🚀 Ejercicios de Jetpack Compose - Nivel

Principiante

Ejercicio 1: Textos Básicos

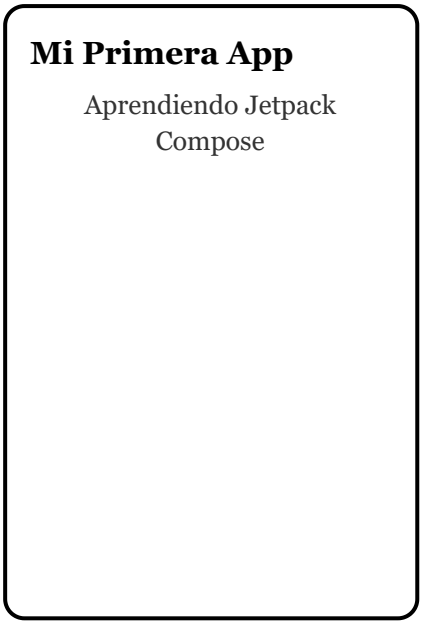
Objetivo: Crear una interfaz simple con dos textos, uno debajo del otro, simulando un título y un subtítulo.

📝 Instrucciones:

- Crea una función **@Composable** llamada `TituloSubtitulo()`
- Usa un **Column** para organizar los elementos verticalmente
- Añade dos componentes **Text**
- El primer texto debe ser el título con tamaño de fuente grande
- El segundo texto debe ser el subtítulo con tamaño menor

```
@Composable
fun TituloSubtitulo() {
    Column {
        Text(
            text = "Mi Primera App",
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold
        )
        Text(
            text = "Aprendiendo Jetpack Compose",
            fontSize = 16.sp
        )
    }
}
```

📱 **Resultado Esperado:**



Ejercicio 2: Centrado de Contenido

Objetivo: Tomar el ejercicio anterior y centrar todo el contenido tanto vertical como horizontalmente en la pantalla.

📝 Instrucciones:

- Modifica la función del ejercicio anterior
- Añade **`Modifier.fillMaxSize()`** al Column
- Usa **`verticalArrangement`** y **`horizontalAlignment`**
- Centra el contenido usando `Arrangement.Center` y `Alignment.CenterHorizontally`

```
@Composable
fun TituloSubtituloCentrado() {
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = "Mi Primera App",
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold
        )
        Text(
            text = "Aprendiendo Jetpack Compose",
            fontSize = 16.sp
        )
    }
}
```

📱 **Resultado Esperado:**

Ejercicio 3: Perfil de Usuario con Avatar

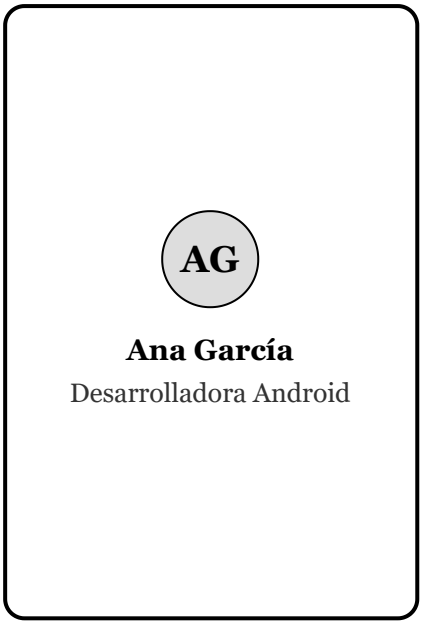
Objetivo: Crear una pantalla de perfil con una imagen circular (avatar), nombre de usuario y tipo de usuario.

📝 Instrucciones:

- Crea una función `PerfilUsuario()`
- Usa **`Image`** con `Modifier.clip(CircleShape)` para hacer la imagen circular
- Añade **`Modifier.size()`** para controlar el tamaño del avatar
- Organiza los elementos en un Column centrado
- Usa diferentes estilos de texto para el nombre y el tipo

```
@Composable
fun PerfilUsuario() {
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Image(
            painter = painterResource(id = R.drawable.avatar),
            contentDescription = "Avatar",
            modifier = Modifier
                .size(80.dp)
                .clip(CircleShape)
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(
            text = "Ana García",
            fontSize = 20.sp,
            fontWeight = FontWeight.Bold
        )
        Text(
            text = "Desarrolladora Android",
            fontSize = 14.sp,
            color = Color.Gray
        )
    }
}
```

📱 **Resultado Esperado:**



💡 **Nota Importante:**

Para usar imágenes reales: Debes añadir tu imagen a la carpeta `res/drawable` de tu proyecto Android Studio. Si eres principiante, te recomendamos empezar con avatares de texto usando **Box** con iniciales.

Ejercicio 4: Botones y Eventos

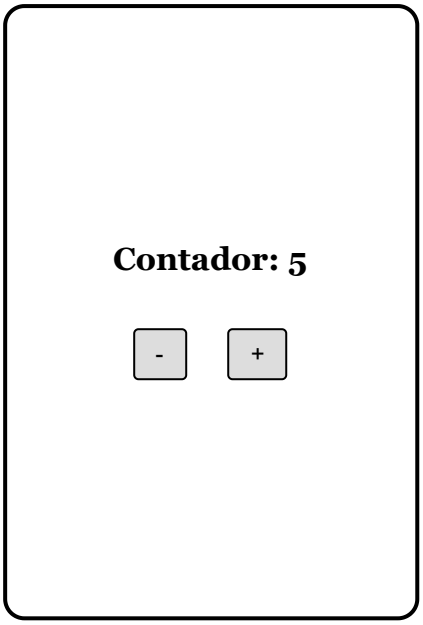
Objetivo: Añadir interactividad con botones que respondan a clicks del usuario.

📝 Instrucciones:

- Crea una pantalla con un contador y dos botones
- Usa **`remember`** y **`mutableStateOf`** para manejar el estado
- Implementa botones para incrementar y decrementar el contador
- Organiza los elementos usando **`Column`** y **`Row`**

```
@Composable
fun ContadorConBotones() {
    var contador by remember { mutableStateOf(0) }

    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = "Contador: $contador",
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold
        )
        Spacer(modifier = Modifier.height(16.dp))
        Row {
            Button(onClick = { contador-- }) {
                Text("-")
            }
            Spacer(modifier = Modifier.width(16.dp))
            Button(onClick = { contador++ }) {
                Text("+")
            }
        }
    }
}
```

Resultado Esperado:



Ejercicio 5: Lista Simple

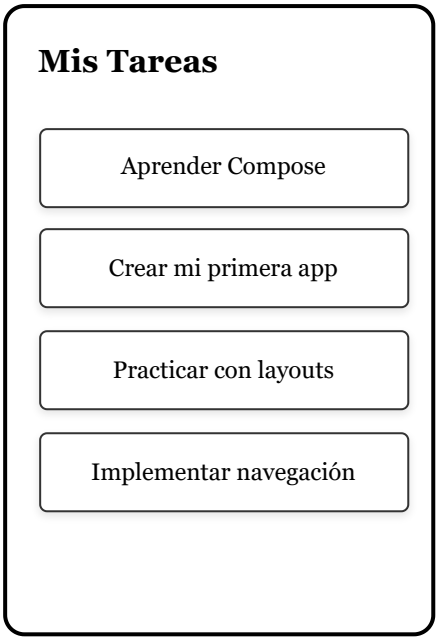
Objetivo: Crear una lista de elementos usando LazyColumn para mostrar una lista de tareas.

📝 Instrucciones:

- Crea una lista de strings con nombres de tareas
- Usa **`LazyColumn`** para mostrar la lista
- Cada elemento debe estar en un **`Card`** con padding
- Añade un título a la pantalla

```
@Composable
fun ListaTareas() {
    val tareas = listOf(
        "Aprender Compose",
        "Crear mi primera app",
        "Practicar con layouts",
        "Implementar navegación",
        "Añadir animaciones"
    )

    Column(
        modifier = Modifier.fillMaxSize()
    ) {
        Text(
            text = "Mis Tareas",
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(16.dp)
        )
        LazyColumn {
            items(tareas) { tarea ->
                Card(
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(8.dp)
                ) {
                    Text(
                        text = tarea,
                        modifier = Modifier.padding(16.dp)
                    )
                }
            }
        }
    }
}
```

📱 Resultado Esperado:



Ejercicio 6: Formulario Básico

> **Objetivo:** Crear un formulario simple con campos de texto y validación básica.

📝 Instrucciones:

- Crea campos **`OutlinedTextField`** para nombre y email
- Usa **`remember`** para manejar el estado de los campos
- Añade un botón para "enviar" el formulario
- Muestra un mensaje cuando se presione el botón

```
@Composable
fun FormularioBasico() {
    var nombre by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var mensaje by remember { mutableStateOf("") }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = "Registro",
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold
        )
        Spacer(modifier = Modifier.height(16.dp))

        OutlinedTextField(
            value = nombre,
            onValueChange = { nombre = it },
            label = { Text("Nombre") },
            modifier = Modifier.fillMaxWidth()
        )

        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Email") },
            modifier = Modifier.fillMaxWidth()
        )

        Spacer(modifier = Modifier.height(16.dp))

        Button(
            onClick = { mensaje = "¡Hola $nombre!" },
            modifier = Modifier.fillMaxWidth()
        ) {
            Text("Enviar")
        }

        if (mensaje.isNotEmpty()) {
            Spacer(modifier = Modifier.height(16.dp))
            Text(
                text = mensaje,
                color = MaterialTheme.colorScheme.primary
            )
        }
    }
}
```

📱 Resultado Esperado:

Ejercicio 7: Navegación Simple

Objetivo: Implementar navegación básica entre dos pantallas usando Navigation Compose.

📝 Instrucciones:

- Configura **`NavController`** y **`NavHost`**
- Crea dos pantallas: "Inicio" y "Perfil"
- Añade botones para navegar entre pantallas
- Usa `navController.navigate()` para cambiar de pantalla

```
@Composable
fun AppNavegacion() {
    val navController = rememberNavController()

    NavHost(
        navController = navController,
        startDestination = "inicio"
    ) {
        composable("inicio") {
            PantallaInicio(navController)
        }
        composable("perfil") {
            PantallaPerfil(navController)
        }
    }
}

@Composable
fun PantallaInicio(navController: NavController) {
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text("Pantalla de Inicio", fontSize = 24.sp)
        Spacer(modifier = Modifier.height(16.dp))
        Button(
            onClick = { navController.navigate("perfil") }
        ) {
            Text("Ir a Perfil")
        }
    }
}
```

📱 Resultado Esperado:

Ejercicio 8: Estados y Efectos

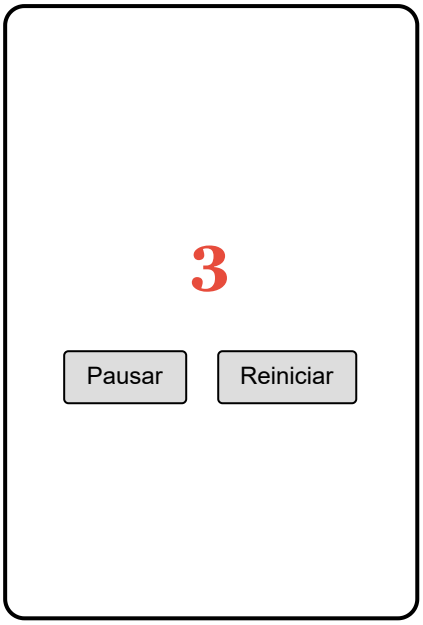
Objetivo: Implementar un temporizador simple usando estados y efectos secundarios.

📝 Instrucciones:

- Crea un temporizador que cuente hacia atrás desde 10 segundos
- Usa **`LaunchedEffect`** para manejar el temporizador
- Añade botones para iniciar, pausar y reiniciar
- Cambia el color del texto cuando el tiempo esté por acabarse

```
@Composable
fun Temporizador() {
    var tiempo by remember { mutableStateOf(10) }
    var activo by remember { mutableStateOf(false) }

    LaunchedEffect(activo) {
        while (activo && tiempo > 0) {
            delay(1000)
            tiempo--
        }
        if (tiempo == 0) {
            activo = false
        }
    }

    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = "$tiempo",
            fontSize = 48.sp,
            fontWeight = FontWeight.Bold,
            color = if (tiempo <= 3) Color.Red else Color.Black
        )

        Spacer(modifier = Modifier.height(32.dp))

        Row {
            Button(onClick = { activo = !activo }) {
                Text(if (activo) "Pausar" else "Iniciar")
            }
            Spacer(modifier = Modifier.width(16.dp))
            Button(onClick = {
                tiempo = 10
                activo = false
            }) {
                Text("Reiniciar")
            }
        }
    }
}
```

Resultado Esperado:



💡 **Consejos Generales para Todos los Ejercicios:**

- **Imports necesarios:** Asegúrate de importar todos los componentes que uses (`androidx.compose.material3.*`, `androidx.compose.ui.*`, etc.)
- **Preview:** Añade `@Preview` a tus funciones composables para ver el resultado en el editor
- **Modifiers:** Experimenta con diferentes modifiers como `padding()`, `fillMaxWidth()`, `background()`
- **Estado:** Recuerda usar `remember` para mantener el estado durante recomposiciones
- **Depuración:** Usa colores de fondo temporales para entender cómo se organizan los elementos
- **Material Design:** Explora los colores y estilos de `MaterialTheme` para conseguir una apariencia profesional

🎉 ¡Felicidades!

Has completado los ejercicios básicos de Jetpack Compose. ¡Ahora estás listo para crear interfaces más complejas!