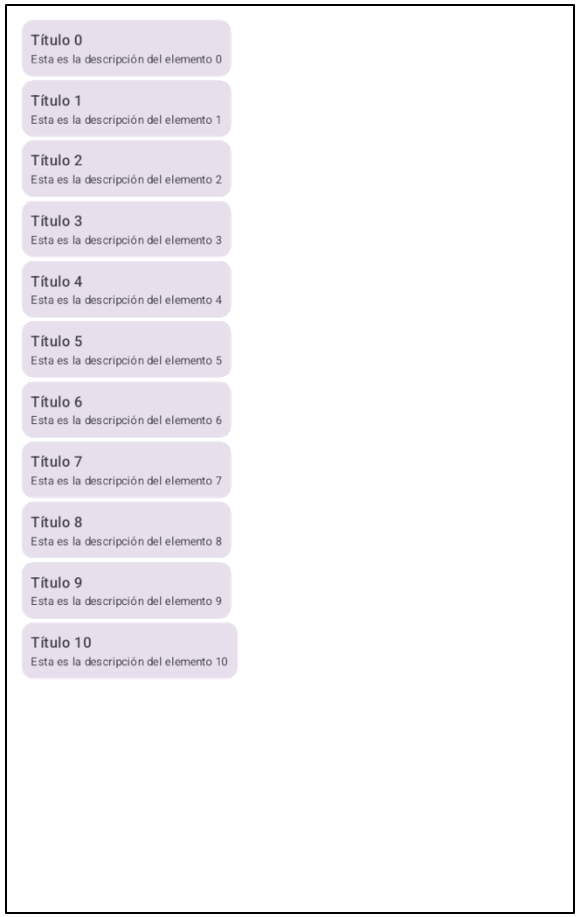
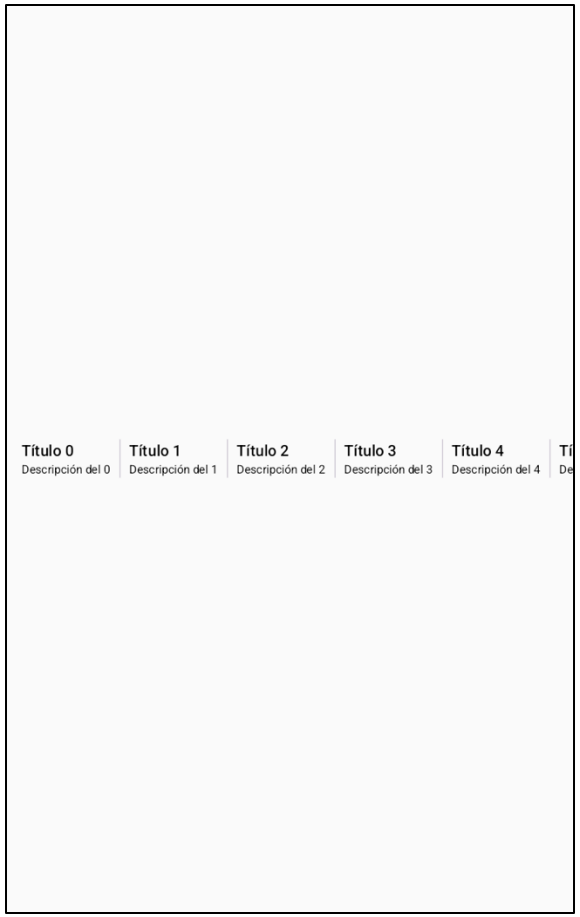
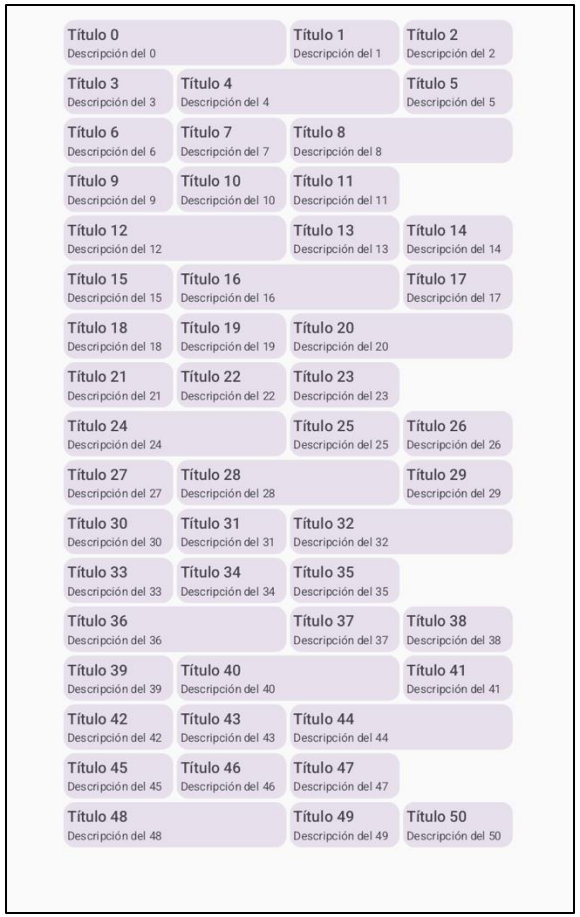
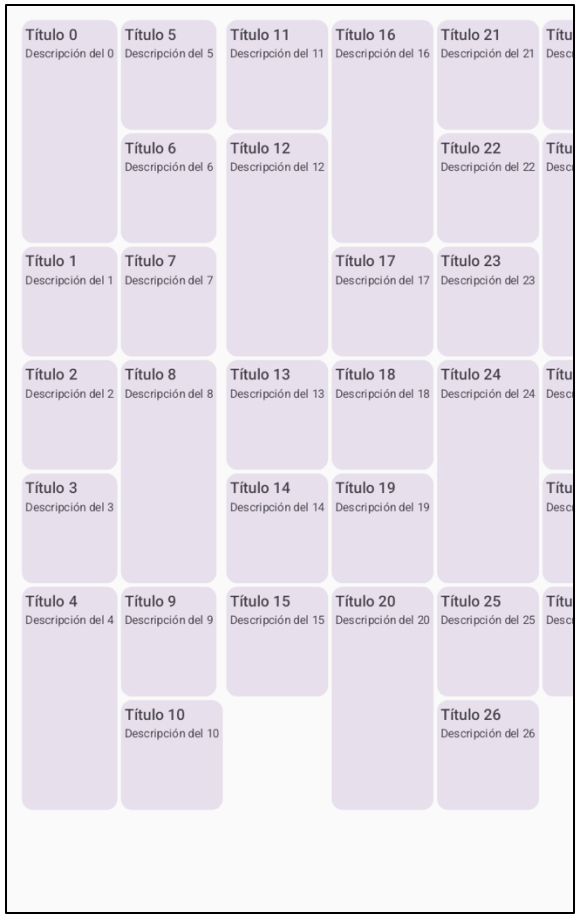
LazyColumn, LazyRow
LazyVerticalGrid, LazyHorizontalGrid









LazyColumn

```
@Composable
fun LazyColumn(
    modifier: Modifier = Modifier,
    state: LazyListState = rememberLazyListState(),
    contentPadding: PaddingValues = PaddingValues(0.dp),
    reverseLayout: Boolean = false,
    verticalArrangement: Arrangement.Vertical = if (!reverseLayout) Arrangement.Top else Arrangement.Bottom,
    horizontalAlignment: Alignment.Horizontal = Alignment.Start,
    flingBehavior: FlingBehavior = ScrollableDefaults.flingBehavior(),
    userScrollEnabled: Boolean = true,
    content: LazyListScope.() -> Unit
)
```

LazyColumn

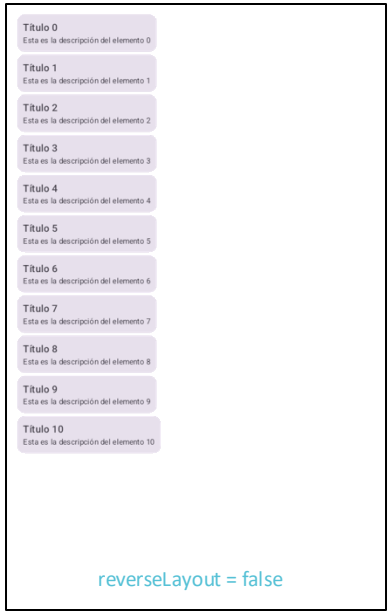
contentPadding:

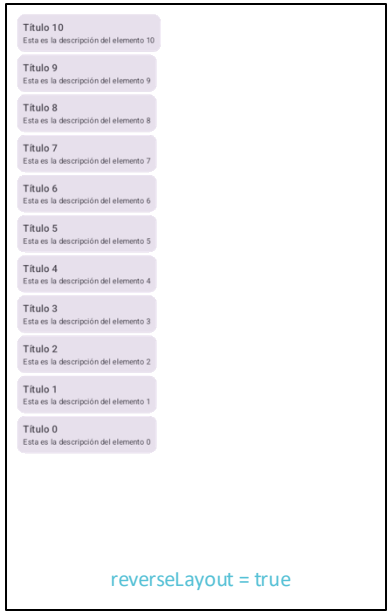
PaddingValues = PaddingValues(0.dp), → Agrega un padding antes del primer elemento o después del último.

```
val allContentPadding  = PaddingValues(all = 16.dp)
val hvContentPadding   = PaddingValues(horizontal = 16.dp, vertical = 16.dp)
val stebContentPadding = PaddingValues(start = 16.dp, top = 16.dp, end = 16.dp, bottom = 16.dp)

LazyColumn (contentPadding = hvContentPadding,
        ) {
           : : :
         }
```

reverseLayout

Boolean = false → Indica si visualmente se invertirá el orden en el que aparecerán los elementos.





LazyColumn

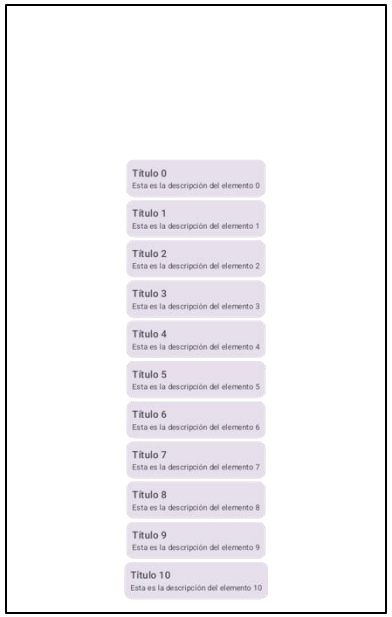
verticalArrangement:

Arrangement.Vertical → Indica cómo alinear verticalmente el conjunto de elementos dentro de las dimensiones del LazyColumn.

horizontalAlignment: :

Alignment.Horizontal → Indica cómo se alinean horizontalmente los items mostrados en el LazyColumn.

```
LazyColumn (modifier = Modifier.fillMaxWidth()
                        .fillMaxHeight(),
          verticalArrangement = Arrangement.Bottom,
          horizontalAlignment = Alignment. CenterHorizontally,
          )
{
   : : :
}
```



LazyColumn

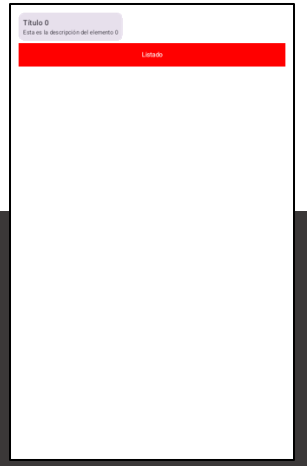
content:

LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyColumn.

```
LazyListScope
{
    fun item(key: Any? = null,
             contentType: Any? = null,
             content: @Composable LazyItemScope.() -> Unit
            )
}
```

Añade un item al LazyColumn

content	es el composable que describe el item.
key	representa el identificador del item dentro del LazyColumn. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados).
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo




```
LazyColumn(modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp),
       )
{
  item {
        ShowItem( Item(0) )
      }
  item {
        Box(modifier = Modifier.fillMaxWidth()
                        .height(50.dp)
                        .background(Color.Red, RectangleShape),
            contentAlignment = Alignment.Center
           ) {
             Text("Listado", color = Color.White, )
           }
}

@Composable
fun ShowItem(item: Item)
{
 Card(modifier = Modifier.padding(bottom = 4.dp))
 {
  Column(modifier = Modifier.padding(10.dp))
  {
   Text(text= item.title, style= MaterialTheme.typography.titleMedium)
   Text(text= item.description, style= MaterialTheme.typography.bodySmall)
  }
 }
}
```

```
class Item(val value: Int)
{
 val title: String = "Título $value"
 val description: String = "Esta es la descripción del elemento $value"
}
```

LazyColumn

content:

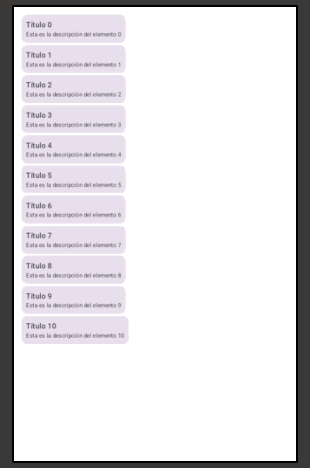
LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyColumn.

```
fun <T> LazyListScope.items(items: List<T>,
                key: ((item: T) -> Any)? = null,
                contentType: (item: T) -> Any? = { null },
                itemContent: @Composable LazyItemScope.(item: T) -> Unit
)
}
```

Añade una lista de items al LazyColumn

items	Representa una lista conteniendo los datos a mostrar.
itemContent	El composable en forma de lambda que recibe como parámetro un elemento de la lista y describe cómo debe visualizarse
key	representa el identificador del item dentro del LazyColumn. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados). Se representa como una lambda que recibe como parámetro el elemento.
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo. Se representa como una lambda que recibe como parámetro el elemento.

```
val items = (0..10).map { index -> Item(index) }

LazyColumn(modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp),
       )
{
 items(items = items)
 {
  item -> ShowItem(item)
 }
}

@Composable
fun ShowItem(item: Item)
{
 Card(modifier = Modifier.padding(bottom = 4.dp))
 {
  Column(modifier = Modifier.padding(10.dp))
  {
   Text(text= item.title, style= MaterialTheme.typography.titleMedium)
   Text(text= item.description, style= MaterialTheme.typography.bodySmall)
  }
 }
}

class Item(val value: Int)
{
 val title: String = "Título $value"
 val description: String = "Esta es la descripción del elemento $value"
}
```



LazyColumn

content:

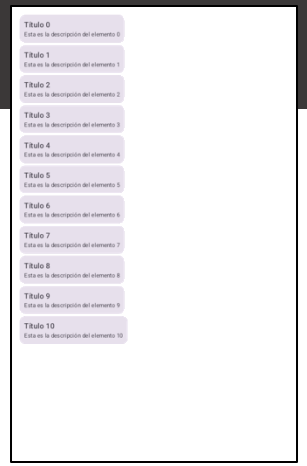
LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyColumn.

```
fun <T> LazyListScope.itemsIndexed(items: List<T>,
                                   key: ((index: Int, item: T) -> Any)? = null,
                                   contentType: (index: Int, item: T)-> Any? = {null},
                                   itemContent: @Composable LazyItemScope.(index: Int, item: T) -> Unit
                                  )
```

Añade una lista de items al LazyColumn

items	Representa una lista conteniendo los datos a mostrar.
itemContent	El composable en forma de lambda que recibe como parámetro el índex del elemento y el elemento de la lista
key	representa el identificador del item dentro del LazyColumn. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados) Se representa como una lambda que recibe como parámetros el index del elemento y el elemento.
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo, Se representa como una lambda que recibe como parámetros el index del elemento y el elemento

```
val items = (0..10).map { index -> Item(index) }

LazyColumn(modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp),
       )
{
 itemsIndexed(items = items)
 {
  index, item -> ShowItem(item)
 }
}
```



LazyColumn

content:

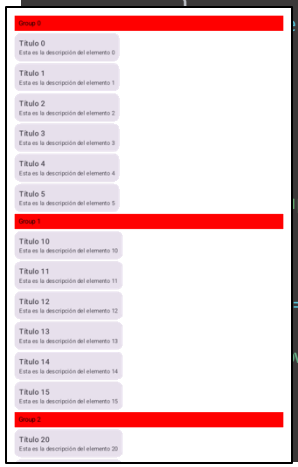
LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyColumn.

```
fun <T> LazyListScope.stickyHeader(key: Any? = null,
                                   contentType: Any? = null,
                                   content: @Composable LazyItemScope.() -> Unit
                                  )
```

Define un encabezado que se "pega" a la parte superior de la lista mientras se hace scroll. Esto significa que, al desplazarte, el elemento declarado con stickyHeader permanece visible en la parte superior hasta que otro stickyHeader lo reemplaza.

content	El composable que describe el encabezado
key	representa el identificador del item dentro del LazyColumn. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados)
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo,

```
val mapItems = (0..5).flatMap {group-> (0..5).map {
                                    index-> Item(group, index+group*10)
                                   }
                              }
                    .groupBy { item -> item.group }

LazyColumn(modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp),
       )
{
   mapItems.forEach {entry-> stickyHeader()
                             {
                                                    r = Modifier.padding(bottom = 4.dp)
                                                    .fillMaxSize()
                                                    .clip(RectangleShape)
                                                    .background(Color.Red)
                                                    .padding(8.dp)


                                                    p ${entry.key}")



                                                    = entry.value)

                                                    wItem(item)
```



LazyColumn

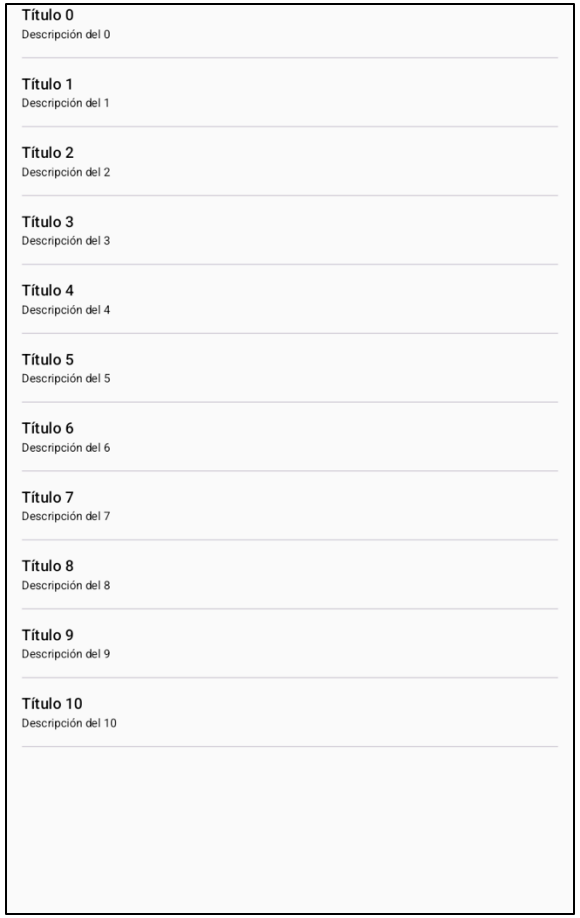
```
fun HorizontalDivider(modifier: Modifier = Modifier,
                      thickness: Dp = DividerDefaults.Thickness,
                      color: Color = DividerDefaults.color,
                     )
```

Línea horizontal que actúa como delimitador entre los items de un LazyColumn

modifier	Por ejemplo Modifier.*padding*(vertical = 16.*dp*)
thickness	representa el grosor de la línea
color	Color de la línea.

```
val items = (0..10).map { Item(value = it) }

LazyColumn(modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(horizontal = 16.dp),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.Start
       )
{
  items(items = items)
  {
    item -> ShowItem(item)
  }
}
```

```
@Composable
fun ShowItem(item: Item)
{
  Column(modifier = Modifier.height(IntrinsicSize.Min))
  {
    Text(text = item.title, style = MaterialTheme.typography.titleMedium)
    Text(text = item.description, style = MaterialTheme.typography.bodySmall)
    HorizontalDivider(modifier = Modifier.padding(vertical = 16.dp),
                      thickness = 1.dp,
                     )
  }
}
```



LazyRow

```
@Composable
fun LazyRow(
    modifier: Modifier = Modifier,
    state: LazyListState = rememberLazyListState(),
    contentPadding: PaddingValues = PaddingValues(0.dp),
    reverseLayout: Boolean = false,
    horizontalArrangement: Arrangement.Horizontal = if (!reverseLayout) Arrangement.Start else Arrangement.End,
    verticalAlignment: Alignment.Vertical = Alignment.Top,
    flingBehavior: FlingBehavior = ScrollableDefaults.flingBehavior(),
    userScrollEnabled: Boolean = true,
    content: LazyListScope.() -> Unit
)
```

LazyRow

contentPadding:

PaddingValues = PaddingValues(0.dp), → Agrega un padding antes del primer elemento o después del último.

```
val allContentPadding  = PaddingValues(all = 16.dp)
val hvContentPadding   = PaddingValues(horizontal = 16.dp, vertical = 16.dp)
val stebContentPadding = PaddingValues(start = 16.dp, top = 16.dp, end = 16.dp, bottom = 16.dp)

LazyRow (contentPadding = hvContentPadding,
    ) {
        : : :
     }
```

reverseLayout

Boolean = false → Indica si visualmente se invertirá el orden en el que aparecerán los elementos.

LazyRow

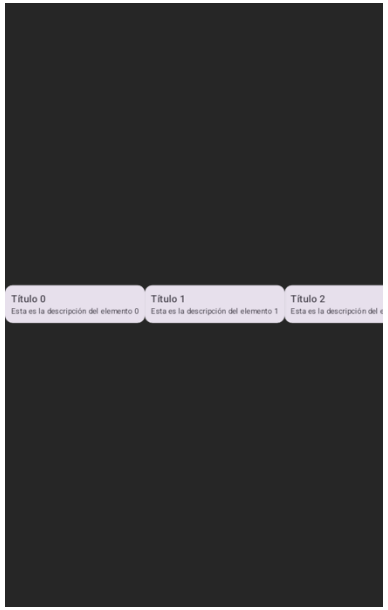
horizontalArrangement:

Arrangement.Horizontal → Indica cómo alinear horizontalmente el conjunto de elementos dentro de las dimensiones del LazyRow.

verticalAlignment: :

Alignment.Vertical → Indica cómo se alinean verticalmente los items mostrados en el LazyRow.

```
LazyRow(modifier = Modifier.fillMaxSize(),
      horizontalArrangement = Arrangement.Start,
      verticalAlignment = Alignment.CenterVertically
     )
{
  : : :
}
```



LazyRow

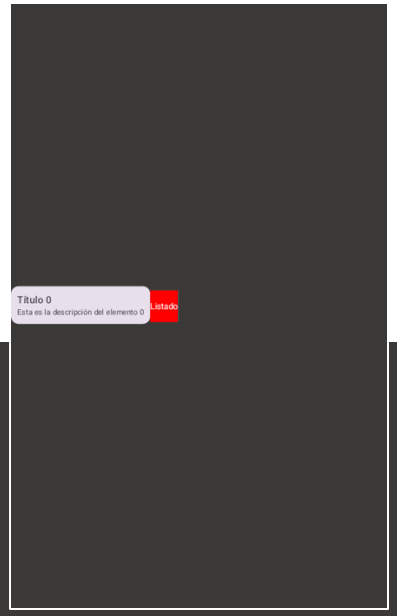
content:

LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyRow.

```
LazyListScope
{
    fun item(key: Any? = null,
             contentType: Any? = null,
             content: @Composable LazyItemScope.() -> Unit
            )
}
```

Añade un item al LazyRow

content	es el composable que describe el item.
key	representa el identificador del item dentro del LazyRow. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados).
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo




```
LazyRow(modifier = Modifier.fillMaxSize(),
      contentPadding = PaddingValues(16.dp),
     )
{
  item {
       ShowItem( Item(0) )
     }
  item {
       Box(modifier = Modifier.fillMaxWidth()
                        .height(50.dp)
                        .background(Color.Red, RectangleShape),
           contentAlignment = Alignment.Center
          ) {
            Text("Listado", color = Color.White, )
          }
}

@Composable
fun ShowItem(item: Item)
{
 Card(modifier = Modifier.padding(bottom = 4.dp))
 {
  Column(modifier = Modifier.padding(10.dp))
  {
   Text(text= item.title, style= MaterialTheme.typography.titleMedium)
   Text(text= item.description, style= MaterialTheme.typography.bodySmall)
  }
 }
}
```

```
class Item(val value: Int)
{
 val title: String = "Título $value"
 val description: String = "Esta es la descripción del elemento $value"
}
```

LazyRow

content:

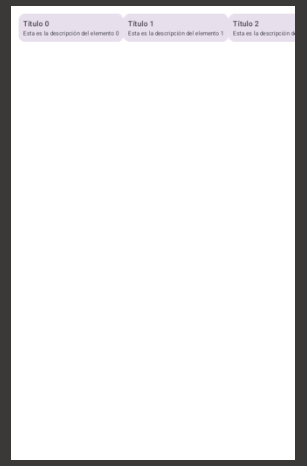
LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyRow.

```
fun <T> LazyListScope.items(items: List<T>,
                            key: ((item: T) -> Any)? = null,
                            contentType: (item: T) -> Any? = { null },
                            itemContent: @Composable LazyItemScope.(item: T) -> Unit
                           )
}
```

Añade una lista de items al LazyRow

items	Representa una lista conteniendo los datos a mostrar.
itemContent	El composable en forma de lambda que recibe como parámetro un elemento de la lista y describe cómo debe visualizarse
key	representa el identificador del item dentro del LazyRow. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados). Se representa como una lambda que recibe como parámetro el elemento.
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo. Se representa como una lambda que recibe como parámetro el elemento.

```
val items = (0..10).map { index -> Item(index) }

LazyRow(modifier = Modifier.fillMaxSize(),
      contentPadding = PaddingValues(16.dp),
     )
{
 items(items = items)
 {
  item -> ShowItem(item)
 }
}

@Composable
fun ShowItem(item: Item)
{
 Card(modifier = Modifier.padding(bottom = 4.dp))
 {
  Column(modifier = Modifier.padding(10.dp))
  {
   Text(text= item.title, style= MaterialTheme.typography.titleMedium)
   Text(text= item.description, style= MaterialTheme.typography.bodySmall)
  }
 }
}

class Item(val value: Int)
{
 val title: String = "Título $value"
 val description: String = "Esta es la descripción del elemento $value"
}
```



LazyRow

content:

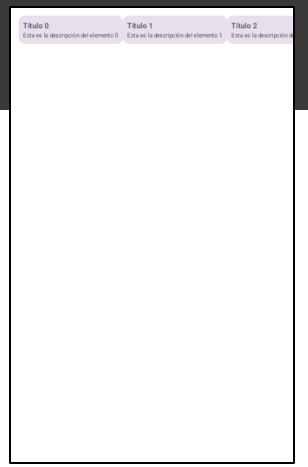
LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyRow.

```
fun <T> LazyListScope.itemsIndexed(items: List<T>,
                                   key: ((index: Int, item: T) -> Any)? = null,
                                   contentType: (index: Int, item: T)-> Any? = {null},
                                   itemContent: @Composable LazyItemScope.(index: Int, item: T) -> Unit
                                  )
```

Añade una lista de items al LazyColumn

items	Representa una lista conteniendo los datos a mostrar.
itemContent	El composable en forma de lambda que recibe como parámetro el índex del elemento y el elemento de la lista
key	representa el identificador del item dentro del LazyRow. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados) Se representa como una lambda que recibe como parámetros el index del elemento y el elemento.
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo, Se representa como una lambda que recibe como parámetros el index del elemento y el elemento

```
val items = (0..10).map { index -> Item(index) }

LazyRow(modifier = Modifier.fillMaxSize(),
      contentPadding = PaddingValues(16.dp),
     )
{
  itemsIndexed(items = items)
  {
    index, item -> ShowItem(item)
  }
}
```



LazyRow

content:

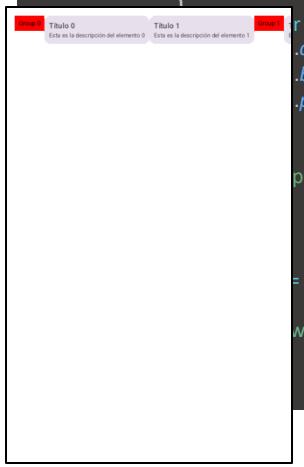
LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyRow.

```
fun <T> LazyListScope.stickyHeader(key: Any? = null,
                                   contentType: Any? = null,
                                   content: @Composable LazyItemScope.() -> Unit
                                  )
```

Define un encabezado que se "pega" a la parte superior de la lista mientras se hace scroll. Esto significa que, al desplazarte, el elemento declarado con stickyHeader permanece visible en la parte superior hasta que otro stickyHeader lo reemplaza.

content	El composable que describe el encabezado
key	representa el identificador del item dentro del LazyRow. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados)
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo,

```
val mapItems = (0..5).flatMap {group-> (0..1).map {
                                  index-> Item(group, index+group*10)
                                 }
                              }
                .groupBy { item -> item.group }

LazyRow(modifier = Modifier.fillMaxSize(),
     contentPadding = PaddingValues(16.dp),
    )
{
   mapItems.forEach {entry-> stickyHeader()
                             {
                                r = Modifier.padding(bottom = 4.dp)
                                .clip(RectangleShape)
                                .background(Color.Red)
                                .padding(8.dp)


                                p ${entry.key}")



                                = entry.value)

                                wItem(item)
```



LazyRow

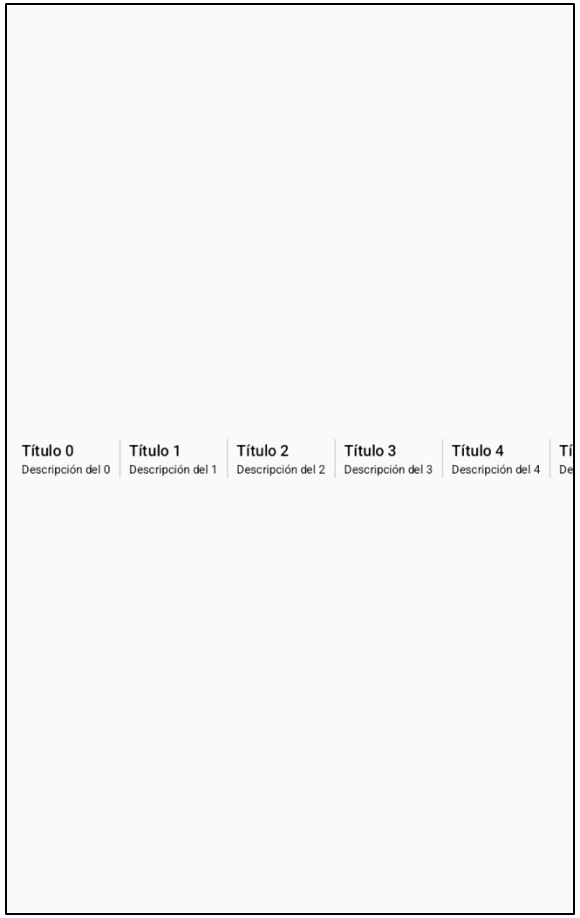
```
fun VerticalDivider(modifier: Modifier = Modifier,
                    thickness: Dp = DividerDefaults.Thickness,
                    color: Color = DividerDefaults.color,
                   )
```

Línea vertical que actúa como delimitador entre los items de un LazyRow.

modifier	Por ejemplo Modifier.*padding*(horizontal = 16.*dp*)
thickness	representa el grosor de la línea
color	Color de la línea.

```
val items = (0..10).map { Item(value = it) }

LazyRow(modifier = Modifier.fillMaxSize(),
      contentPadding = PaddingValues(horizontal = 16.dp),
      horizontalArrangement = Arrangement.Start,
      verticalAlignment = Alignment.CenterVertically
     )
{
 items(items = items)
 {
  item -> ShowItem(item)
 }
}
```

```
@Composable
fun ShowItem(item: Item)
{
 Row(modifier = Modifier.height(IntrinsicSize.Min))
 {
  Column()
  {
   Text(text = item.title, style = MaterialTheme.typography.titleMedium)
   Text(text = item.description, style = MaterialTheme.typography.bodySmall)
  }
  VerticalDivider(modifier = Modifier.padding(start = 10.dp, end = 10.dp),
                  thickness = 1.dp,
                 )
  }
}
```



LazyVerticalGrid

```
@Composable
fun LazyVerticalGrid (
    columns: GridCells,
    modifier: Modifier = Modifier,
    contentPadding: PaddingValues = PaddingValues(0.dp),
    reverseLayout: Boolean = false,
    verticalArrangement: Arrangement.Vertical = if (!reverseLayout) Arrangement.Top else Arrangement.Bottom,
    horizontalArrangement: Arrangement.Horizontal = Arrangement.Start,
    userScrollEnabled: Boolean = true,
    content: LazyGridScope.() -> Unit
)
```

LazyVerticalGrid

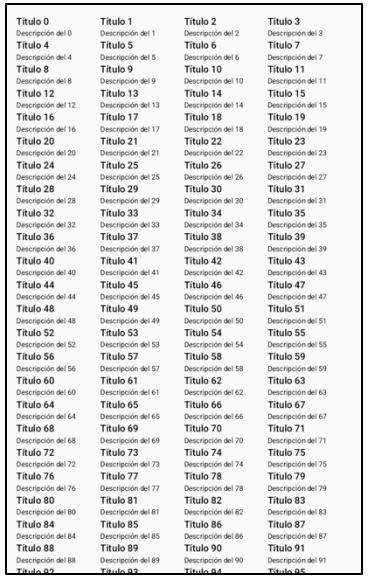
contentPadding:

PaddingValues = PaddingValues(0.dp), → Agrega un padding antes del primer elemento o después del último.

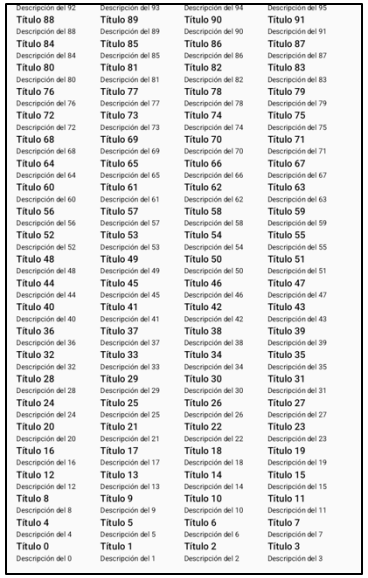
```
val allContentPadding  = PaddingValues(all = 16.dp)
val hvContentPadding   = PaddingValues(horizontal = 16.dp, vertical = 16.dp)
val stebContentPadding = PaddingValues(start = 16.dp, top = 16.dp, end = 16.dp, bottom = 16.dp)

LazyVerticalGrid (contentPadding = hvContentPadding)
{
  : : :
}
```

reverseLayout

Boolean = false → Indica si visualmente se invertirá el orden en el que aparecerán los elementos.



reverseLayout = false



reverseLayout = true

LazyVerticalGrid

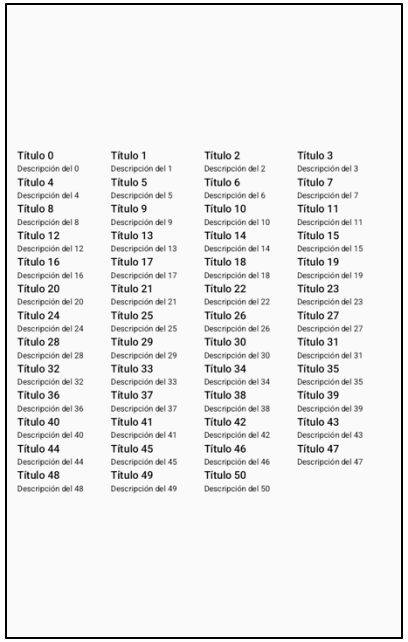
horizontalArrangement:

Arrangement.Horizontal → Indica cómo alinear horizontalmente el conjunto de elementos dentro de las dimensiones del LazyVerticalGrid.

verticalAlignment: :

Alignment.Vertical → Indica cómo se alinean verticalmente los items mostrados en el LazyVerticalGrid.

```
LazyVerticalGrid(modifier = Modifier.fillMaxSize(),
            horizontalArrangement = Arrangement.Start,
            verticalAlignment = Alignment.CenterVertically
           )
{
  : : :
}
```

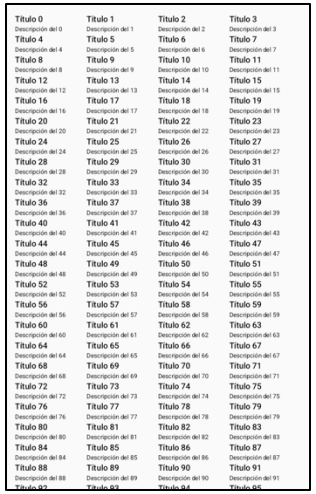


LazyVerticalGrid

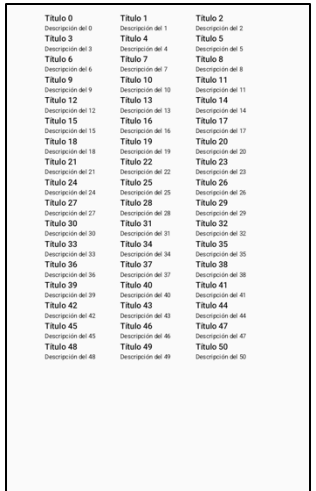
columns:

GridCells → Indica cuántas columnas se mostrarán.

```
columns = GridCells.Fixed(4)

Establece que las columnas deben ser 4 .

LazyVerticalGrid(modifier = Modifier.fillMaxSize(),
        columns = GridCells.Fixed(4)
        )
{ : : : }
```



```
columns = GridCells.FixedSize(150.dp)

Establece el número de columnas en función de su tamaño.

LazyVerticalGrid(modifier= Modifier.fillMaxSize(),
        columns= GridCells.FixedSize(150.dp),
        horizontalArrangement= Arrangement.Center,
        verticalArrangement= Arrangement.Top,
        )
{ : : : }
```



```
columns = GridCells.Adaptative(150.dp)

Establece el número de columnas en función de su tamaño mínimo.

LazyVerticalGrid(modifier= Modifier.fillMaxSize(),
        columns= GridCells.Adaptative(150.dp),
        horizontalArrangement= Arrangement.Center,
        verticalArrangement= Arrangement.Top,
        )
{ : : : }
```

LazyVerticalGrid

content:

LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyGrid.

```
LazyGridScope
{
    fun item(key: Any? = null,
        span: (LazyGridItemSpanScope.(item: T) -> GridItemSpan)? = null,
        contentType: Any? = null,
        content: @Composable LazyItemScope.() -> Unit
        )
}
```

Añade un item al LazyRow

content	es el composable que describe el item.
key	representa el identificador del item dentro del LazyGrid. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados).
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo,
span	Define cuántas columnas ocupará cada elemento a mostrar. Por defecto vale 1 (un elemento ocupa una columna).. Se representa como una lambda que recibe como parámetro el elemento.

LazyVerticalGrid

content:

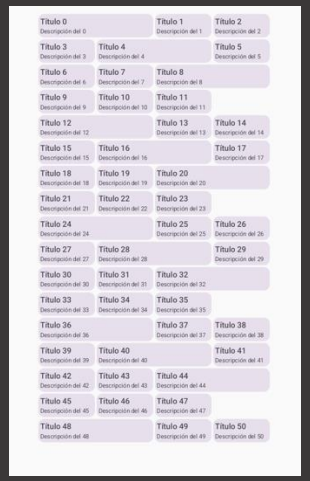
LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyGrid.

```
fun <T> LazyGridScope.items(items: List<T>,
                            key: ((item: T) -> Any)? = null,
                            span: (LazyGridItemSpanScope.(item: T) -> GridItemSpan)? = null,
                            contentType: (item: T) -> Any? = { null },
                            itemContent: @Composable LazyGridItemScope.(item: T) -> Unit
                           )
}
```

Añade una lista de items al LazyGrid

items	Representa una lista conteniendo los datos a mostrar.
itemContent	El composable en forma de lambda que recibe un elemento de la lista y describe cómo se visualiza.
key	representa el identificador del item dentro del LazyGrid. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados). Se representa como una lambda que recibe como parámetro el elemento.
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo. Se representa como una lambda que recibe como parámetro el elemento.
span	Define cuántas columnas ocupará cada elemento a mostrar. Por defecto vale 1 (un elemento ocupa una columna).. Se representa como una lambda que recibe como parámetro el elemento.

```
val items = (0..50).map { Item(value = it) }
Box( contentAlignment = Alignment.Center )
{
 LazyVerticalGrid(modifier = Modifier.fillMaxSize(),
            columns = GridCells.FixedSize(120.dp),
            contentPadding = PaddingValues(all = 16.dp),
            horizontalArrangement = Arrangement.Center,
            verticalArrangement = Arrangement.Top,
           )
 {
  items(items= items,
      span= {item-> GridItemSpan(if (item.value % 4 == 0) 2
                          else 1
                         )
          }
     )
  {
   item -> ShowItem(item)
  }
 }
}

@Composable
fun ShowItem(item: Item)
{
 Card(modifier = Modifier.padding(bottom=4.dp,
                      end = 4.dp))
 {
  Column(modifier = Modifier.padding(4.dp))
  {
   Text(text = item.title,
      style = MaterialTheme.typography.titleMedium)
   Text(text = item.description,
      style = MaterialTheme.typography.bodySmall)
  }
 }
}
```



LazyVerticalGrid

content:

LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyGrid.

```
fun <T> LazyGridScope.itemsIndexed(items: List<T>,
                    key: ((index: Int, item: T) -> Any)? = null,
                    contentType: (index: Int, item: T)-> Any? = {null},
                    itemContent: @Composable LazyItemScope.(index: Int, item: T) -> Unit
                   )
```

Añade una lista de items al LazyColumn

items	Representa una lista conteniendo los datos a mostrar.
itemContent	El composable en forma de lambda que recibe como parámetro el índex del elemento y el elemento de la lista
key	representa el identificador del item dentro del LazyGrid. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados) Se representa como una lambda que recibe como parámetros el index del elemento y el elemento.
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo, Se representa como una lambda que recibe como parámetros el index del elemento y el elemento.
span	Define cuántas columnas ocupará cada elemento a mostrar. Por defecto vale 1 (un elemento ocupa una columna).. Se representa como una lambda que recibe como parámetro el elemento.

LazyHorizontalGrid

```
@Composable
fun LazyHorizontalGrid (
    rows: GridCells,
    modifier: Modifier = Modifier,
    contentPadding: PaddingValues = PaddingValues(0.dp),
    reverseLayout: Boolean = false,
    horizontalArrangement: Arrangement.Horizontal = if(!reverseLayout) Arrangement.Start else Arrangement.End,
    verticalArrangement: Arrangement.Vertical = Arrangement.Top,
    userScrollEnabled: Boolean = true,
    content: LazyGridScope.() -> Unit
)
```

LazyHorizontalGrid

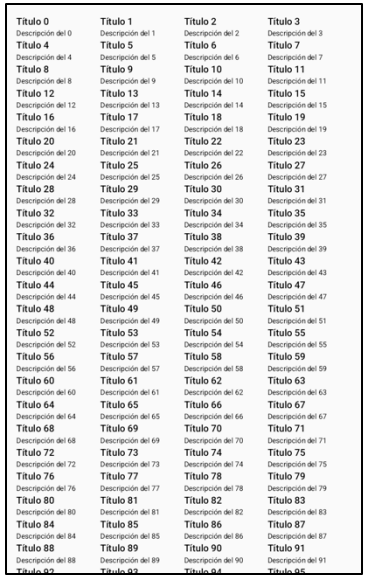
contentPadding:

PaddingValues = PaddingValues(0.dp), → Agrega un padding antes del primer elemento o después del último.

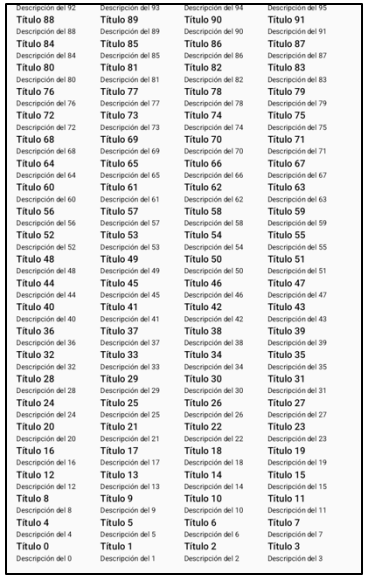
```
val allContentPadding  = PaddingValues(all = 16.dp)
val hvContentPadding   = PaddingValues(horizontal = 16.dp, vertical = 16.dp)
val stebContentPadding = PaddingValues(start = 16.dp, top = 16.dp, end = 16.dp, bottom = 16.dp)

LazyHorizontalGrid (contentPadding = hvContentPadding)
{
  : : :
}
```

reverseLayout

Boolean = false → Indica si visualmente se invertirá el orden en el que aparecerán los elementos.



reverseLayout = false



reverseLayout = true

LazyHorizontalGrid

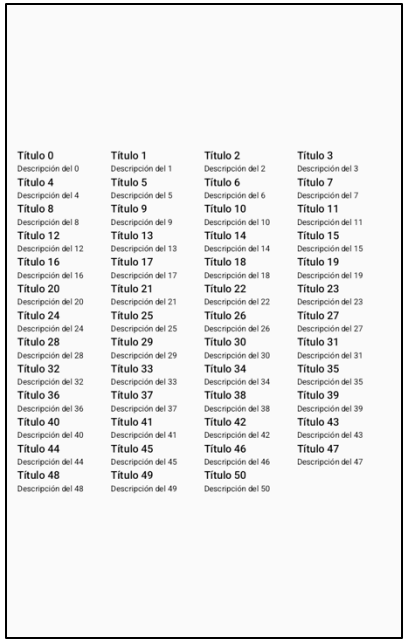
horizontalArrangement:

Arrangement.Horizontal → Indica cómo alinear horizontalmente el conjunto de elementos dentro de las dimensiones del LazyHorizontalGrid.

verticalAlignment:

Alignment.Vertical → Indica cómo se alinean verticalmente los items mostrados en el LazyHorizontalGrid.

```
LazyHorizontalGrid(modifier = Modifier.fillMaxSize(),
                   horizontalArrangement = Arrangement.Start,
                   verticalAlignment = Alignment.CenterVertically
                  )
{
  : : :
}
```

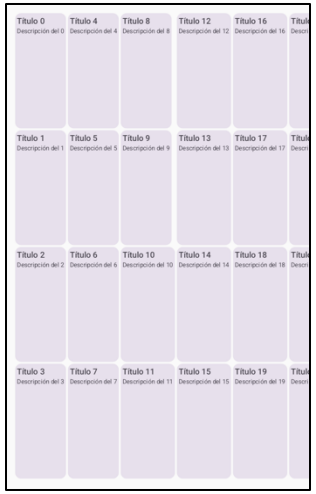


LazyHorizontalGrid

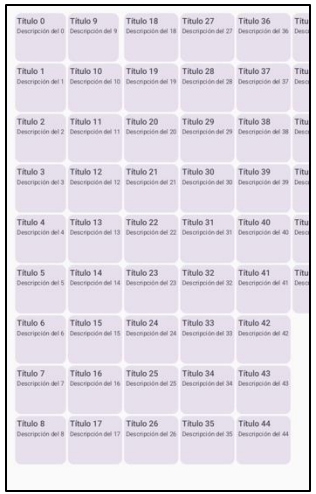
rows:

GridCells → Indica cuántas filas se mostrarán.

```
columns = GridCells.Fixed(4)

Establece que las columnas deben ser 4 .

LazyHorizontalGrid(modifier = Modifier.fillMaxSize(),
            columns = GridCells.Fixed(4)
          )
{ : : : }
```



```
columns = GridCells.FixedSize(100.dp)

Establece el número de columnas en función de su tamaño.

LazyHorizontalGrid(modifier= Modifier.fillMaxSize(),
            columns= GridCells.FixedSize(150.dp),
            horizontalArrangement= Arrangement.Center,
            verticalArrangement= Arrangement.Top,
          )
{ : : : }
```



```
columns = GridCells.Adaptative(150.dp)

Establece el número de columnas en función de su tamaño mínimo.

LazyVerticalGrid(modifier= Modifier.fillMaxSize(),
            columns= GridCells.Adaptative(150.dp),
            horizontalArrangement= Arrangement.Center,
            verticalArrangement= Arrangement.Top,
          )
{ : : : }
```

LazyHorizontalGrid

content:

LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyGrid.

```
LazyGridScope
{
    fun item(key: Any? = null,
        span: (LazyGridItemSpanScope.(item: T) -> GridItemSpan)? = null,
        contentType: Any? = null,
        content: @Composable LazyItemScope.() -> Unit
       )
}
```

Añade un item al LazyRow

content	es el composable que describe el item.
key	representa el identificador del item dentro del LazyGrid. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados).
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo,
span	Define cuántas columnas ocupará cada elemento a mostrar. Por defecto vale 1 (un elemento ocupa una columna).. Se representa como una lambda que recibe como parámetro el elemento.

LazyHorizontalGrid

content:

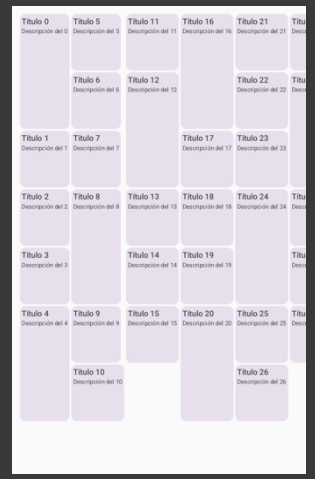
LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyGrid.

```
fun <T> LazyGridScope.items(items: List<T>,
                            key: ((item: T) -> Any)? = null,
                            span: (LazyGridItemSpanScope.(item: T) -> GridItemSpan)? = null,
                            contentType: (item: T) -> Any? = { null },
                            itemContent: @Composable LazyGridItemScope.(item: T) -> Unit
                           )
}
```

Añade una lista de items al LazyGrid

items	Representa una lista conteniendo los datos a mostrar.
itemContent	El composable en forma de lambda que recibe un elemento de la lista y describe cómo se visualiza.
key	representa el identificador del item dentro del LazyGrid. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados). Se representa como una lambda que recibe como parámetro el elemento.
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo. Se representa como una lambda que recibe como parámetro el elemento.
span	Define cuántas filas ocupará cada elemento a mostrar. Por defecto vale 1 (un elemento ocupa una fila). Se representa como una lambda que recibe como parámetro el elemento.

```
val items = (0..50).map { Item(value = it) }
Box( contentAlignment = Alignment.Center )
{
 LazyHorizontalGrid(modifier = Modifier.fillMaxSize(),
            rows = GridCells.FixedSize(120.dp),
            contentPadding = PaddingValues(all = 16.dp),
            horizontalArrangement = Arrangement.Center,
            verticalArrangement = Arrangement.Top,
           )
 {
  items(items= items,
      span= {item-> GridItemSpan(if (item.value % 4 == 0) 2
                    else 1
                   )
            }
     )
  {
   item -> ShowItem(item)
  }
 }
}

@Composable
fun ShowItem(item: Item)
{
 Card(modifier = Modifier.padding(bottom=4.dp,
                  end = 4.dp))
 {
  Column(modifier = Modifier.padding(4.dp))
  {
   Text(text = item.title,
     style = MaterialTheme.typography.titleMedium)
   Text(text = item.description,
     style = MaterialTheme.typography.bodySmall)
  }
 }
}
```



LazyHorizontalGrid

content:

LazyListScope.() -> Unit → Indica el contenido que se mostrará en el LazyGrid.

```
fun <T> LazyGridScope.itemsIndexed(items: List<T>,
                    key: ((index: Int, item: T) -> Any)? = null,
                    contentType: (index: Int, item: T)-> Any? = {null},
                    itemContent: @Composable LazyItemScope.(index: Int, item: T) -> Unit
                   )
```

Añade una lista de items al LazyColumn

items	Representa una lista conteniendo los datos a mostrar.
itemContent	El composable en forma de lambda que recibe como parámetro el índex del elemento y el elemento de la lista
key	representa el identificador del item dentro del LazyGrid. Si no se especifica, su valor será el *index* del item (posición que ocupa el item dentro de la lista). La key puede ser estable o no estable. Se considera estable si su valor se mantiene inmutable entre recomposiciones (por ejemplo la propiedad id de un entity). Se considera no estable si su valor procede de un valor mutable o del index del item. Los estados creados con *remember* o *rememberSaveable* dentro de content se asocian a la key del item. Si el key es estable y único, Compose podrá reutilizar el estado guardado entre recomposiciones o cambios en la lista. Si el key es no estable se recrearán los estados internos (es decir, se reinicializan los valores recordados) Se representa como una lambda que recibe como parámetros el index del elemento y el elemento.
contentType	su valor representa el tipo del content. Las composiciones de items del mismo tipo podrían reutilizarse más eficientemente. null es un valor válido. indica que todos los items son del mismo tipo, Se representa como una lambda que recibe como parámetros el index del elemento y el elemento.
span	Define cuántas columnas ocupará cada elemento a mostrar. Por defecto vale 1 (un elemento ocupa una columna).. Se representa como una lambda que recibe como parámetro el elemento.