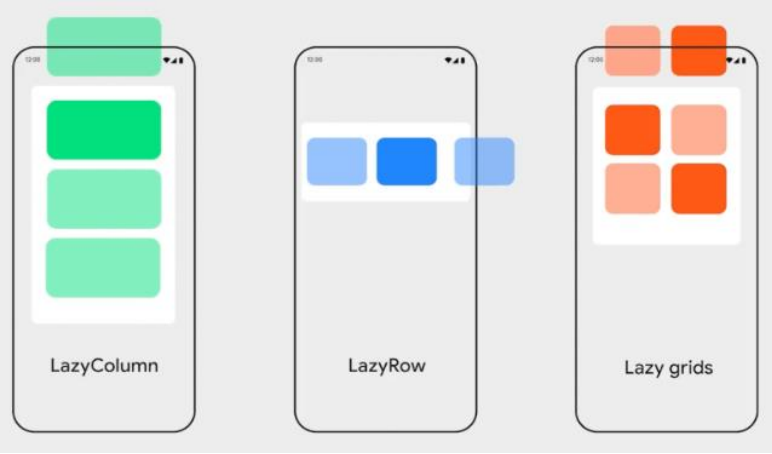
Lazy Layout

https://developer.android.com/jetpack/compose/lists

Lazy Composable

- 리스트나 그리드와 같은 대량의 아이템을 효율적으로 관리하기 위한 컴포저블
- **모든 요소를 동시에 렌더링 하는 대신에 화면에 표시되는 요소만 렌더링하여 앱의 성능을 유지**
  - 기존 RecyclerView과는 다르게 재사용 안됨
  - 표시 영역을 벗어난 아이템들은 제거되고, 보여지는 시점에 만들어짐
- Lazy Composable의 종류
  - Lazy Lists
    - LazyColumn / LazyRow
  - Lazy Grid
    - LazyVerticalGrid / LazyHorizontalGrid
  - Lazy Staggered Grid
    - LazyVertialStaggeredGrid / LazyHorizontalStaggeredGrid



LazyColumn / LazyRow

- LazyColumn/Row 컴포저블
  - 세로/가로 스크롤 리스트

```
@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
) {
```

```
@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
) {
```

LazyColumn / LazyRow

- **LazyListScope 내에서 아이템 기술**
  - **item() : 개별 아이템 추가**
  - **items() : 여러 아이템 한번에 추가**
  - **itemsIndexed() : 아이템의 콘텐츠와 인덱스값을 함께 얻음**
- **리스트에 저장할 리스트 데이터의 상태**
  - **mutableStateListOf()** 이용

```
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.material.Text

val itemsList = (0..5).toList()
val itemsIndexedList = listOf("A", "B", "C")

LazyColumn {
    items(itemsList) {
        Text("Item is $it")
    }

    item {
        Text("Single item")
    }

    itemsIndexed(itemsIndexedList) { index, item ->
        Text("Item at index $index is $item")
    }
}
```

LazyColumn 예

```
LazyColumn {
    // Add a single item
    item {
        Text(text = "First item")
    }

    // Add 5 items
    items(5) { index ->
        Text(text = "Item: $index")
    }

    // Add another single item
    item {
        Text(text = "Last item")
    }
}
```

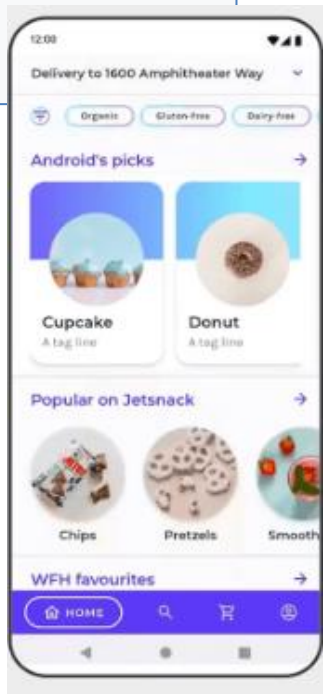
```
/**
 * import androidx.compose.foundation.lazy.items
 */
LazyColumn {
    items(messages) { message ->
        MessageRow(message)
    }
}
```

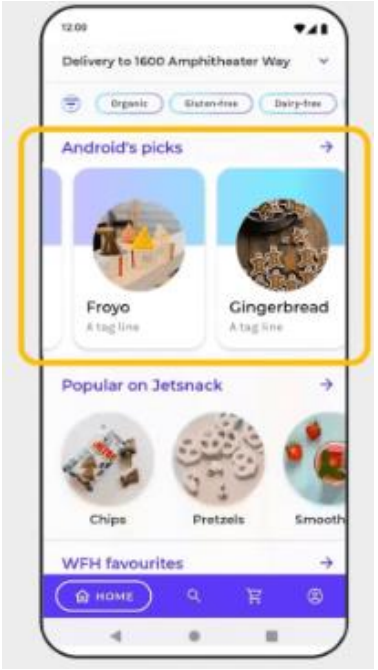
```
LazyColumn {
    itemsIndexed(messages) { index, message ->
        Text(text = "$index = $message")
    }
}
```

Content padding / spacing

- Content Padding (컨텐츠가 잘릴때 유용)

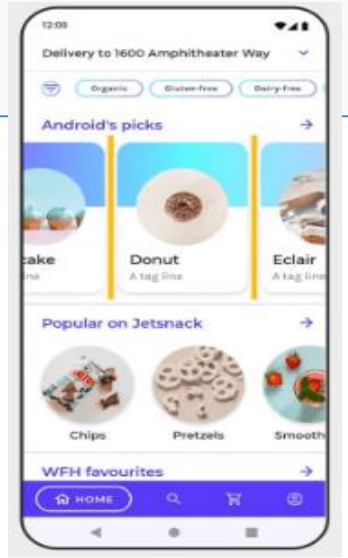
```
LazyColumn(
    contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),
) {
    // ...
}
```





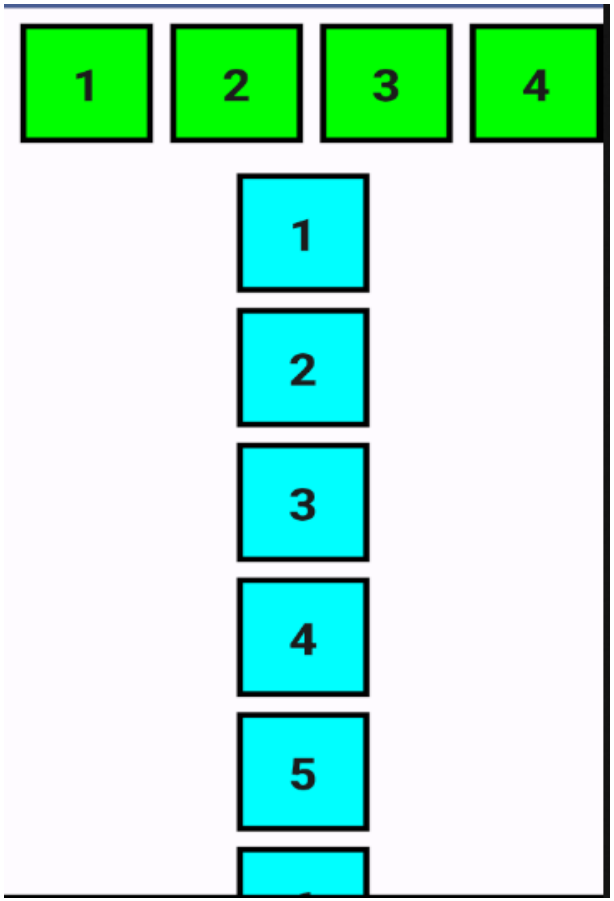
- Content Spacing

```
LazyColumn(
    verticalArrangement = Arrangement.spacedBy(4.dp),
) {
    // ...
}
```



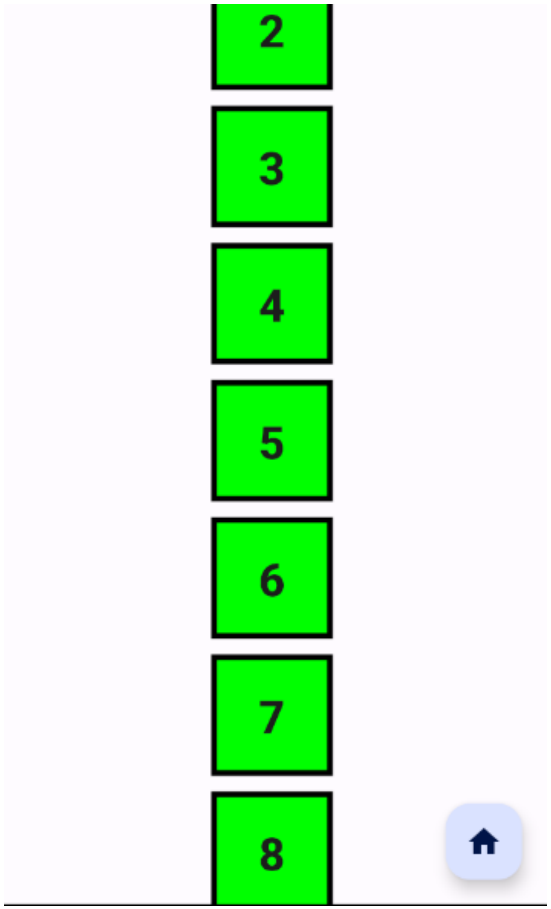
기본 사용법 실습 1

- LazyRow 와 LazyColumn 사용하기



기본 사용법 실습 2

- AnimatedVisibility 이용한 위치 이동하기



스크롤 위치

- LazyListState 객체
  - 스크롤의 위치를 저장하고 리스트의 정보를 포함하는 객체
  - 상태를 기억할 때는 rememberLazyListState() 사용
  - 위치 이동 함수
    - state.scrollToItem(0)
    - scrollToItem/animateScrollToItem
      - Suspend 함수
      - coroutineScope 내에서 실행

```
val showButton by remember {
    derivedStateOf {
        state.firstVisibleItemIndex > 0
    }
}
```

```
@Composable
fun MessageList(messages: List<Message>) {
    val listState = rememberLazyListState()
    // Remember a CoroutineScope to be able to launch
    val coroutineScope = rememberCoroutineScope()

    LazyColumn(state = listState) {
        // ...
    }

    ScrollToTopButton(
        onClick = {
            coroutineScope.launch {
                // Animate scroll to the first item
                listState.animateScrollToItem(index = 0)
            }
        }
    )
}
```

Coroutine

- 비동기적으로 실행되는 코드를 간소화하기 위해 사용
  - 서로 협력해서 실행을 주고 받으면서 동작하는 여러 서브루틴
  - **https://developer.android.com/kotlin/coroutines**
- Coroutine은 Thread가 아니라 서브루틴
  - Thread에서 실행되는 서브루틴으로, 하나의 Thread에 여러 개의 coroutine이 존재할 수 있음 (멀티태스킹 가능)
  - Thread를 Block 시키지 않고 다른 작업을 처리할 수 있어 동시성 프로그램이 가능
    - 여러 서브루틴을 번갈아 실행하는 비동기적인 프로그래밍 기법
    - 멀티 Thread를 운영할 때 발생하는 Context Switching의 부담을 줄임
      - 하나의 Thread로 여러 coroutine을 운영할 수 있음
  - UI 스레드(메인 스레드)에서 실행 할 수 없는 작업들은 코루틴을 이용
    - 네트워크 관련 작업
    - 데이터베이스 관련 잡업
    - 시간이 많이 소요되는 IO 관련 작업

코루틴 스코프(Coroutine Scope)

- 모든 코루틴은 명시적인 스코프 안에서 실행되어야 함
  - 코루틴을 취소 및 정리 할 때 누수가 발생하지 않음을 보장
    - 코루틴 스코프에 있는 코루틴은 일괄 취소
- 종류
  - GlobalScope : 애플리케이션 라이프사이클 전체와 관련된 코루틴 (권장하지 않음)
  - **rememberCoroutineScope()**
    - **컴포저블안에서 코루틴 실행하기 위해 사용**
    - 컴포저블이 스크린에 있는 동안 지속되는 스코프
  - viewLifeCycleOwner.lifecycleScope
    - Activity/Fragment 가 유지 되는 동안 지속되는 스코프
  - viewModelScope
    - ViewModel 인스턴스 안에서 코루틴을 실행하기 위해 사용하는 스코프
    - ViewModel 인스턴스가 파기되는 시점에 자동 취소

코루틴 스코프(Coroutine Scope)

- 코루틴 스코프 객체 생성

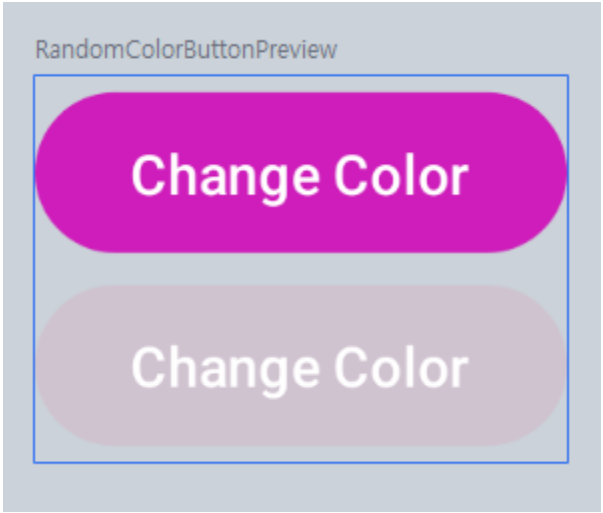
```
val scope = rememberCoroutineScope()

scope.launch{

}

val job = scope.launch{

}

job.cancel()
```

Suspend 함수

- 일시 중단 함수 (suspend function)
  - 코루틴에서 사용하는 특수한 유형의 코틀린 함수
  - Suspend 키워드 사용하여 선언
    - 일시 정 지 및 재시작 될 수 있는 함수
    - UI 스레드를 Block 시키지 않고 오랜 시간 작업할 수 있는 함수

예제. 확인하기

- 코루틴 동작 방식 이해하기



AnimatedVisibility & FAB

- AnimatedVisibility
  - 값에 따라 컨텐츠의 표시 및 사라짐에 애니메이션 적용하는 컴포저블

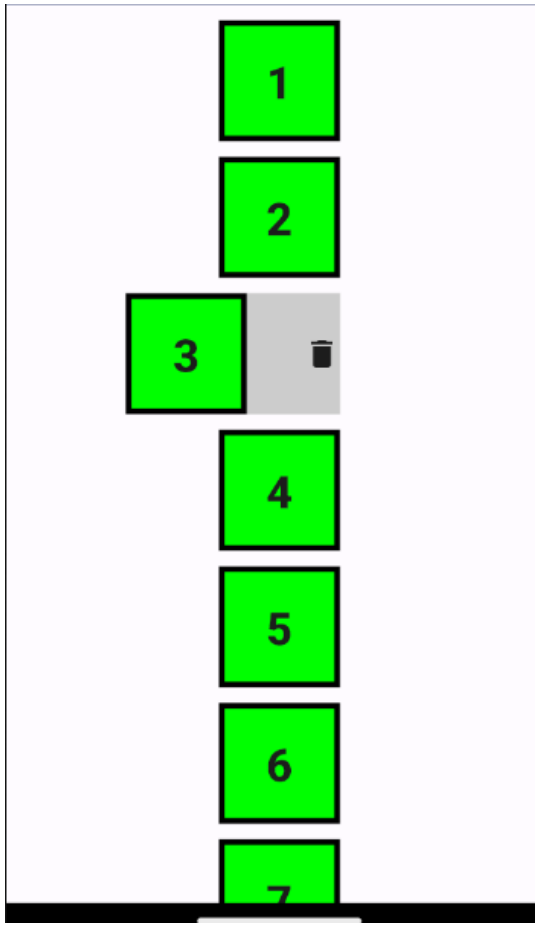
```
@Composable
fun AnimatedVisibility(
    visible: Boolean,
    modifier: Modifier = Modifier,
    enter: EnterTransition = fadeIn() + expandIn(),
    exit: ExitTransition = shrinkOut() + fadeOut(),
    label: String = "AnimatedVisibility",
    content: @Composable() AnimatedVisibilityScope.() -> Unit
) {
    val transition : Transition<Boolean> = updateTransition(visible, label)
    AnimatedEnterExitImpl(transition, { it }, modifier, enter, exit, content)
}
```

- FloatingActionButton
  - 플로팅되어 있는 작은 원형 버튼
  - 주로 화면 하단의 오른쪽 또는 왼쪽에 배치
    - 사용자가 편리하게 터치하여 주요 작업 수행할 수 있도록 함

```
FloatingActionButton(
    modifier = Modifier
        .padding(16.dp)
        .size(50.dp)
        .align(Alignment.BottomEnd),
    onClick = goToTop
) {
    Icon(
        Icons.Default.Home,
        contentDescription = "go to top"
    )
}
```

기본 사용법 실습 3

- 스와이프 할 때 삭제 기능 추가해 보기



Item Keys

- 리스트는 아이템의 위치에 따라 키가 결정됨
  - 데이터 삭제 등으로 위치가 변경되는 경우 원치 않는 결과가 초래될 수 있음
  - 키를 지정해서 아이템을 생성할 수 있음

```
LazyColumn {
    items(
        items = messages,
        key = { message ->
            // Return a stable + unique key for the item
            message.id
        }
    ) { message ->
        MessageRow(message)
    }
}
```

SwipeToDismiss

- Swipe(밀어서)로 삭제 등의 행위를 할 때 사용
  - 상태 선언

```
val dismissState = rememberDismissState(confirmStateChange = {
        if (it == DismissValue.DismissedToStart) {  // 오른쪽에서 왼쪽
                /* Dismiss 되었을 때 작업할 코드 작성 */
            true
        } else
            false
    }
)
```

*** State의 targetValue**
DismissValue.DismissedToStart
: 오른쪽에서 왼쪽으로 Dismiss 됨
DismissValue.DismissedToEnd
: 왼쪽에서 오른쪽으로 Dismiss 됨
Default : Dismiss 되지 않은 상태

  - SwipeToDismiss() 호출

```
SwipeToDismiss(
    state = dismissState,
    background = ,  // 스와이프 했을 때 적용할 배경 설정
    dismissContent = // 적용할 컨텐츠
)
```

SwipeToDismiss

- background 설정

```
background = {
    val color = when (state.dismissDirection) {
        DismissDirection.EndToStart -> Color.Red
        else -> Color.Transparent
    }

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(color)
    ) {
        Icon(
            imageVector = Icons.Default.Delete,
            contentDescription = "Delete Icon",
            modifier = Modifier.align(Alignment.CenterEnd)
        )
    }
},
```

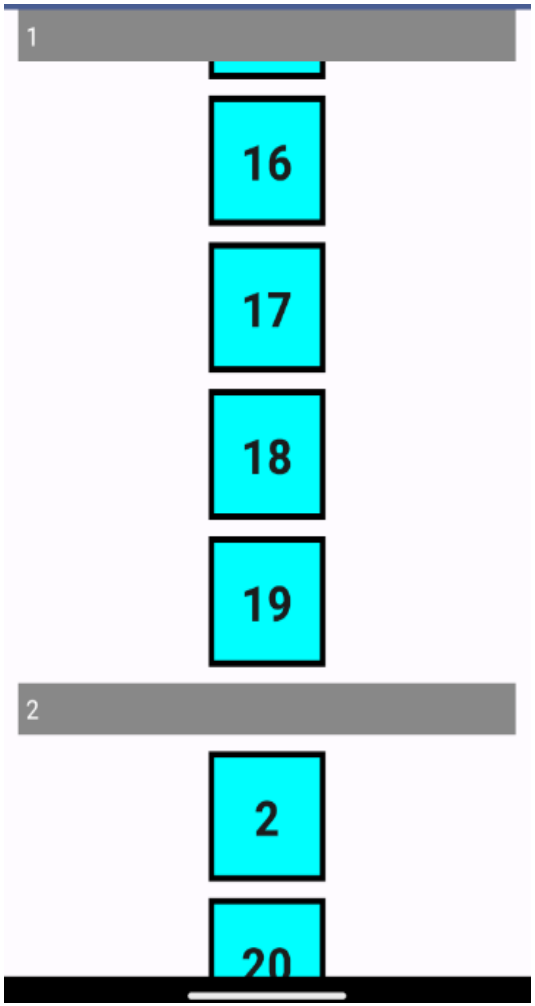
- dismissContent 설정

```
dismissContent = {
    // 스와이프할 컴포저블
}
```

*** State의 dismissDirection**
DismissDirection.EndToStart
: 오른쪽에서 왼쪽으로 Dismiss 됨
DismissDirection.StartToEnd
: 왼쪽에서 오른쪽으로 Dismiss 됨

기본 사용법 실습 4

- Sticker Header 기능 추가하기

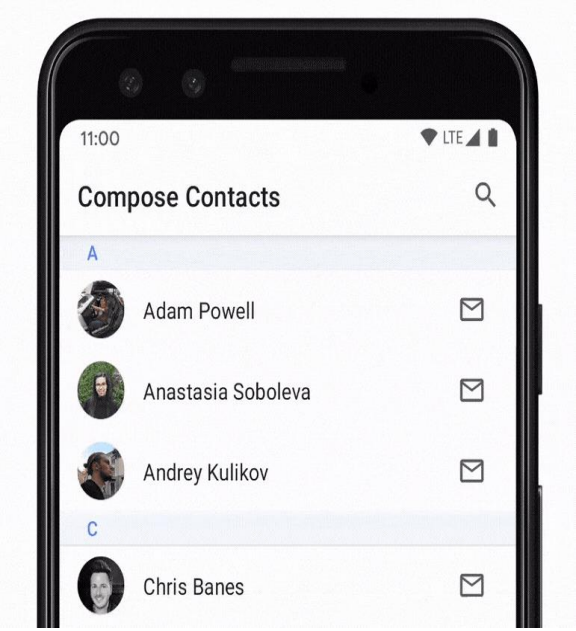


Sticky Header

- 리스트 아이템들을 한 헤더 그룹 아래 모으는 기능

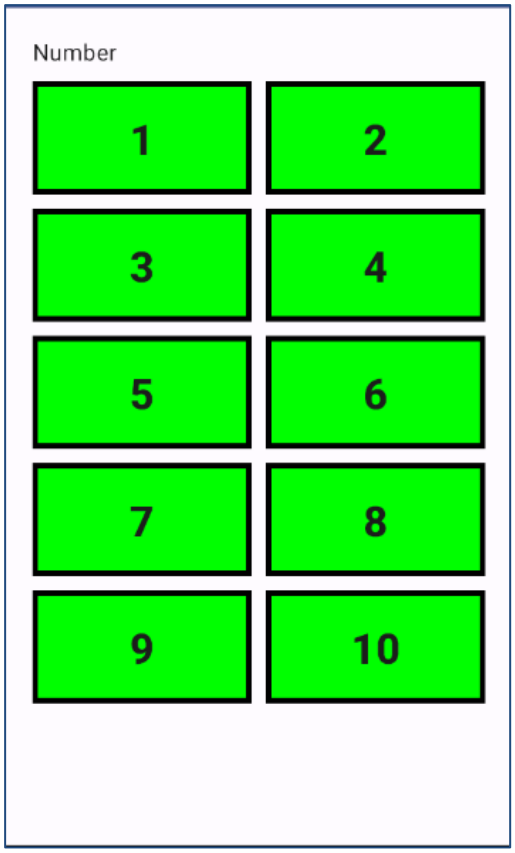
@OptIn(ExperimentalFoundationApi::class)

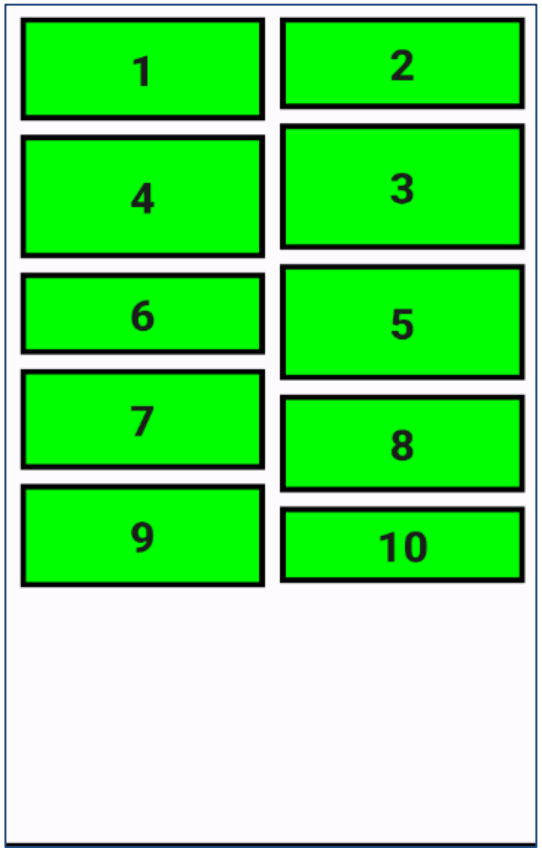
```
// This ideally would be done in the ViewModel
val grouped = contacts.groupBy { it.firstName[0] }

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun ContactsList(grouped: Map<Char, List<Contact>>) {
    LazyColumn {
        grouped.forEach { (initial, contactsForInitial) ->
            stickyHeader {
                CharacterHeader(initial)
            }

            items(contactsForInitial) { contact ->
                ContactListItem(contact)
            }
        }
    }
}
```



기본 사용법 실습 5, 6

- LazyVerticalGrid 사용하기
- LazyVerticalStaggeredGrid 사용하기





Lazy Grid

- 그리드 형태의 리스트
  - LazyHorizontalGrid / LazyVerticalGrid
  - Staggered grids
    - LazyVerticalStaggeredGrid
    - LazyHorizontalStaggeredGrid

```
LazyVerticalGrid(
    columns = GridCells.Adaptive(minSize = 128.dp)
) {
    items(photos) { photo ->
        PhotoItem(photo)
    }
}
```

Lazy Grid

- 고정된 크기

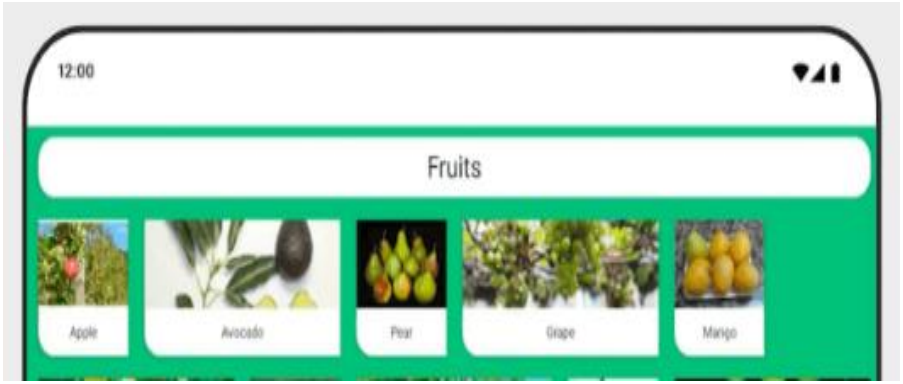
```
LazyVerticalGrid(
    columns = GridCells.Fixed(2),
    verticalArrangement = Arrangement.spacedBy(16.dp),
    horizontalArrangement = Arrangement.spacedBy(16.dp)
) {
    items(photos) { item ->
        PhotoItem(item)
    }
}
```

- Adaptive 크기

```
LazyVerticalGrid(
    columns = GridCells.Adaptive(minSize = 128.dp)
) {
    items(photos) { photo ->
        PhotoItem(photo)
    }
}
```

- Span size

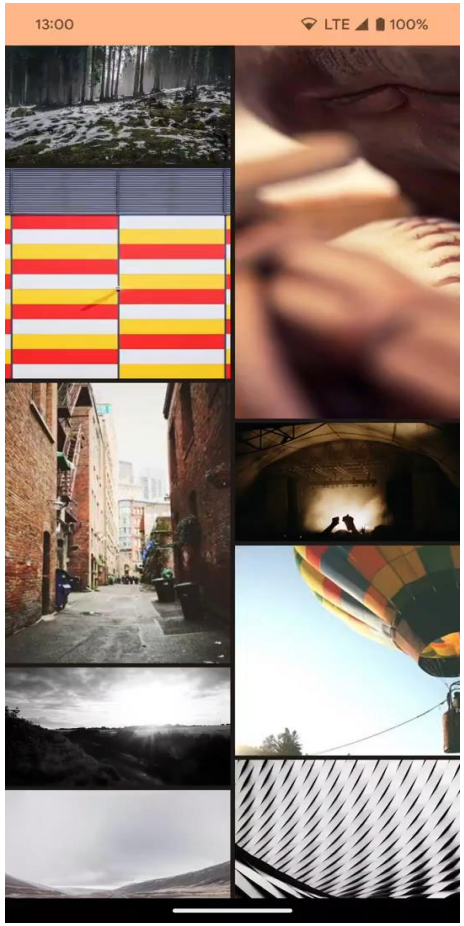
```
LazyVerticalGrid(
    columns = GridCells.Adaptive(minSize = 30.dp)
) {
    item(span = {
        // LazyGridItemSpanScope:
        // maxLineSpan
        GridItemSpan(maxLineSpan)
    }) {
        CategoryCard("Fruits")
    }
    // ...
}
```



LazyVertialStaggeredGrid

- 다른 크기의 아이템 리스트

```
LazyVerticalStaggeredGrid(
    columns = StaggeredGridCells.Adaptive(200.dp),
    verticalItemSpacing = 4.dp,
    horizontalArrangement = Arrangement.spacedBy(4.dp),
    content = {
        items(randomSizedPhotos) { photo ->
            AsyncImage(
                model = photo,
                contentScale = ContentScale.Crop,
                contentDescription = null,
                modifier = Modifier.fillMaxWidth().wrapContentHeight(
            )
        }
    },
    modifier = Modifier.fillMaxSize()
)
```



커스텀 리스트 생성 절차

- 커스텀 리스트 생성 절차
  - 리스트에 보여줄 데이터 준비
    - 데이터 클래스 정의
    - 데이터 클래스의 객체를 저장하는 리스트 생성
    - 저장 리스트에 데이터 추가
  - 리스트의 아이템 컴포저블 (레이아웃) 정의
  - Lazy Composable을 사용하여 리스트 생성

실습. 영어 단어장 만들기

- 영어 단어장
  - 파일 리소스 사용
  - TextToSpeech 사용
  - SwipeToDismiss 적용하기

File에서 데이터 읽어오기

- 내부 저장장치에서 파일 읽어오기

*파일 위치 : res/raw/datafile.txt

```
val scan =
    Scanner(context.resources.openRawResource(R.raw.datafile))

while(scan.hasNextLine()){
    val line = scan.nextLine()
}

scan.close();
```

*raw : 안드로이드 시스템에 의한 압축이나 변형없이 그대로 저장될 파일

AndroidViewModel

- ViewModel에서 Context 사용시 주의점
  - ViewModel은 Activity보다 오래 유지됨
    - Activity가 destroy 된 후 다시 create되면 실제 존재하지 않는 context를 참조할 위험성이 있음
    - ViewModel에서 context를 참조하고 하는 것은 좋지 않음
- AndroidViewModel
  - ViewModel에서 리소스 참조등의 이유로 context가 필요한 경우

    AndroidViewModel을 상속받는 클래스로 생성(applicationContext 사용)

```
class MyViewModel(private val application: Application) :
                                    AndroidViewModel(application)
    val context = application.applicationContext
```

Text To Speech

- TextToSpeech 클래스
  - https://developer.android.com/reference/android/speech/tts/TextToSpeech.html
  - 간단하게 사용하는 방법
    - TextToSpeech 클래스 객체 생성
    - Speak 함수 호출

***구글 플레이 스토어가 설치된 기기에서 실행해야 함**

Text To Speech

- 생성자 : **TextToSpeech(context, Listener)**
  - Listener는 TTS 서비스가 로딩이 완료되면 호출됨. 즉, TTS의 OnInit method가 호출되기 전까지는 대기.

```
var ttsReady =false

val tts =  TextToSpeech(context) {
                ttsReady = true
           }
```

Text To Speech

- Speak Method (읽을 문자열, Mode, RequestParams, RequestID)
  - Mode
    - TextToSpeech.QUEUE_ADD : 현재 큐에 추가
    - TextToSpeech.QUEUE_FLUSH : 현재 큐 무시하고, 추가하기

```
if(ttsReady){
    tts.speak(text, TextToSpeech.QUEUE_ADD, null, null)
}
```

DisposableEffect

- Composable이 제거된 후에 정리해야 하는 side effect를 처리하기 위한 Effect
  - 컴포저블이 컴포지션에서 제거되는 경우 (화면에서 없어짐)
  - 키 값이 변화되는 경우

```
@Composable
@NonRestartableComposable
fun DisposableEffect(
    key1: Any?,
    effect: DisposableEffectScope.() -> DisposableEffectResult
) {
    remember(key1) { DisposableEffectImpl(effect) }
}
```

- Key : 재 수행되는 것을 결정하는 파라미터
- Effect 람다식 : DisposableEffectResult를 반환하는 식

```
DisposableEffect(LocalLifecycleOwner.current)
{
  // Dispose 되어야 하는 effect 초기화
  // 예) TextToSpeech 초기화

 onDispose {
  // 이 컴포저블이 제거될 때 effect 제거
  // 예) TextToSpeech 리소스 해제

  }
}
```

실습. 영어 단어장 만들기

- 영어 단어장
  - AnimatedVisibility 추가하기

수고하셨습니다.