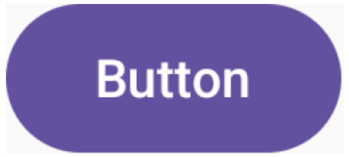
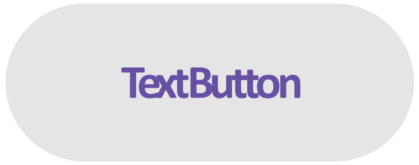
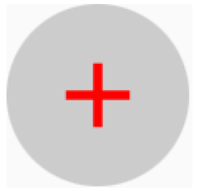
Botones

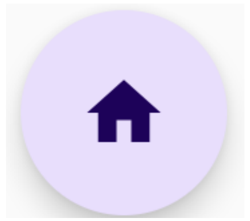
Compose



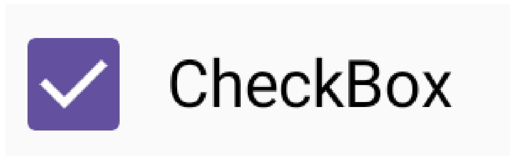


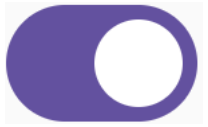


IconButton

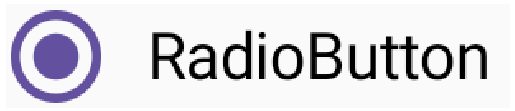


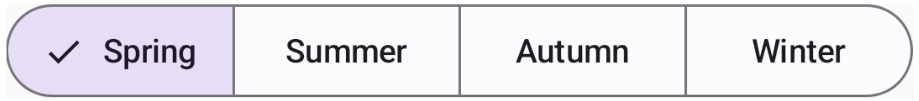
FloatingActionButton





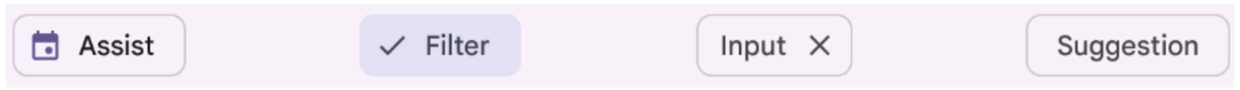
Switch



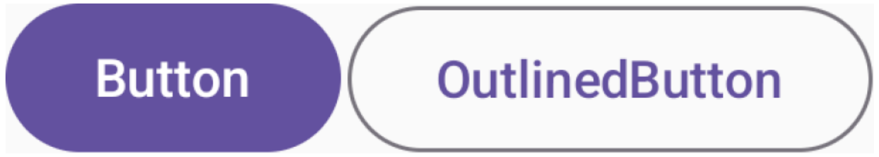


MultiChoiceSegmentedButtonRowScope.SegmentedButton

SingleChoiceSegmentedButtonRowScope.SegmentedButton

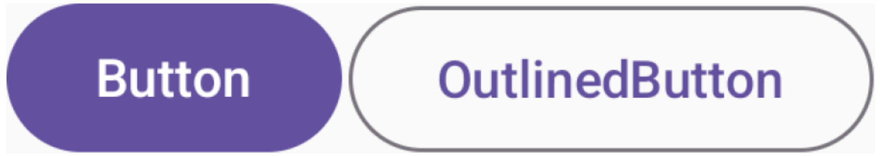


Chips



```
@Composable
fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,

    enabled: Boolean = true,

    shape: Shape = ButtonDefaults.shape,
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),

    content: @Composable RowScope.() -> Unit
)
```

```
@Composable
fun OutlinedButton(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,

    enabled: Boolean = true,

    shape: Shape = ButtonDefaults.outlinedShape,
    border: BorderStroke? = ButtonDefaults.outlinedButtonBorder,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    colors: ButtonColors = ButtonDefaults.outlinedButtonColors(),
    elevation: ButtonElevation? = null,

    content: @Composable RowScope.() -> Unit
)
```



onClick:

() -> Unit → lambda que representa qué hacer si se pulsa el botón.

```
Button(onClick = { … },
    …
  ) {
      Text(text="Púlsame")
   }
```

enabled:

Boolean = true → Indica si el botón está habilitado.

shape:

Shape = ButtonDefaults.shape → Indica la forma del botón.

Shape → Forma del botón

MaterialTheme.shapes.extraSmall
MaterialTheme.shapes.small
MaterialTheme.shapes.medium
MaterialTheme.shapes.large
MaterialTheme.shapes.extraLarge

contentPadding:

PaddingValues = ButtonDefaults.ContentPadding → Espaciado interno.

```
Button(onClick = { … },
     contentPadding = PaddingValues(all = 20.dp),
     …
    ) {
       Text(text="Púlsame")
     }
```

```
Button(onClick = { … },
     contentPadding = PaddingValues(vertical = 10.dp,
                                  horizontal= 20.dp
                                 ),
     …
    ) {
       Text(text="Púlsame")
     }
```

```
Button(onClick = { … },
     contentPadding = PaddingValues(start= 10.dp,
                                  end= 20.dp,
                                  top= 20.dp,
                                  bottom= 20.dp
                                 ),
     …
    ) {
       Text(text="Púlsame")
     }
```

border:

BorderStroke? = null → Borde del botón.

```
Button(onClick = { … },
      border = BorderStroke(width = 2.dp,
                         color = Color.Green),
      …
     ) {
        Text(text="Púlsame")
      }
```

colors:

ButtonColors? = null → colores del botón según sea su estado.

```
ButtonDefaults.buttonColors(containerColor = Color.Green,
                           contentColor = Color.White,
                           disabledContainerColor = Color.Gray,
                           disabledContentColor = Color.Black
                          )
```

elevation:

ButtonElevation? = null → Elevación del botón.

```
ButtonDefaults.elevatedButtonElevation(defaultElevation  = 10.dp,
                                      pressedElevation  = 15.dp,
                                      disabledElevation =  0.dp,
                                      hoveredElevation  = 20.dp,
                                      focusedElevation  = 25.dp
                                     ),
```

```kotlin
@Composable
fun TextButton(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,

    enabled: Boolean = true,

    shape: Shape = ButtonDefaults.textShape,
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.TextButtonContentPadding,
    colors: ButtonColors = ButtonDefaults.textButtonColors(),
    elevation: ButtonElevation? = null,

    content: @Composable RowScope.() -> Unit
)
```

onClick:

() -> Unit → lambda que representa qué hacer si se pulsa el *text button*.

```
TextButton(onClick = { … },
        …
      ) {
         Text(text="Púlsame")
       }
```

enabled:

Boolean = true → Indica si el *text button* está habilitado.

shape:

Shape = ButtonDefaults.shape → Indica la forma del botón.

Shape → Forma del botón

```
MaterialTheme.shapes.extraSmall
MaterialTheme.shapes.small
MaterialTheme.shapes.medium
MaterialTheme.shapes.large
MaterialTheme.shapes.extraLarge
```

TextButton

contentPadding:

PaddingValues = ButtonDefaults.ContentPadding → Espaciado interno.

```
TextButton(onClick = { … },
        contentPadding = PaddingValues(all = 20.dp),
        …
       ) {
          Text(text="Púlsame")
        }
```

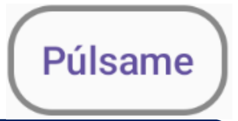
```
TextButton(onClick = { … },
        contentPadding = PaddingValues(vertical = 10.dp,
                                       horizontal= 20.dp
                                      ),
        …
       ) {
          Text(text="Púlsame")
        }
```

```
TextButton(onClick = { … },
        contentPadding = PaddingValues(start= 10.dp,
                                       end= 20.dp,
                                       top= 20.dp,
                                       bottom= 20.dp
                                      ),
        …
       ) {
          Text(text="Púlsame")
        }
```

TextButton

border:

BorderStroke? = null → Borde del botón.



```
TextButton(onClick = { … },
        border = BorderStroke(width = 2.dp,
                    color = Color.Gray),
        …
       ) {
         Text(text="Púlsame")
        }
```

elevation:

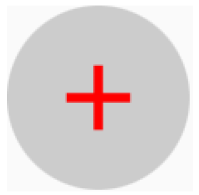
ButtonElevation? = null → Elevación del botón.

```
ButtonDefaults.elevatedButtonElevation(defaultElevation  = 10.dp,
                                        pressedElevation  = 15.dp,
                                        disabledElevation =  0.dp,
                                        hoveredElevation  = 20.dp,
                                        focusedElevation  = 25.dp
                                       ),
```

colors:

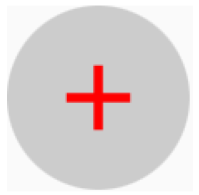
ButtonColors? = null → colores del botón según sea su estado.

```
ButtonDefaults.buttonColors(containerColor = Color.Green,
                            contentColor = Color.White,
                            disabledContainerColor = Color.Gray,
                            disabledContentColor = Color.Black
                           )
```



IconButton

```
@Composable
fun IconButton(
   onClick: () -> Unit,
   modifier: Modifier = Modifier,

   enabled: Boolean = true,

   colors: IconButtonColors = IconButtonDefaults.iconButtonColors(),
   content: @Composable () -> Unit
)
```



IconButton

onClick:

() -> Unit → lambda que representa qué hacer si se pulsa el botón.

```
IconButton(onClick = { … },
        …
       ) {
           Icon(Icons.Filled.Add, contentDescription = "Añadir", tint = Color.Red)
        }
```

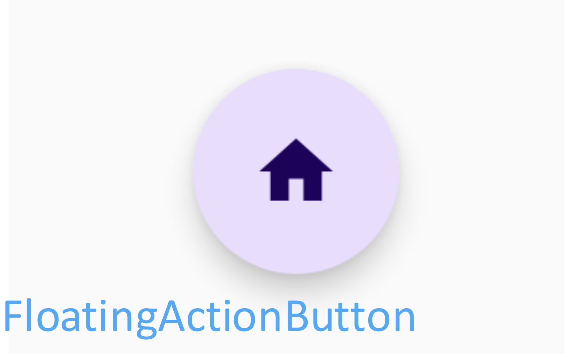
enabled:

Boolean = true → Indica si el botón está habilitado.

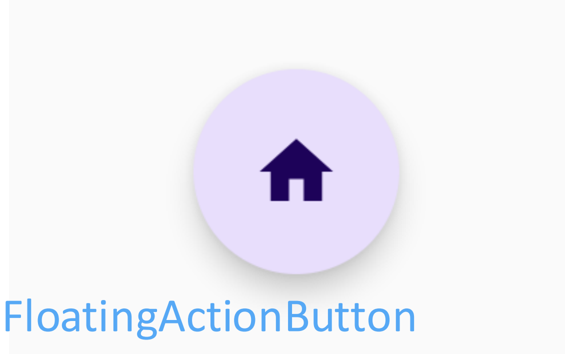
colors:

IconButtonColors? = IconButtonDefaults.iconButtonColors() → colores del botón según sea su estado.

```
IconButtonColors(containerColor = Color.LightGray,
                 contentColor = Color.Red,
                 disabledContainerColor = Color.Gray,
                 disabledContentColor = Color.White
                )
```



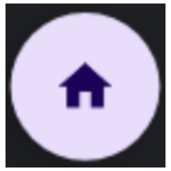
```
@Composable
fun FloatingActionButton(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,

    shape: Shape = FloatingActionButtonDefaults.shape,

    containerColor: Color = FloatingActionButtonDefaults.containerColor,
    contentColor: Color = contentColorFor(containerColor),
    elevation: FloatingActionButtonElevation = FloatingActionButtonDefaults.elevation(),
    content: @Composable () -> Unit,
)
```



onClick:

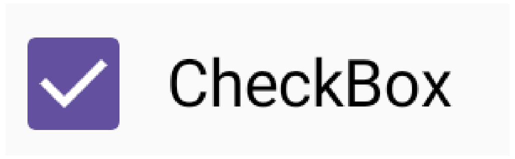
() -> Unit → lambda que representa qué hacer si se pulsa el botón.

```
FloatingActionButton(onClick = {},
                   shape = MaterialTheme.shapes.small.copy(CornerSize(percent = 50)),
                  )
      {
         Icon(Icons.Filled.Home, contentDescription = "Home")
      }
```



enabled:

Boolean = true → Indica si el botón está habilitado.



```
@Composable
fun Checkbox(
    checked: Boolean,
    onCheckedChange: ((Boolean) -> Unit)?,
    modifier: Modifier = Modifier,

    enabled: Boolean = true,
    colors: CheckboxColors = CheckboxDefaults.colors(),
)
```

checked:

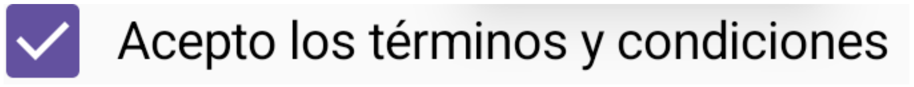
Boolean → indica si está en estado *checked* o no

onCheckedChange:

((Boolean) -> Unit)? → lambda que representa qué hacer si se pulsa el checkbox.

```
val checked by rememberSaveable( mutableStateOf(false) )

Row()
{
 Checkbox(checked= checked,
         onCheckedChange= { checked = it }
        )
 Text(text="Acepto términos y condiciones")
}
```

Sólo es válido al hacer click en el Checkbox



```
val checked by rememberSaveable( mutableStateOf(false) )
val listener : (Boolean) -> Unit = { checked = it }

Row(modifier = Modifier.fillMaxWidth()
                      .toggleable(value = checked,
                                 enabled = true,
                                 role = Role.Checkbox,
                                 onValueChange = { checked = it }
                                ),
    verticalAlignment = Alignment.CenterVertically,
   )
{
  Checkbox(checked = checked,
          onCheckedChange = null
         )
  Text(text = "Acepto los términos y condiciones", Modifier.padding(start = 8.dp))
}
```

Es válido hacer click en todo el Row

enabled:

Boolean = true → Indica si el checkbox está habilitado.

colors:

CheckboxColors? = null → colores del botón según sea su estado.

```
CheckboxDefaults.colors(checkedColor = Color.Green,
                        uncheckedColor = Color.Gray,
                        checkmarkColor = Color.Black,
                        disabledCheckedColor = Color.Gray,
                        disabledUncheckedColor = Color.LightGray,
                        disabledIndeterminateColor = Color.DarkGray)
```

Switch

```
@Composable
fun Switch(
        checked: Boolean,
        onCheckedChange: ((Boolean) -> Unit)?,
        modifier: Modifier = Modifier,

        enabled: Boolean = true,
        colors: SwitchColors = SwitchDefaults.colors(),
)
```

checked:

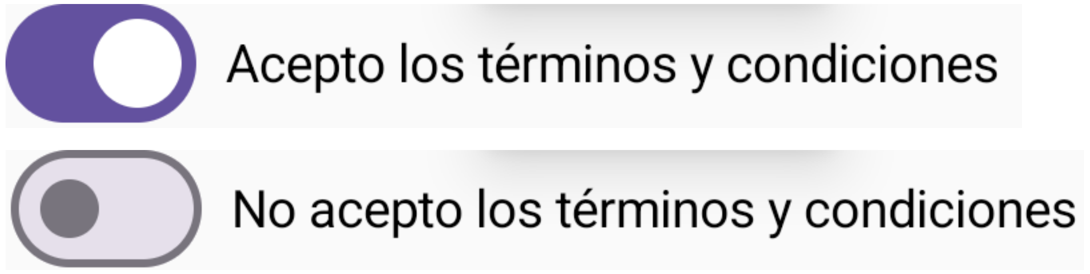
Boolean → indica si está en estado *checked* o no

onCheckedChange:

((Boolean) -> Unit)? → lambda que representa qué hacer si se pulsa el switch.

```
val checked by rememberSaveable( mutableStateOf(false) )

Row()
{
 Switch(checked= checked,
        onCheckedChange= { checked = it }
       )
 Text(text = if (checked) "Acepto los términos y condiciones"
          else "No acepto los términos y condiciones",
      modifier = Modifier.padding(start = 8.dp)
     )
}
```

Sólo es válido al hacer click en el Switch


Acepto los términos y condiciones
No acepto los términos y condiciones

```
val checked by rememberSaveable( mutableStateOf(false) )
val listener : (Boolean) -> Unit = { checked = it }

Row(modifier = Modifier.fillMaxWidth()
                 .toggleable(value = checked,
                             enabled = true,
                             role = Role.Checkbox,
                             onValueChange = { checked = it }
                            ),
    verticalAlignment = Alignment.CenterVertically,
   )
{
  Checkbox(checked = checked,
           onCheckedChange = null
          )
  Text(text = if (checked) "Acepto los términos y condiciones"
            else        "No acepto los términos y condiciones",
       modifier = Modifier.padding(start = 8.dp)
      )
}
```

Es válido hacer click en todo el Row

enabled:

Boolean = true → Indica si el switch está habilitado.

colors:

SwitchColors? = null → colores del botón según sea su estado.

```
SwitchDefaults.colors(checkedThumbColor = Color.www,
                    checkedTrackColor = Color.xxx,
                    checkedBorderColor = Color.yyy,
                    checkedIconColor = Color.zzz,

                    uncheckedThumbColor = Color.www,
                    uncheckedTrackColor = Color.xxx,
                    uncheckedBorderColor = Color.yyy,
                    uncheckedIconColor = Color.zzz,

                    disabledCheckedThumbColor = Color.www,
                    disabledCheckedTrackColor = Color.xxx,
                    disabledCheckedBorderColor = Color.yyy,
                    disabledCheckedIconColor = Color.zzz,

                    disabledUncheckedThumbColor = Color.www,
                    disabledUncheckedTrackColor = Color.xxx,
                    disabledUncheckedBorderColor = Color.yyy,
                    disabledUncheckedIconColor = Color.zzz
                   )
```

```
@Composable
fun RadioButton(selected: Boolean,
            onClick: (() -> Unit)?,
            modifier: Modifier = Modifier,

            enabled: Boolean = true,
            colors: RadioButtonColors = RadioButtonDefaults.colors(),
)
```

RadioButton

selected:

Boolean → indica si el *radio buttom* está seleccionado o no

onClick:

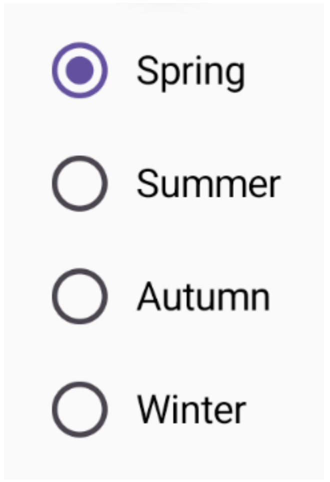
((Boolean) -> Unit)? → lambda que representa qué hacer si se pulsa el *radio buttom*.

```
var currentSeason by rememberSaveable { mutableStateOf(Season.Spring) }

Column(modifier = Modifier.padding(8.dp))
{
   Season.values()
       .forEach { season -> Row(modifier = Modifier.padding(top= 16.dp),
                              verticalAlignment = Alignment.CenterVertically
                            ) {
                                RadioButton(selected = (currentSeason == season),
                                        onClick = { currentSeason = season }
                                      )
                                Text(text = stringArrayResource(R.array.seasons)[season.ordinal],
                                    modifier = Modifier.padding(start = 8.dp)
                                  )
                            }
            }
}
```

Sólo es válido al hacer click en el RadioButton

```
<resources>
   <string-array name="seasons">
      <item>Spring</item>
      <item>Summer</item>
      <item>Autumn</item>
      <item>Winter</item>
   </string-array>
</resources>
```

```
enum class Season
{
   Spring, Summer, Autumn, Winter
}
```



RadioButton

onClick:

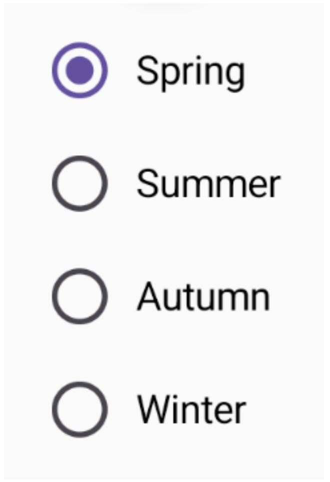
((Boolean) -> Unit)? → lambda que representa qué hacer si se pulsa el *radio buttom*.

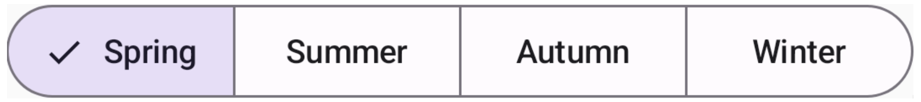
```
var currentSeason by rememberSaveable { mutableStateOf(Season.Spring) }

Column(modifier = Modifier.padding(8.dp))
{
   Season.values()
        .forEach { season -> Row(modifier = Modifier.padding(top= 16.dp)
                                         .selectable(selected = (currentSeason == season),
                                                    role     = Role.RadioButton,
                                                    onClick = { currentSeason = season }
                                                   ),
                                verticalAlignment = Alignment.CenterVertically
                               ) {
                                  RadioButton(selected = (currentSeason == season),
                                              onClick = null
                                             )
                                  Text(text = stringArrayResource(R.array.seasons)[season.ordinal],
                                       modifier = Modifier.padding(start = 8.dp)
                                      )
                               }
                }
}
```

Es válido hacer click en todo el Row

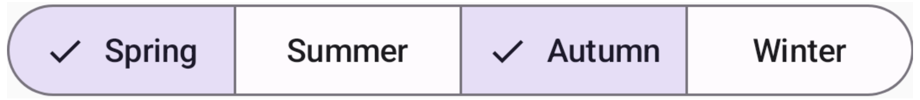
```
<resources>
  <string-array name="seasons">
    <item>Spring</item>
    <item>Summer</item>
    <item>Autumn</item>
    <item>Winter</item>
  </string-array>
</resources>
```

```
enum class Season
{
  Spring, Summer, Autumn, Winter
}
```

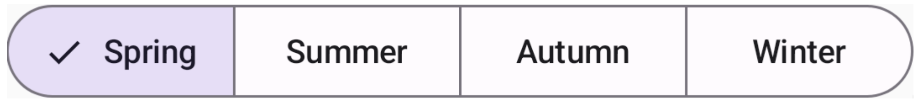




```
@Composable
fun SingleChoiceSegmentedButtonRowScope.SegmentedButton(
   selected: Boolean,
   onClick: () -> Unit,
   shape: Shape,
   modifier: Modifier = Modifier,
   enabled: Boolean = true,
   colors: SegmentedButtonColors = SegmentedButtonDefaults.colors(),
   border: BorderStroke = SegmentedButtonDefaults.borderStroke( colors.borderColor(enabled, selected) ),
   icon: @Composable () -> Unit = { SegmentedButtonDefaults.Icon(selected) },
   label: @Composable () -> Unit,
)
```



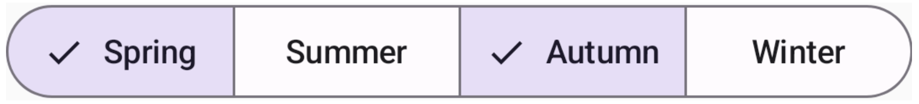
```
@Composable
@ExperimentalMaterial3Api
fun MultiChoiceSegmentedButtonRowScope.SegmentedButton(
   checked: Boolean,
   onCheckedChange: (Boolean) -> Unit,
   shape: Shape,
   modifier: Modifier = Modifier,
   enabled: Boolean = true,
   colors: SegmentedButtonColors = SegmentedButtonDefaults.colors(),
   border: BorderStroke = SegmentedButtonDefaults.borderStroke( colors.borderColor(enabled, checked) ),
icon: @Composable () -> Unit = { SegmentedButtonDefaults.Icon(checked) },
   label: @Composable () -> Unit,
)
```



Selección simple

```
var currentSeason by rememberSaveable { mutableStateOf(Season.Spring) }
val seasonsStringArray = stringArrayResource(R.array.seasons)
val max = Season.values().size


SingleChoiceSegmentedButtonRow()
{
   Season.values().forEach {season ->
                      SegmentedButton(
                        selected = currentSeason =❶eason,
                        onClick = {currentSeason = ❷ason},
                        label = { Text(seasonsString❸ray[season.ordinal]) }
                        shape =  SegmentedButton❹efaults.itemShape(index= season.ordinal, count= max),
                      )
                    }
}
```

1. selected = *Indica si el* SegmentedButton *está o no seleccionado*
2. onClick = *lambda que se ejecutará al hacer click sobre el* SegmentedButton
3. label = *Composable que se mostrará en el* SegmentedButton
4. shape = SegmentedButtonDefaults.itemShape(index= *posición del* SegmentedButton *dentro del* SingleChoiceSegmentedButtonRow,
   count= *cuántos* SegmentedButton *contiene el* SingleChoiceSegmentedButtonRow
   ),



Selección múltiple

```
val selectedSeasons = remember { mutableStateListOf(false, false, false, false) } val seasonsStringArray = stringArrayResource(R.array.seasons)
val max = Season.values().size

MultiChoiceSegmentedButtonRow()
{
   Season.values().forEach { season ->
                     SegmentedButton(
                       checked = selectedSeasons[season.ordinal],
                       onCheckedChange = {isChec[1]d -> selectedSeasons[season.ordinal] = isChecked},
                       label = { Text(seasonsStringA[2]ay[ season.ordinal ]) }
                       shape = SegmentedButtonD[3]ults.itemShape(index = season.ordinal,
                                                    count[4] Season.values().size
                                                   ),
                     )
                  }
}
```

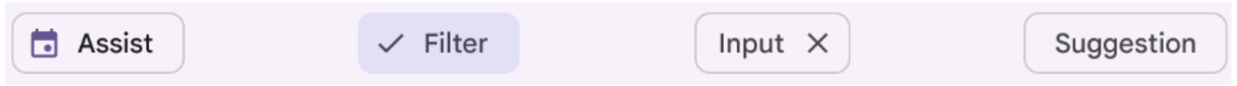
[1] selected = *Indica si el* SegmentedButton *está o no seleccionado*

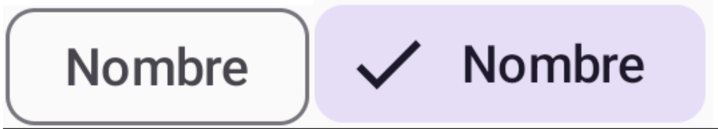
[2] onClick = *lambda que se ejecutará al hacer click sobre el* SegmentedButton

[3] label = *Composable que se mostrará en el* SegmentedButton

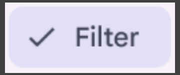
[4] shape = SegmentedButtonDefaults.itemShape(index= *posición del* SegmentedButton *dentro del* MultiChoiceSegmentedButtonRow,
count= *cuántos* SegmentedButton *contiene el* MultiChoiceSegmentedButtonRow
),

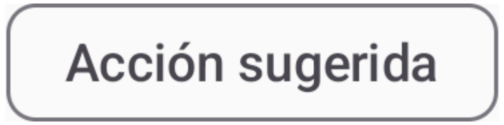
Chips



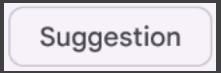


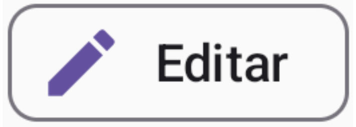
```kotlin
var selectedFieldNombre by remember { mutableStateOf(false) }
FilterChip(selected = selectedFieldNombre,
        onClick = { selectedFieldNombre = !selectedFieldNombre },
        leadingIcon = { if (selectedFieldNombre) Icon(imageVector = Icons.Filled.Check,
                                    contentDescription = "Seleccionado Nombre"
                                   )
                  else null
                 },
        label = { Text("Nombre") }
       )
```



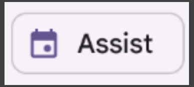


```kotlin
SuggestionChip(onClick = { /* acción  */ },
          label = { Text("Acción sugerida" ) }
         )
```

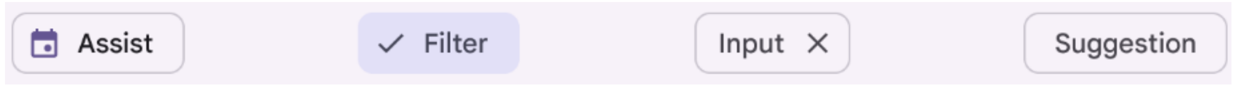


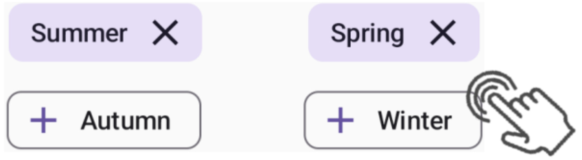


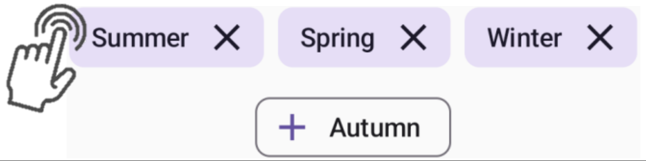
```kotlin
AssistChip(onClick = { /* acción a ejecutar */ },
        leadingIcon = { Icon(imageVector = Icons.Filled.Edit,
                    contentDescription = "Editar"
                   )
                },
        label = { Text("Editar") }
       )
```

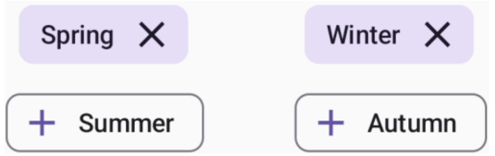


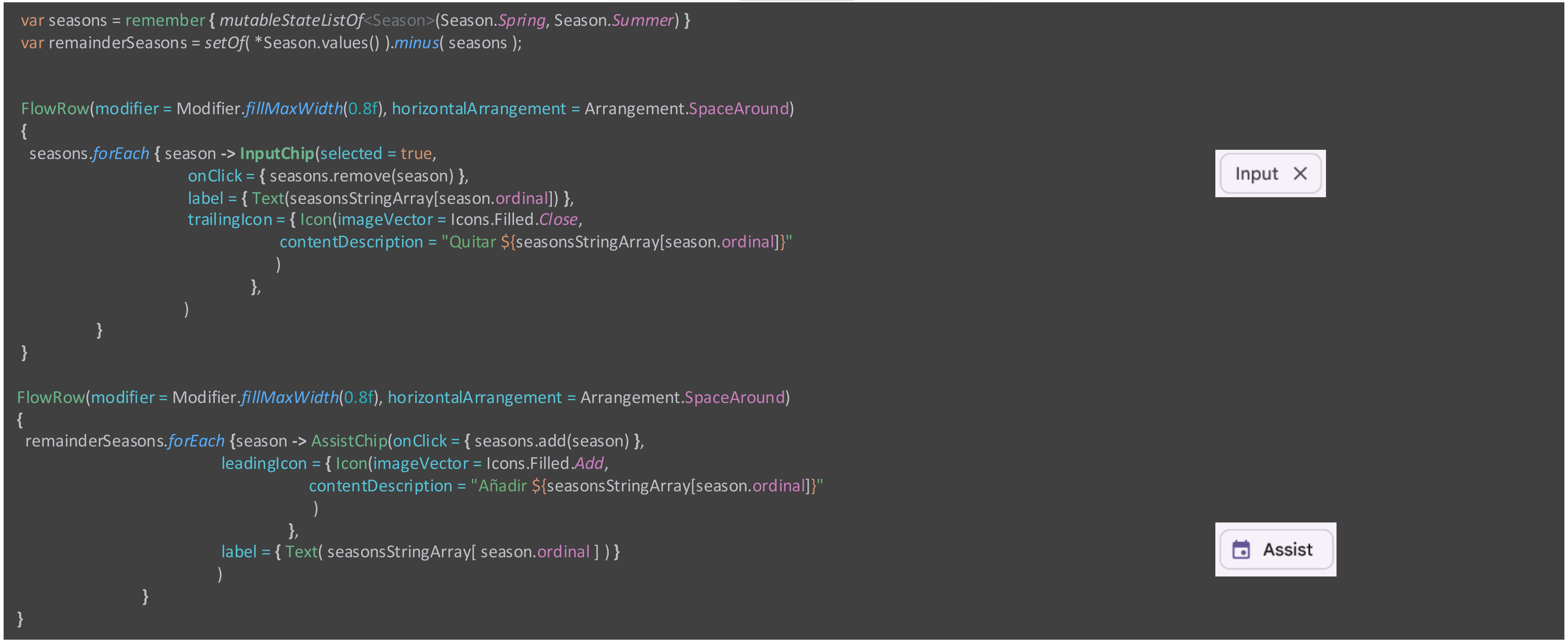
Chips











```kotlin
var seasons = remember { mutableStateListOf<Season>(Season.Spring, Season.Summer) }
var remainderSeasons = setOf( *Season.values() ).minus( seasons );


FlowRow(modifier = Modifier.fillMaxWidth(0.8f), horizontalArrangement = Arrangement.SpaceAround)
{
  seasons.forEach { season -> InputChip(selected = true,
                            onClick = { seasons.remove(season) },
                            label = { Text(seasonsStringArray[season.ordinal]) },
                            trailingIcon = { Icon(imageVector = Icons.Filled.Close,
                                          contentDescription = "Quitar ${seasonsStringArray[season.ordinal]}"
                                          )
                                      },
                            )
              }
}

FlowRow(modifier = Modifier.fillMaxWidth(0.8f), horizontalArrangement = Arrangement.SpaceAround)
{
  remainderSeasons.forEach {season -> AssistChip(onClick = { seasons.add(season) },
                                  leadingIcon = { Icon(imageVector = Icons.Filled.Add,
                                                contentDescription = "Añadir ${seasonsStringArray[season.ordinal]}"
                                                )
                                            },
                                  label = { Text( seasonsStringArray[ season.ordinal ] ) }
                                  )
                    }
}
```