack Compose:

- Se utilizan los parámetros con nombre en las llamadas.
- Si en la llamada hay varios parámetros, estos se escriben cada uno en una línea.
- Si el último parámetro es una función lambda se extrae de los paréntesis.

```
@Composable
fun Content() {
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) { this: ColumnScope
        Text(
            text = "Has hecho clic 0 veces",
            fontSize = 25.sp
        )
        Spacer(modifier = Modifier.height(20.dp))
        Button(onClick = {
            // Instrucciones cuando se pulse el botón
        }) { this: RowScope
            Text(
                text = "¡PÚLSAME!",
                fontSize = 30.sp,
                modifier = Modifier.padding(16.dp),
            )
        }
    }
}
```

# 1.- Primera aplicación Android: Contador de clics

## Funcionalidad de la aplicación

En este punto ya se ha terminado con la UI declarativa.

Ahora es el momento de implementar la funcionalidad de la aplicación.

Se necesita una variable de tipo entera para almacenar la cantidad de veces que se ha hecho clic.

Se crea esa variable inicializada a cero y se incluye en el primer texto.

En el código del botón se añade uno a esa variable.

```
@Composable
fun Content() {
    var times = 0
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) { this: ColumnScope
        Text(
            text = "Has hecho clic $times veces",
            fontSize = 25.sp
        )
        Spacer(modifier = Modifier.height(20.dp))
        Button(onClick = {
            times++
        }) { this: RowScope
            Text(
                text = "¡PÚLSAME!",
                fontSize = 30.sp,
                modifier = Modifier.padding(16.dp),
            )
        }
    }
}
```

# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación

Si se ejecuta en el emulador la aplicación, aunque se puede comprobar que el botón funciona no se actualiza el número de veces.

Esto es debido a que como se indicó anteriormente los elementos de la interfaz se conectan al **estado de la Activity** y si el estado de la Activity no se actualiza la interfaz no se vuelve a pintar.

Para solucionar esto se debe cambiar la variable para que sea una **variable de estado** de esta manera cuando esta variable cambie el estado de la Activity también lo hará y se volverá a pintar la interfaz.

# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación

```
@Composable
fun Content() {
    var times by remember { mutableStateOf( value: 0) }
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
```

Con este cambio se puede ejecutar la aplicación y comprobar que la aplicación funciona.

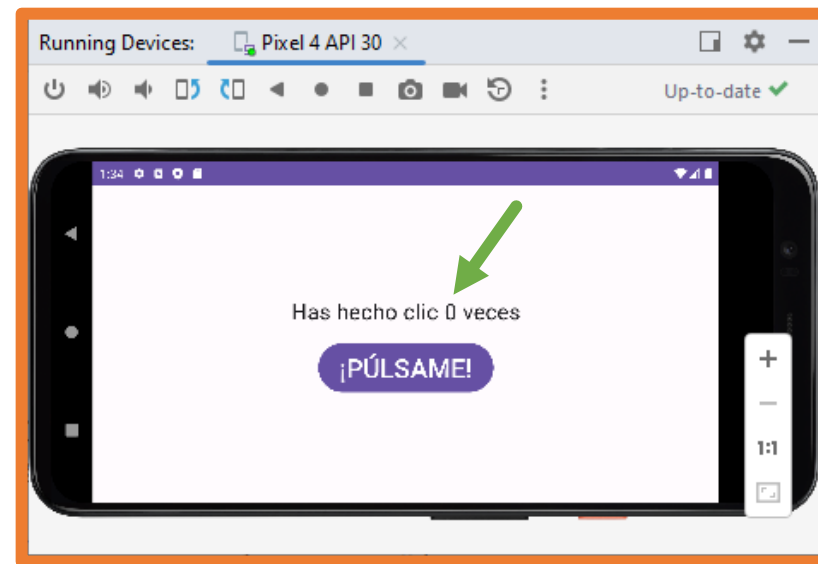
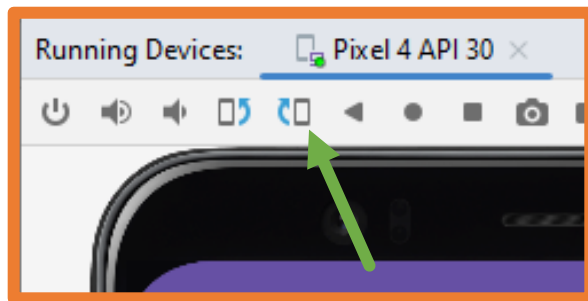
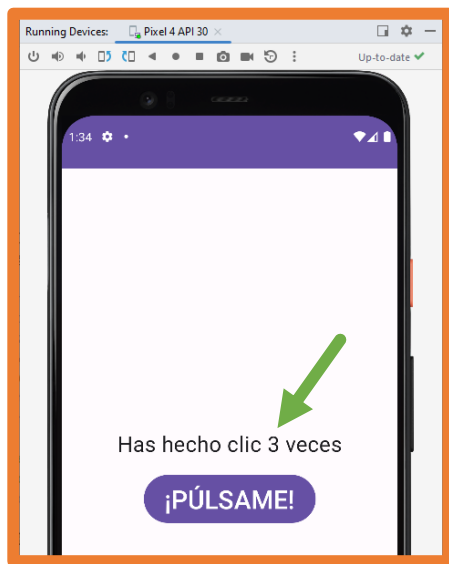
# 1.- Primera aplicación Android: Contador de clics

## Contenido de la aplicación – Estados de Activity

Al principio de la unidad se explicó que hay situaciones en las que la Activity se destruye y se vuelve a crear, por ejemplo, al cambiar la orientación del dispositivo.

Cuando esto ocurre se ejecuta la Activity desde el principio por lo que las variables se vuelven a crear.

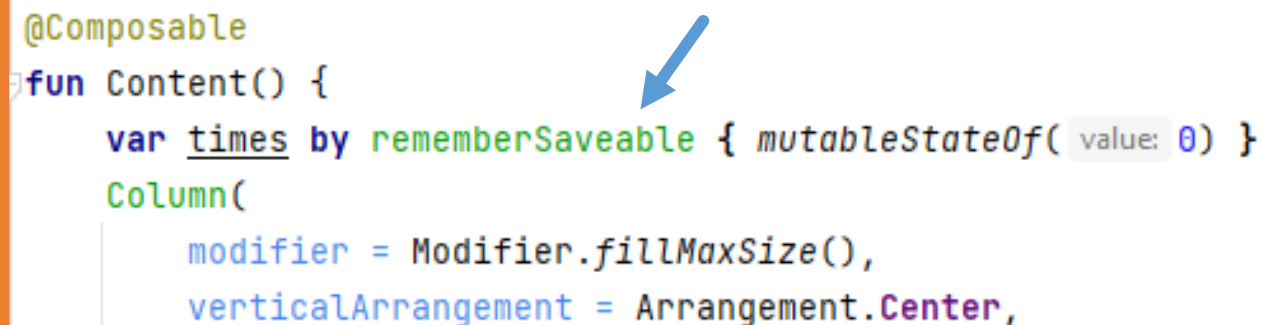
Si se cambia la orientación en la aplicación se observará que el número de clics se pierde.



# 1.- Primera aplicación Android: Contador de clics

## Estados de Activity

Para solucionar este comportamiento se debe cambiar la declaración de la variable para que se guarde aunque se destruya la Activity.



```
@Composable
fun Content() {
    var times by rememberSaveable { mutableStateOf(0) }
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
```

A blue arrow points to the `rememberSaveable` function in the code snippet, highlighting the change from `remember` to `rememberSaveable` to ensure state persistence across Activity restarts.

Con este cambio se puede ejecutar la aplicación y comprobar que la aplicación funciona correctamente aunque se cambie de orientación, de modo claro/oscuro o incluso la configuración del dispositivo.



# Práctica

## **Actividad 0**

Contador de clics

## **Actividad 1**

Estadísticas

## **Actividad 2**

Conversor