Практическая работа 2. Создание простого приложения с текстовыми композициями

Подготовка

В этоё практической работе вы используете `Jetpack Compose` для создания простого приложения для Android, которое выводит на экран сообщение о дне рождения.

Предварительные условия

Знать:

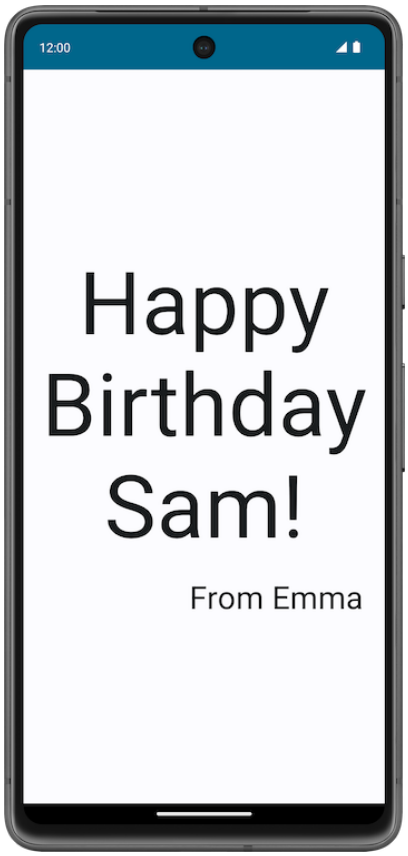
- как создать приложение в Android Studio.
- как запустить приложение на эмуляторе или устройстве Android.

Что вы узнаете

- Как писать композитные функции, такие как композитные функции Text, Column и Row.
- Как отображать текст в приложении в виде макета.
- Как форматировать текст, например, изменять его размер.

Что вы создадите

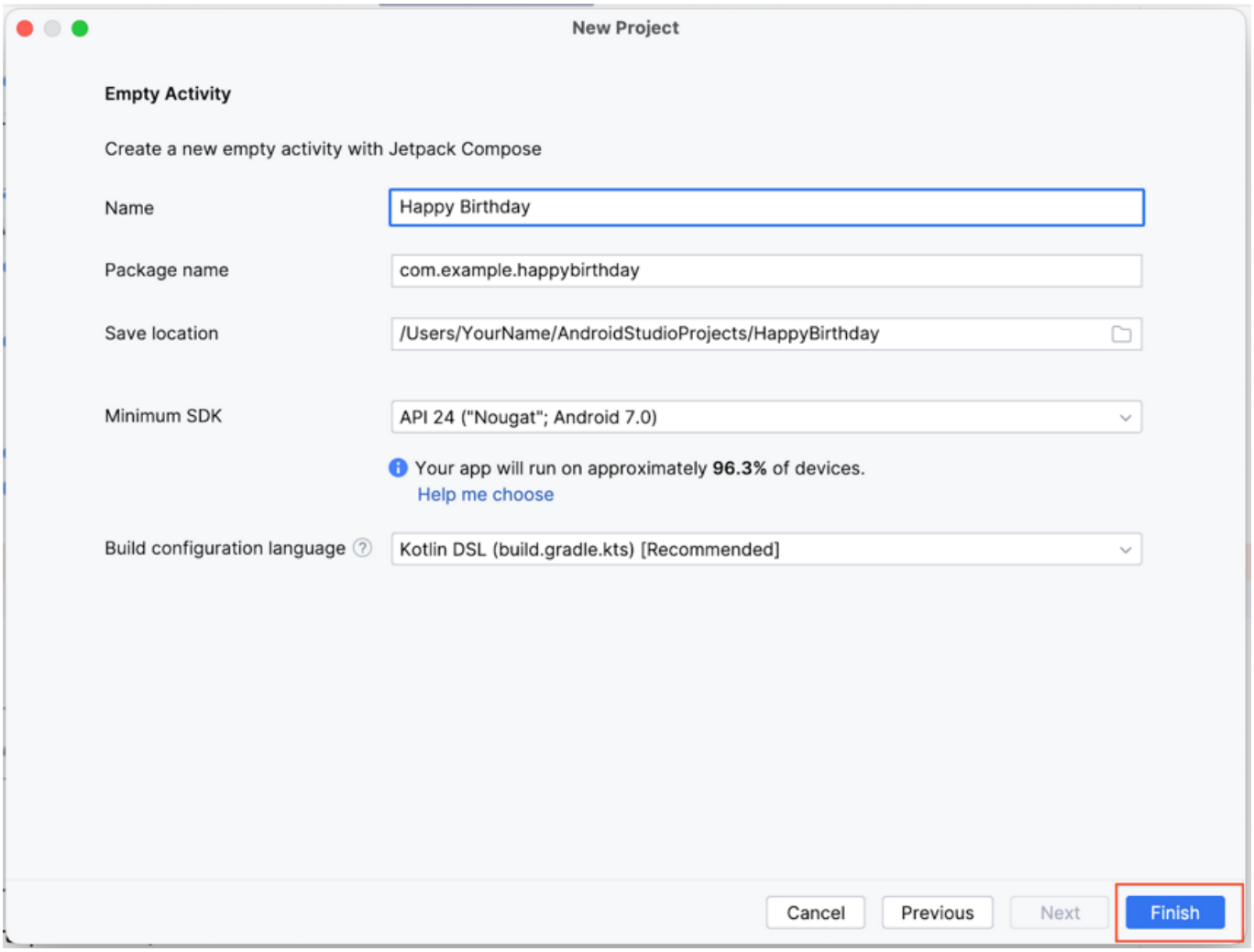
Приложение для Android, отображающее поздравление с днем рождения в текстовом формате, которое в готовом виде выглядит как на этом скриншоте:



Создание приложения "С днем рождения"

В этом задании вы создадите проект в Android Studio с шаблоном `Empty Compose Activity` и измените текстовое сообщение на персонализированное поздравление с днем рождения.

- Создайте проект `Empty Compose Activity`
- В диалоговом окне Welcome to Android Studio выберите New Project.
- В диалоговом окне Новый проект выберите Empty Compose Activity и нажмите кнопку Далее.
- В поле `Name` введите Happy Birthday, затем выберите минимальный уровень API 29 в поле Minimum SDK и нажмите Finish.



{style="width:700px;"}

- Дождитесь, пока Android Studio создаст файлы проекта и соберет его.
- Нажмите `Run app`.
- Приложение должно выглядеть как на этом скриншоте:

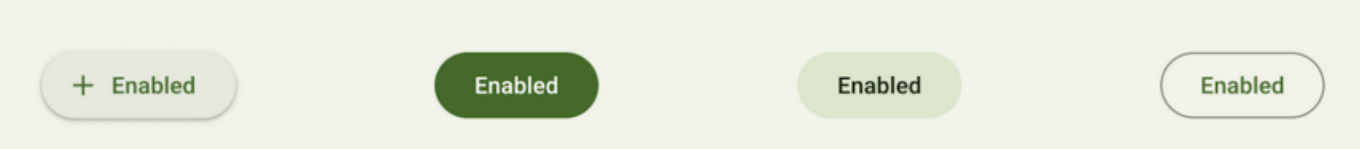
{style="width:300px;"}

Когда вы создали это приложение Happy Birthday с помощью шаблона Empty Compose Activity, Android Studio настроила ресурсы для базового приложения Android, включая сообщение Hello Android! на экране. В этом руководстве вы узнаете, как это сообщение появилось, как изменить его текст на поздравление с днем рождения, а также как добавить и отформатировать дополнительные сообщения.

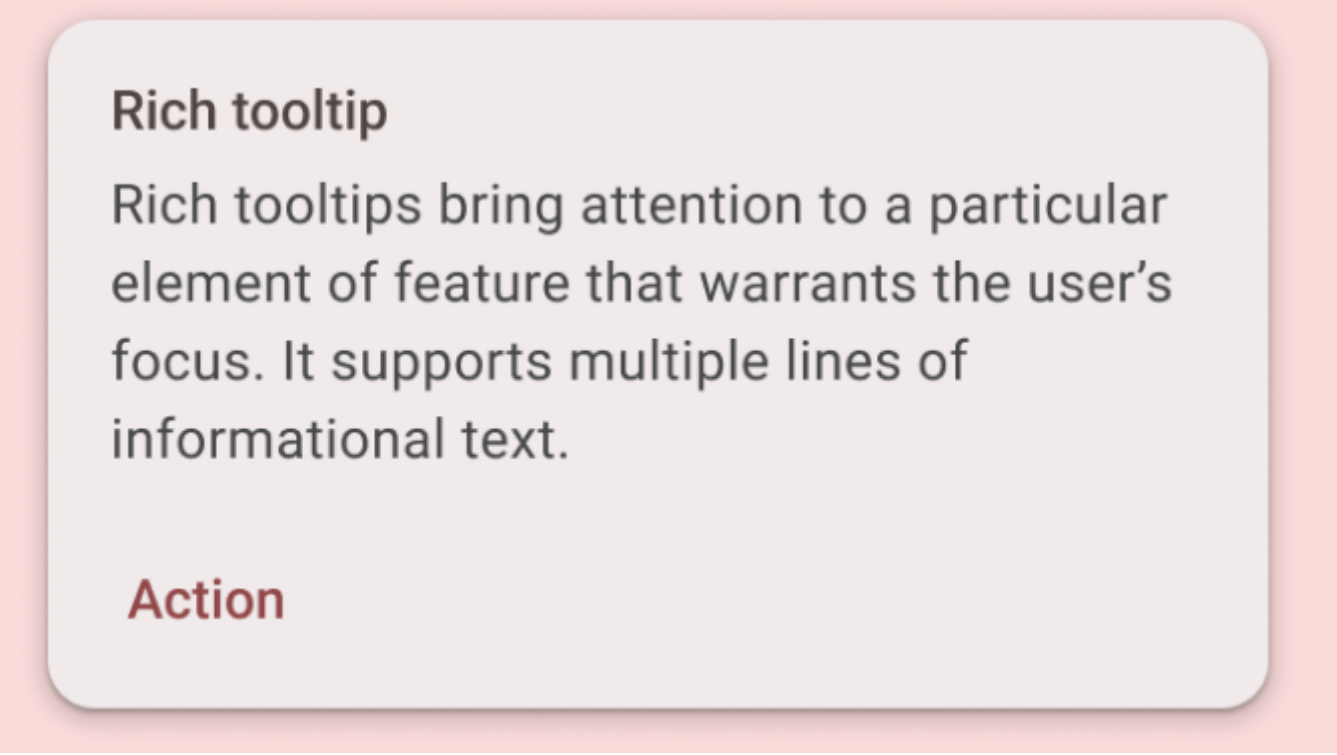
Что такое пользовательский интерфейс (UI)?

Пользовательский интерфейс (UI) приложения - это то, что вы видите на экране: текст, изображения, кнопки и многие другие элементы, а также то, как они расположены на экране. Это то, как приложение показывает все пользователю и как пользователь взаимодействует с приложением.

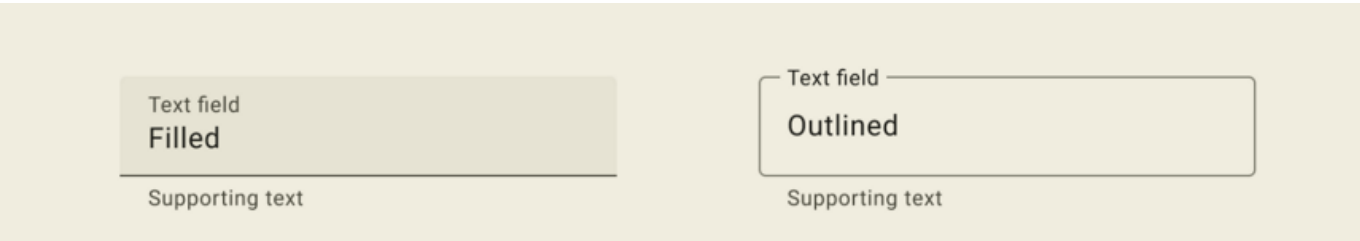
На этом изображении есть кнопка, на которую можно нажать, текстовое сообщение и поле ввода, в которое пользователь может ввести данные.



Кнопка



{style="width:400px;"} Текстовое сообщение внутри открытки

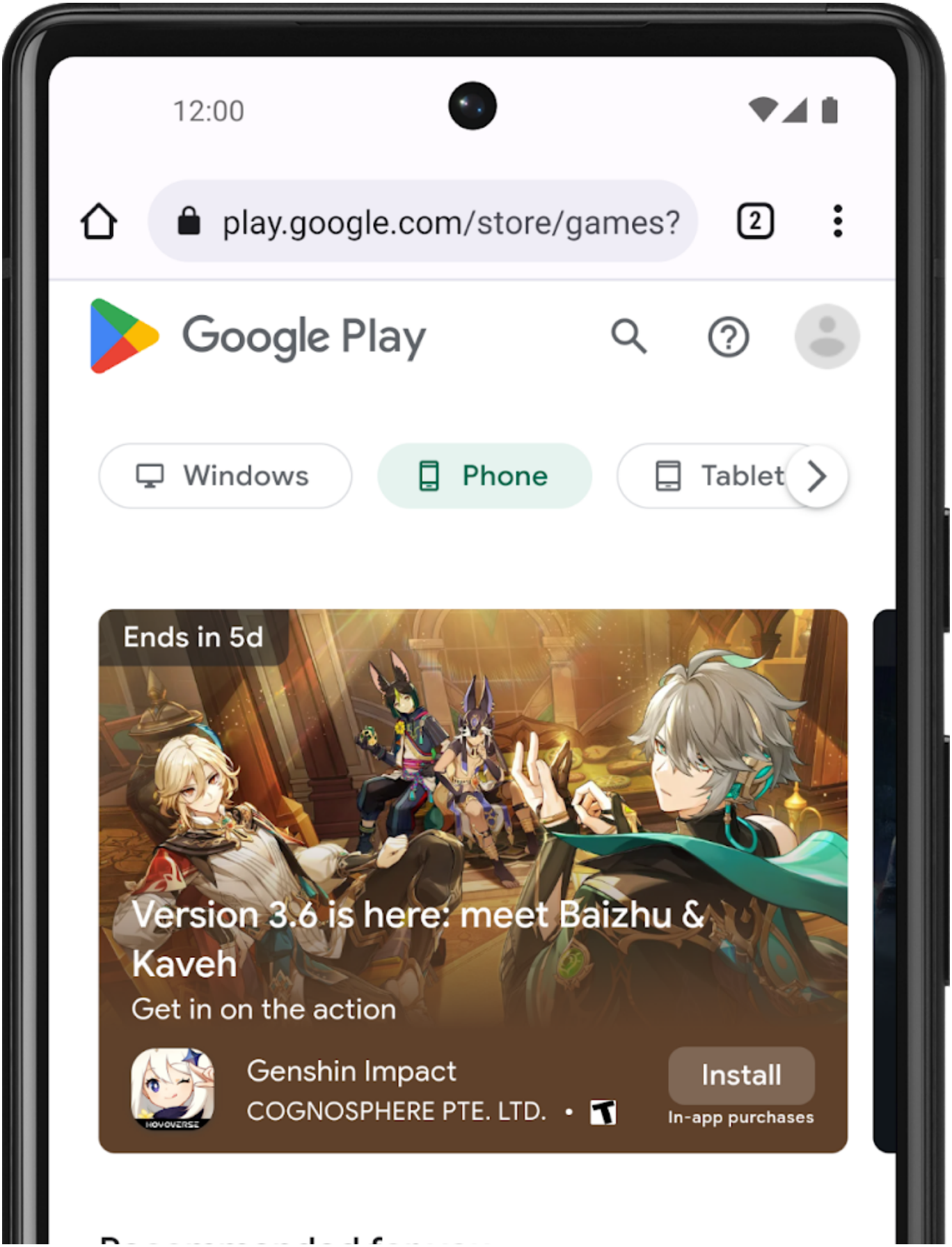


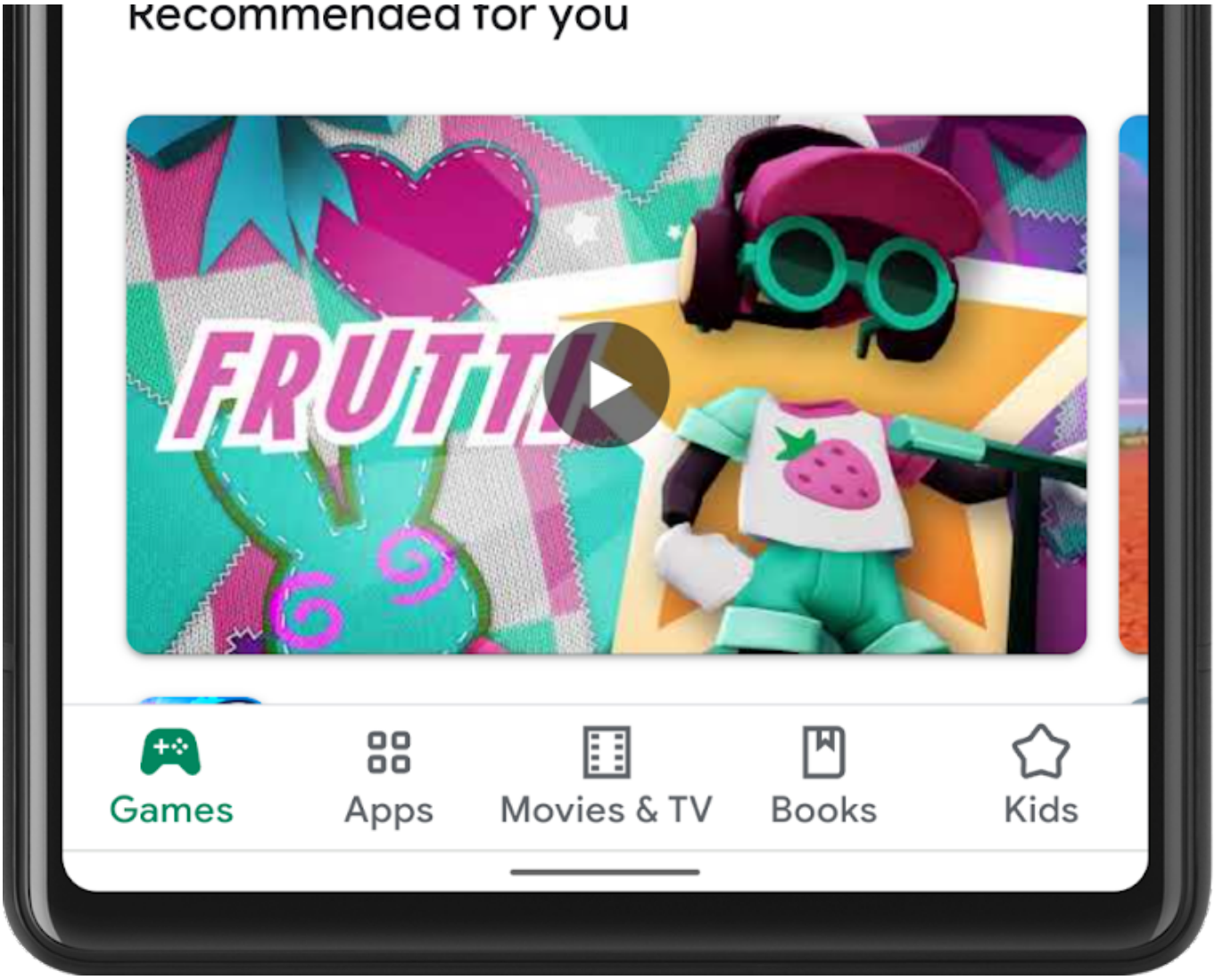
Поле ввода текста

Каждый из этих элементов называется компонентом пользовательского интерфейса. Почти все, что вы видите на экране вашего приложения, является элементом пользовательского интерфейса (также

известным как компонент пользовательского интерфейса). Они могут быть интерактивными, например, кликабельными кнопками или редактируемыми полями ввода, а могут быть декоративными изображениями.

В следующих приложениях постарайтесь найти как можно больше компонентов пользовательского интерфейса.





{style="width:400px"}

В этой практичекой работе вы работаете с элементом пользовательского интерфейса, который отображает текст, называемым элементом `Text`.

Что такое Jetpack Compose?

Jetpack Compose - это современный набор инструментов для создания пользовательских интерфейсов для Android. Compose упрощает и ускоряет разработку пользовательских интерфейсов на Android благодаря меньшему количеству кода, мощным инструментам и интуитивно понятным возможностям Kotlin. С помощью Compose вы можете создавать пользовательский интерфейс, определяя набор функций, называемых композитными функциями, которые принимают данные и описывают элементы пользовательского интерфейса.

Compose function

Compose function - это основной строительный блок пользовательского интерфейса в Compose. Составная функция:

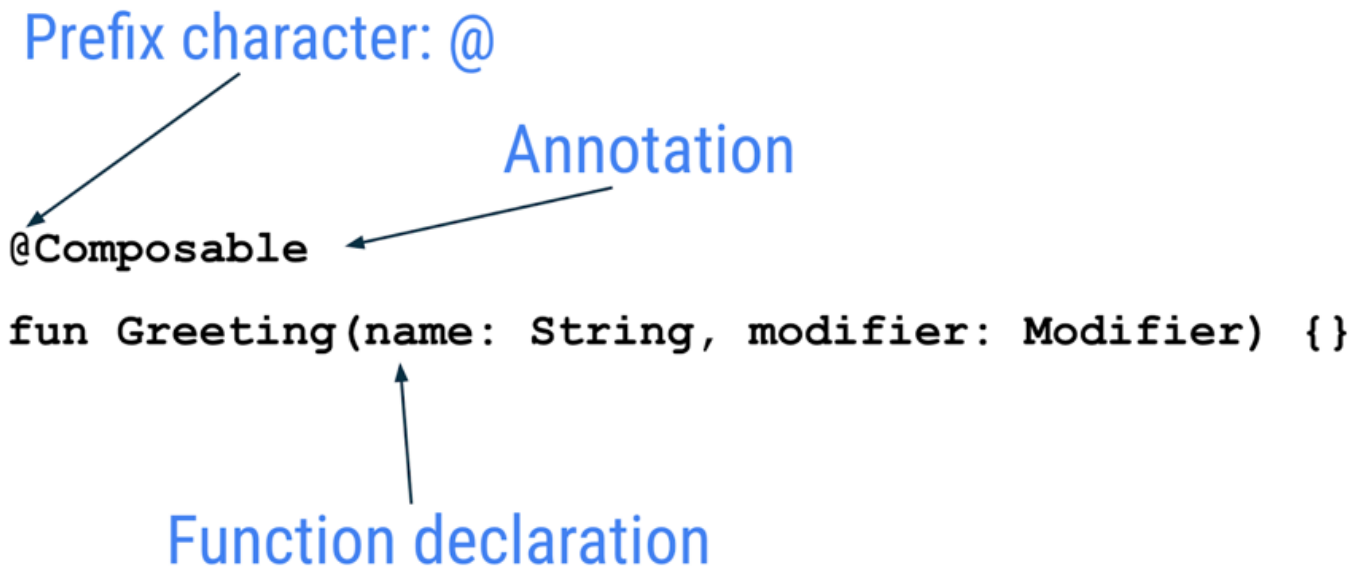
- Описывает некоторую часть вашего пользовательского интерфейса.
- Ничего не возвращает.
- Принимает некоторые входные данные и генерирует то, что отображается на экране.

Аннотации

Аннотации - это средства прикрепления дополнительной информации к коду. Эта информация помогает таким инструментам, как компилятор Jetpack Compose, и другим разработчикам понять код приложения.

Аннотация применяется путем префиксации ее имени (аннотации) с символом @ в начале объявления, которое вы аннотируете. Аннотировать можно различные элементы кода, включая свойства, функции и классы.

На следующей диаграмме приведен пример аннотированной функции:



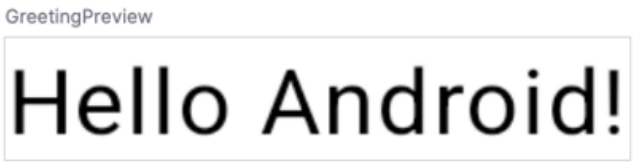
В следующем фрагменте кода приведены примеры аннотированных свойств.

```
// Example code, do not copy it over

@Json
val imgSrcUrl: String

@Volatile
private var INSTANCE: AppDatabase? = null
```

Аннотации с параметрами

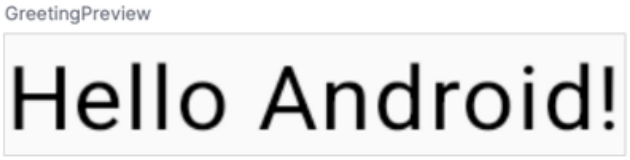
Аннотации могут принимать параметры. Параметры предоставляют дополнительную информацию инструментам, обрабатывающим их. Ниже приведены примеры аннотации `@Preview` с параметрами и без них.

```
@Preview
@Composable
fun GreetingPreview() {
    HappyBirthdayTheme {
        Greeting( name: "Android")
    }
}
```



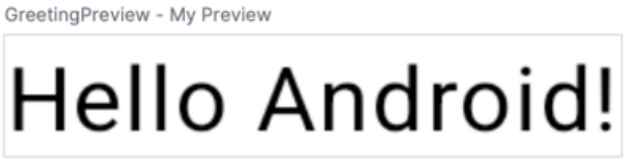
Аннотация без параметров

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



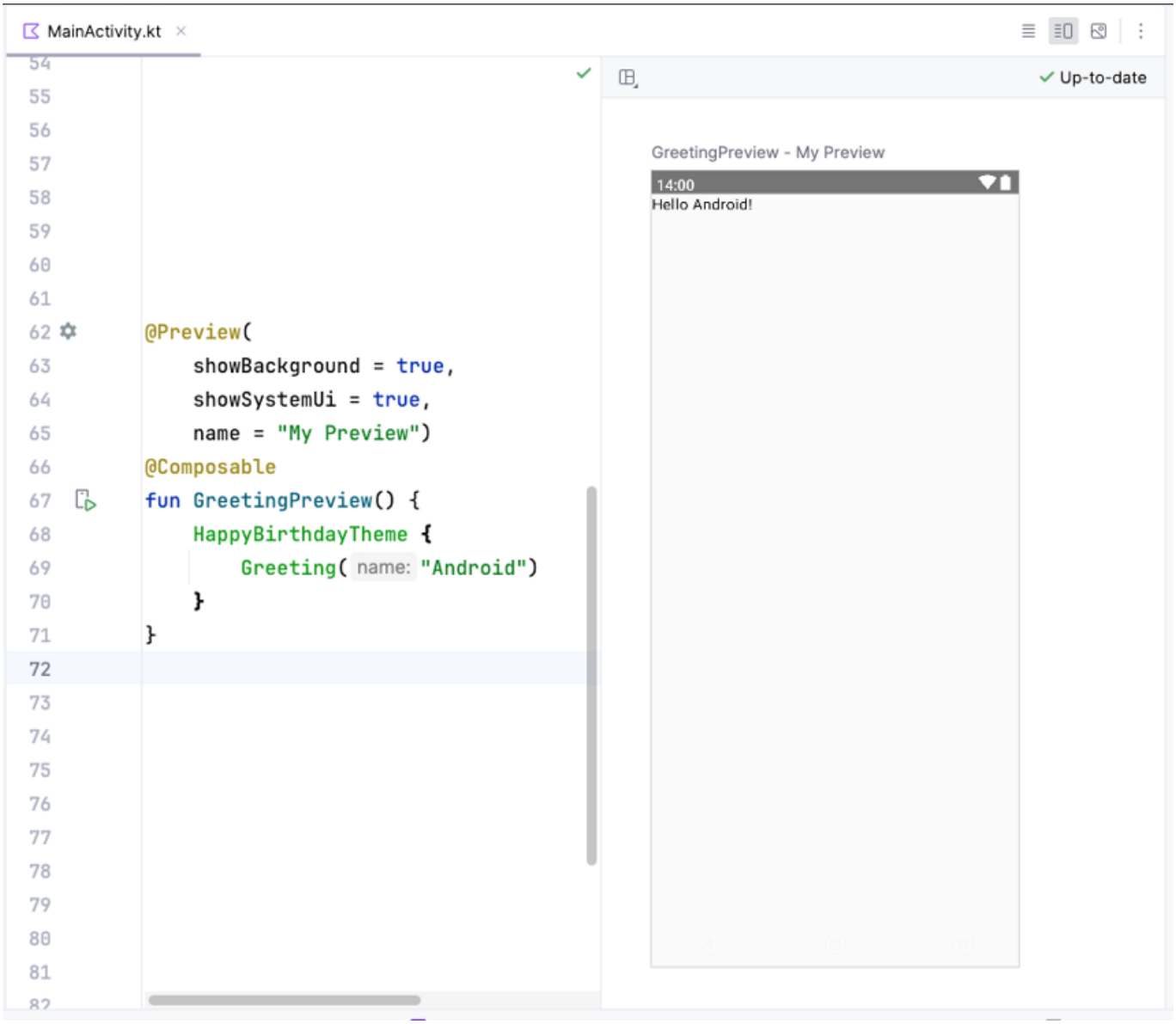
Фон для предварительного просмотра аннотаций

```
@Preview(name = "My Preview")
@Composable
fun GreetingPreview() {
    HappyBirthdayTheme {
        Greeting( name: "Android")
    }
}
```



Аннотация с заголовком предварительного просмотра

Вы можете передать аннотации несколько аргументов, как показано здесь



Снимок экрана Android studio с кодом и предварительным просмотром

Аннотация с заголовком превью и системным пользовательским интерфейсом (экран телефона)

Jetpack Compose включает в себя широкий спектр встроенных аннотаций, вы уже познакомились с аннотациями `@Composable` и `@Preview`.

Пример композитной функции

Функция Composable аннотирована аннотацией `@Composable`. Все композитные функции должны иметь эту аннотацию. Эта аннотация сообщает компилятору Compose, что данная функция предназначена для преобразования данных в пользовательский интерфейс. Напомним, что компилятор - это специальная программа, которая берет написанный вами код, просматривает его построчно и переводит в понятный компьютеру язык (машинный язык).

Этот фрагмент кода - пример простой композитной функции, которой передаются данные (параметр функции name) и которая использует их для вывода текстового элемента на экран.

```
@Composable
fun Greeting(name: String) {
```

```
    Text(text = "Hello $name!")
}
```

Несколько замечаний о композитной функции:

- Jetpack Compose построен на основе композитных функций. Эти функции позволяют вам программно определять пользовательский интерфейс вашего приложения, описывая, как он должен выглядеть, а не сосредотачиваясь на процессе создания пользовательского интерфейса. Чтобы создать композитную функцию, просто добавьте аннотацию `@Composable` к имени функции.
- Композитные функции могут принимать аргументы, которые позволяют логике приложения описывать или изменять пользовательский интерфейс. В данном случае ваш элемент пользовательского интерфейса принимает строку String, чтобы поприветствовать пользователя по имени.

Обратите внимание на составные функции в коде В Android Studio откройте файл `MainActivity.kt`. Прокрутите его до функции `GreetingPreview()`. Эта составная функция помогает выполнить предварительный просмотр функции Greeting(). Как правило, функции всегда следует называть или переименовывать, чтобы описать их функциональность. Измените название этой функции на `BirthdayCardPreview()`.

```
@Preview(showBackground = true)
@Composable
fun BirthdayCardPreview() {
    HappyBirthdayTheme {
        Greeting("Android")
    }
}
```

Композитные функции могут вызывать другие композитные функции. В этом фрагменте кода функция предварительного просмотра вызывает композитную функцию Greeting().

Обратите внимание, что предыдущая функция также имеет другую аннотацию, аннотацию @Preview, с параметром перед аннотацией @Composable. Подробнее об аргументах, передаваемых аннотации @Preview, вы узнаете позже в курсе.

Имена компонуемых функций

Функция-композитор, которая ничего не возвращает и имеет аннотацию @Composable, ДОЛЖНА быть названа с использованием регистра Паскаля. Падеж Pascal означает соглашение об именовании, в котором первая буква каждого слова в составном слове пишется с заглавной буквы. Разница между регистром Паскаля и верблюжьим регистром заключается в том, что все слова в регистре Паскаля пишутся с заглавной буквы. В верблюжьем регистре первое слово может быть в любом регистре.

Функция Compose:

- ДОЛЖНА быть существительным: DoneButton()
- НЕ глагол или глагольная фраза: DrawTextField()

- НЕ существительное с предлогом: TextFieldWithLink()
- НЕ прилагательное: Bright()
- НЕ наречие: Outside()

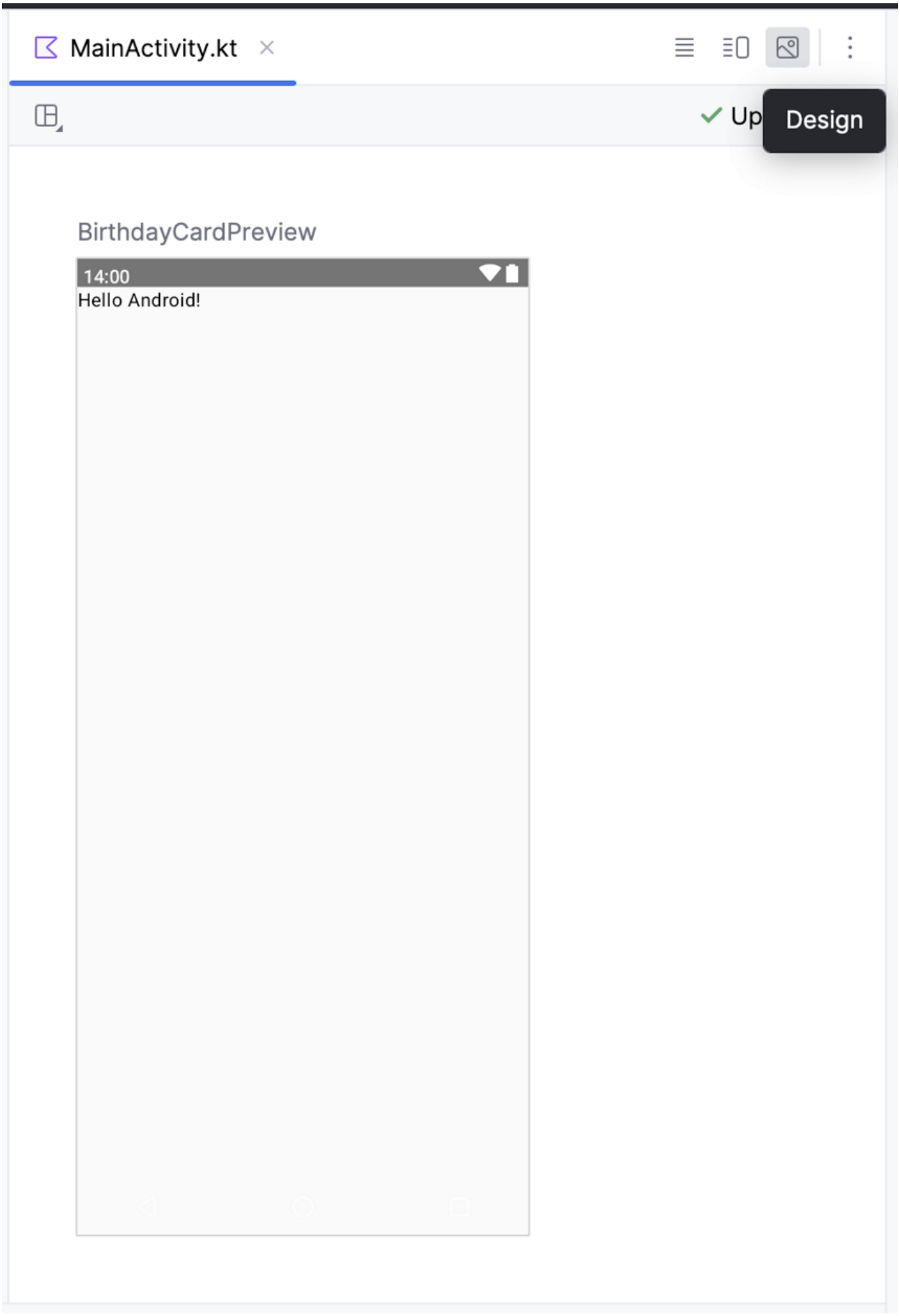
К существительным МОЖНО добавлять описательные прилагательные: RoundIcon()

Пример кода. Не копируйте

```
// Do: Эта функция представляет собой описательное существительное в паскале в
виде визуального элемента пользовательского интерфейса
@Composable
fun FancyButton(text: String) {}


// Do: Эта функция представляет собой описательное существительное в паскале как
невизуальный элемент.
// с присутствием в композиции
@Composable
fun BackButtonHandler() {}


// Не надо: эта функция является существительным, но не имеет PascalCased!
@Composable
fun fancyButton(text: String) {}


// Не надо: эта функция имеет PascalCased, но не является существительным!
@Composable
fun RenderFancyButton(text: String) {}


// Не надо: эта функция не является ни PascalCased, ни существительным!
@Composable
fun drawProfileImage(image: ImageAsset) {}
```

Панель дизайна в Android Studio

Android Studio позволяет просматривать композитные функции в IDE, а не устанавливать приложение на Android-устройство или эмулятор. Как вы узнали в предыдущем разделе, вы можете просмотреть, как выглядит ваше приложение, в панели Design в Android Studio.



{style="width:500px"}

Композитная функция должна предоставлять значения по умолчанию для любых параметров для предварительного просмотра. По этой причине не рекомендуется выполнять предварительный просмотр функции Greeting() напрямую. Вместо этого необходимо добавить другую функцию, в данном случае BirthdayCardPreview(), которая вызывает функцию Greeting() с соответствующим параметром.

```
@Preview(showBackground = true)
@Composable
fun BirthdayCardPreview() {
    HappyBirthdayTheme {
        Greeting("Android")
    }
}
```

Чтобы просмотреть предварительный просмотр:

В функции BirthdayCardPreview() измените аргумент «Android» в функции Greeting() на свое имя.

```
@Preview(showBackground = true)
@Composable
fun BirthdayCardPreview() {
    HappyBirthdayTheme {
        Greeting("James")
    }
}
```

Предварительный просмотр автоматически обновится. Вы должны увидеть обновленный предварительный просмотр.

> Важно: Код, который вы добавили в функцию BirthdayCardPreview() с аннотацией @Preview, предназначен только для предварительного просмотра в панели Design в Android Studio. Эти изменения не отражаются в приложении. Как вносить изменения в приложение, вы узнаете позже в этом уроке.

Добавление нового текстового элемента

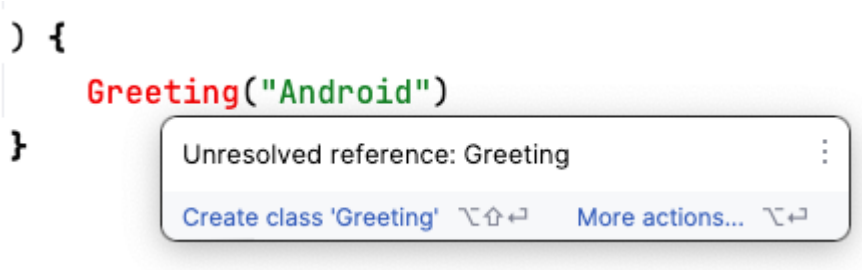
В этом задании вы удалите приветствие `Hello $name!` и добавите поздравление с днем рождения.

Добавьте новую составную функцию В файле `MainActivity.kt` удалите определение функции Greeting(). Позже вы добавите собственную функцию для отображения приветствия в codelab.

Удалите следующий код

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

Внутри функции `onCreate()` обратите внимание, что вызов функции Greeting() теперь окрашен в красный цвет. Этот красный цвет указывает на ошибку. Наведите курсор на этот вызов функции, и Android Studio отобразит информацию об ошибке.



Удалите вызов функции Greeting() вместе с ее аргументами из функций onCreate() и BirthdayCardPreview(). Ваш файл MainActivity.kt будет выглядеть примерно так:

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

@Preview(showBackground = true)
@Composable
fun BirthdayCardPreview() {
    HappyBirthdayTheme {
    }
}
```

Перед функцией BirthdayCardPreview() добавьте новую функцию GreetingText(). Не забудьте добавить аннотацию @Composable перед функцией, потому что это будет композитная функция, описывающая композитный текст.

```
@Composable
    fun GreetingText() {
}
```

Лучше всего, если ваш Composable будет принимать параметр `Modifier` и передавать этот модификатор своему первому дочернему элементу. Подробнее о `Modifier` и дочерних элементах вы узнаете в последующих заданиях. А пока добавьте параметр `Modifier` в функцию GreetingText().

```
@Composable
fun GreetingText(modifier: Modifier = Modifier) {
}
```

Добавьте параметр сообщения типа String в составную функцию GreetingText().

```
@Composable
fun GreetingText(message: String, modifier: Modifier = Modifier) {
}
```

В функцию GreetingText() добавьте составной элемент Text, передающий текстовое сообщение в качестве именованного аргумента.

```
@Composable
fun GreetingText(message: String, modifier: Modifier = Modifier) {
    Text(
        text = message
    )
}
```

Эта функция GreetingText() отображает текст в пользовательском интерфейсе. Она делает это, вызывая композитную функцию Text().

Предварительный просмотр функции

В этом задании вы сделаете предварительный просмотр функции GreetingText() в панели Design.

Вызовите функцию GreetingText() внутри функции BirthdayCardPreview(). Передайте функции GreetingText() аргумент String - поздравление с днем рождения для вашего друга. При желании вы можете добавить в него его имя, например «Happy Birthday Sam!».

```
@Preview(showBackground = true)
@Composable
fun BirthdayCardPreview() {
    HappyBirthdayTheme {
        GreetingText(message = "Happy Birthday Sam!")
    }
}
```

В панели Design обновляется автоматически. Просмотрите изменения.

Изменение размера шрифта

Вы добавили текст в пользовательский интерфейс, но он еще не выглядит как готовое приложение. В этом задании вы узнаете, как изменить размер, цвет текста и другие атрибуты, влияющие на внешний вид текстового элемента. Вы также сможете поэкспериментировать с различными размерами и цветами шрифта.

Масштабируемые пиксели Масштабируемые пиксели (SP) - это единица измерения размера шрифта. Элементы пользовательского интерфейса в приложениях Android используют две разные единицы измерения: пиксели, не зависящие от плотности (DP), которые вы используете позже для макета, и масштабируемые пиксели (SP). По умолчанию единица SP имеет тот же размер, что и DP, но ее размер изменяется в зависимости от предпочитаемого пользователем размера текста в настройках телефона.

- В файле MainActivity.kt прокрутите страницу до композита Text() в функции GreetingText().
- Передайте функции Text() аргумент fontSize в качестве второго именованного аргумента и установите для него значение 100.sp.

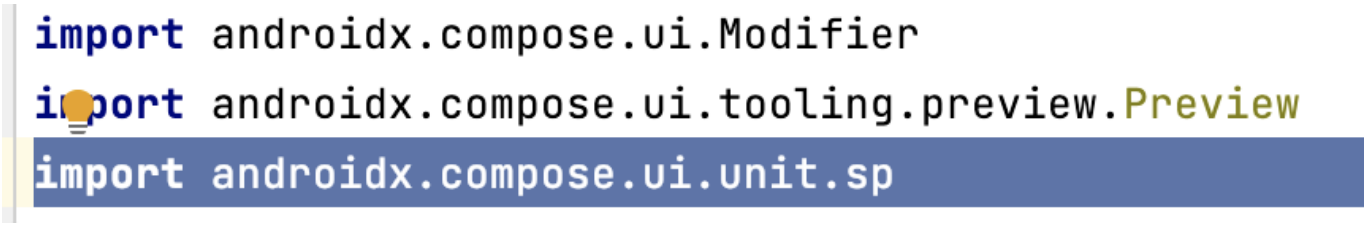
```
Text(
    text = message,
    fontSize = 100.sp
)
```

Android Studio выделяет код `.sp`, потому что для компиляции приложения необходимо импортировать некоторые классы или свойства.

- Щелкните `.sp`, который выделяется Android Studio.
- Нажмите Import во всплывающем окне, чтобы импортировать `androidx.compose.ui.unit.sp` для использования свойства расширения `.sp`.

> Примечание: Библиотека AndroidX (Android Extension) содержит набор библиотек и классов, которые помогают ускорить разработку приложений, предоставляя вам основную функциональность. Вы можете получить доступ к классам, свойствам и другим артефактам с помощью пакета androidx.

Прокрутите верхнюю часть файла и обратите внимание на операторы импорта, где вы должны увидеть оператор `import androidx.compose.ui.unit.sp`, который означает, что Android Studio добавляет пакет в ваш файл.



> Примечание: Если у вас возникнут проблемы с импортом с помощью Android Studio, вы можете вручную добавить импорт в верхней части файла.

Обратите внимание на обновленный предварительный просмотр размера шрифта. Причина перекрытия сообщений заключается в том, что вам нужно указать высоту строки.

BirthdayCardPreview

> Примечание: `sp` - это свойство расширения для Int, которое создает единицу sp. Аналогично можно использовать свойство расширения .sp в других типах данных, таких как Float и Double.

Обновите составной элемент Text, чтобы включить в него высоту строки.

```
@Composable
fun GreetingText(message: String, modifier: Modifier = Modifier) {
    Text(
        text = message,
        fontSize = 100.sp,
        lineHeight = 116.sp,
    )
}
```

Теперь вы можете поэкспериментировать с разными размерами шрифта.

Добавьте еще один текстовый элемент

В предыдущих заданиях вы добавили поздравление с днем рождения своему другу. В этом задании вы подписываете открытку своим именем.

В файле MainActivity.kt прокрутите страницу до функции GreetingText(). Передайте функции параметр from типа String для вашей подписи.

```
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier)
```

> Примечание: Порядок параметров функции не имеет значения, если вы используете именованные аргументы в вызове функции.

После текстового композита «Поздравление с днем рождения» добавьте еще один текстовый композит, который принимает текстовый аргумент, установленный в значение from.

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Text(
        // ...
    )
    Text(
        text = from
    )
}
```

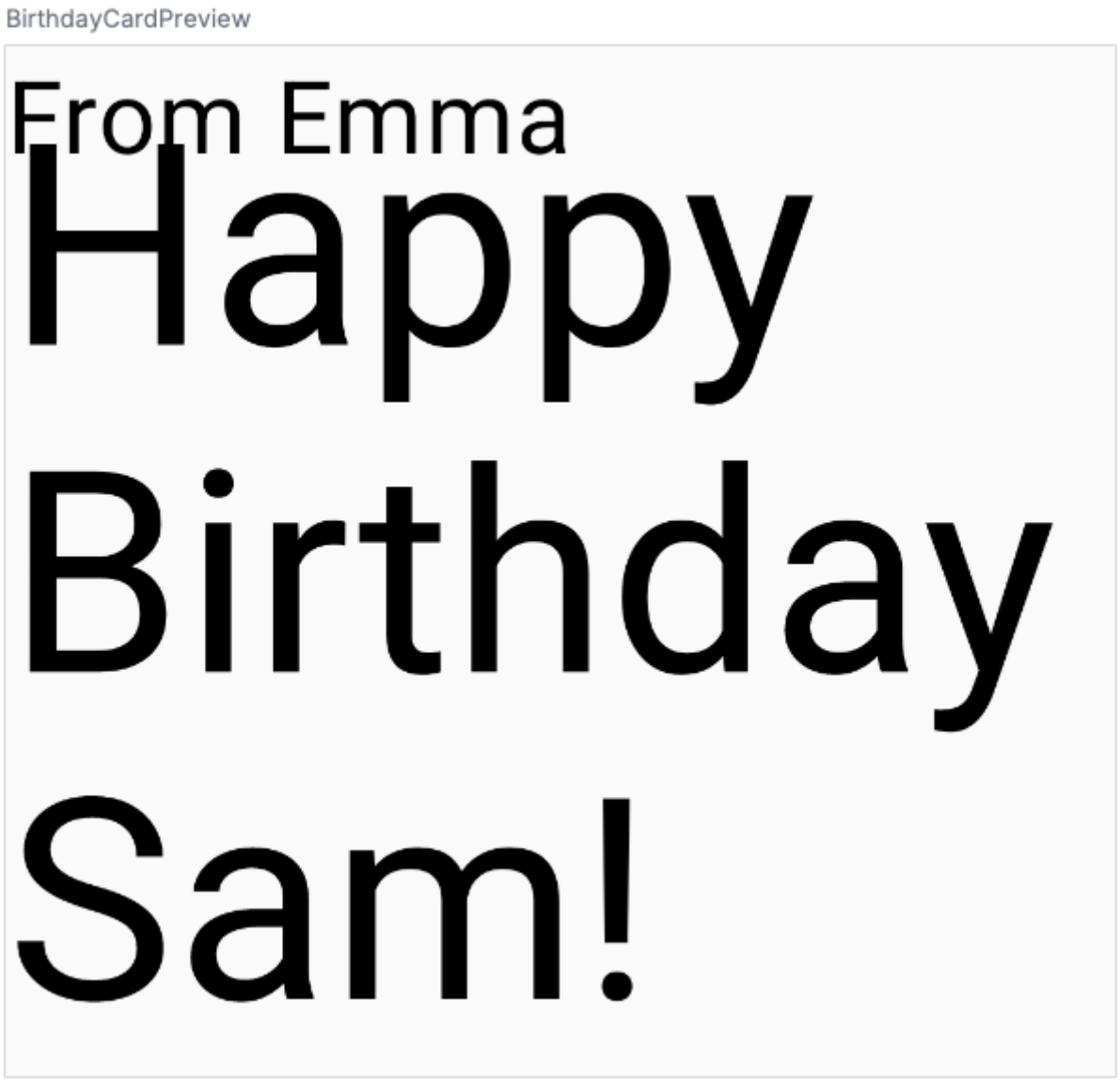
Добавьте именованный аргумент fontSize, установленный на значение 36.sp.

```
Text(
    text = from,
    fontSize = 36.sp
)
```

Перейдите к функции BirthdayCardPreview(). Добавьте еще один аргумент String для подписи вашей открытки, например «From Emma».

```
GreetingText(message = "Happy Birthday Sam!", from = "From Emma")
```

Обратите внимание на предварительный просмотр.

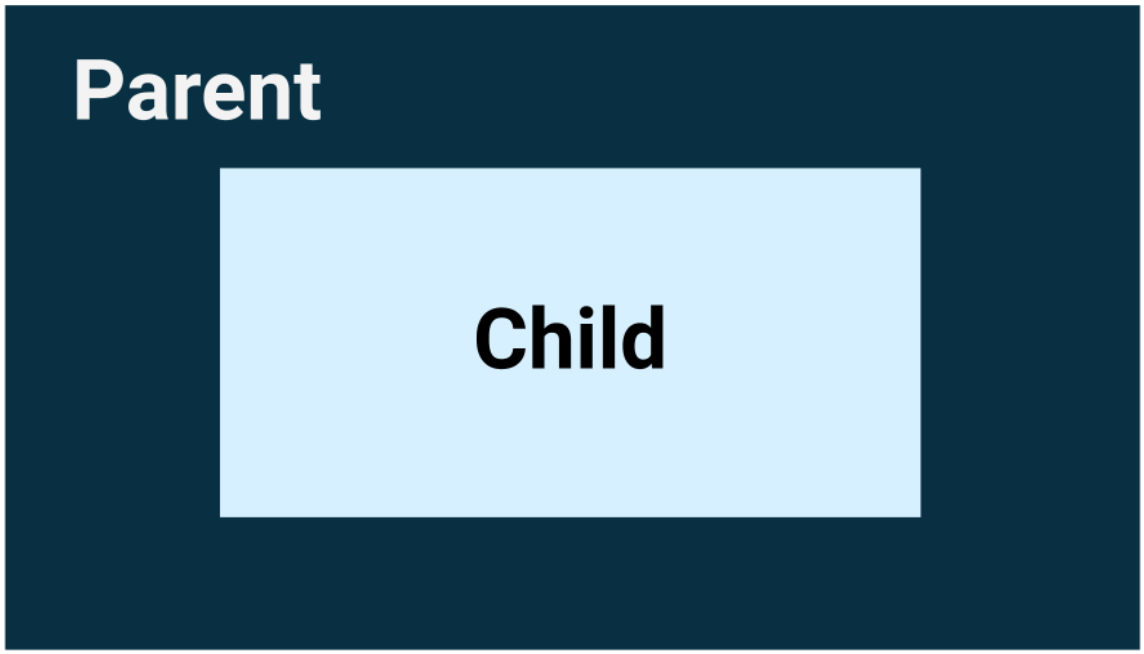


Композитная функция может описывать несколько элементов пользовательского интерфейса. Однако если вы не дадите указаний, как их расположить, Compose может расположить элементы так, как вам не понравится. Например, предыдущий код генерирует два текстовых элемента, которые накладываются друг на друга, потому что в нем нет указаний, как расположить два составных элемента.

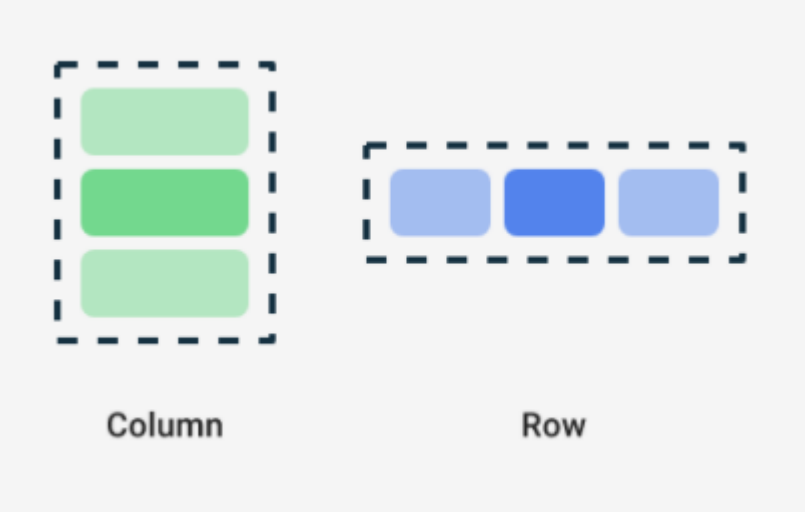
В следующем задании вы узнаете, как расположить составные элементы в строке и в столбце.

Расположите текстовые элементы в строку и колонку

Иерархия пользовательского интерфейса Иерархия пользовательского интерфейса основана на содержании, то есть один компонент может содержать один или несколько компонентов, и иногда используются термины «родитель» и «ребенок». В данном случае речь идет о том, что родительские элементы пользовательского интерфейса содержат дочерние элементы пользовательского интерфейса, которые, в свою очередь, могут содержать дочерние элементы пользовательского интерфейса. В этом разделе вы узнаете о композитных элементах `Column`, `Row` и `Box`, которые могут выступать в качестве родительских элементов пользовательского интерфейса.



Три основных стандартных элемента макета в Compose - это композиты `Column`, `Row` и `Box`.



`Column`, `Row` и `Box` - это составные функции, которые принимают в качестве аргументов составное содержимое, поэтому вы можете размещать элементы внутри этих элементов макета. Например, каждый дочерний элемент внутри композитного элемента `Row` размещается горизонтально рядом друг с другом в строке.

```
// Don't copy.
Row {
    Text("First Column")
```

```
    Text("Second Column")
}
```

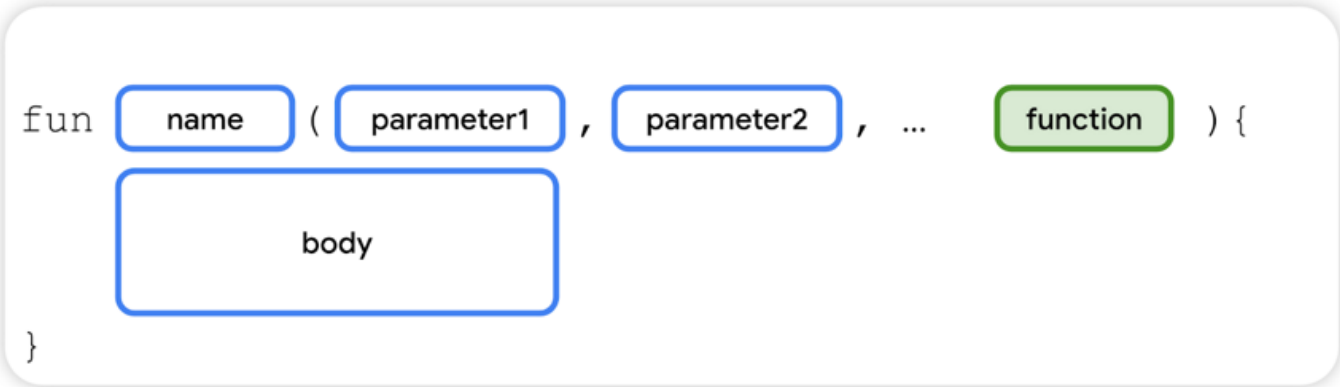
Эти текстовые элементы отображаются рядом друг с другом на экране, как показано на этом изображении.

Синие границы используются только в демонстрационных целях и не отображаются.

Синтаксис лямбды в конце строки

> Обратите внимание, что в предыдущем фрагменте кода вместо круглых скобок в функции Row composable используются фигурные скобки. Это называется синтаксисом прицепных лямбд. Подробно о лямбдах и синтаксисе спускающихся лямбд вы узнаете позже в курсе. Пока же ознакомьтесь с этим часто используемым синтаксисом Compose.

Kotlin предлагает специальный синтаксис для передачи функций в качестве параметров функций, когда последний параметр является функцией.



Когда в качестве параметра передается функция, можно использовать синтаксис лямбды. Вместо того чтобы помещать функцию внутри круглых скобок, вы можете поместить ее вне круглых скобок в фигурных скобках. Это рекомендуемая и распространенная практика в Compose, поэтому вам нужно знать, как выглядит код.

Например, последним параметром в функции Row() composable является параметр `content` - функция, описывающая дочерние элементы пользовательского интерфейса. Предположим, вы хотите создать строку, содержащую три текстовых элемента. Этот код будет работать, но использование именованного параметра для последующей лямбды очень громоздко:

```
Row(
    content = {
        Text("Some text")
```

```
        Text("Some more text")
        Text("Last text")
    }
)
```

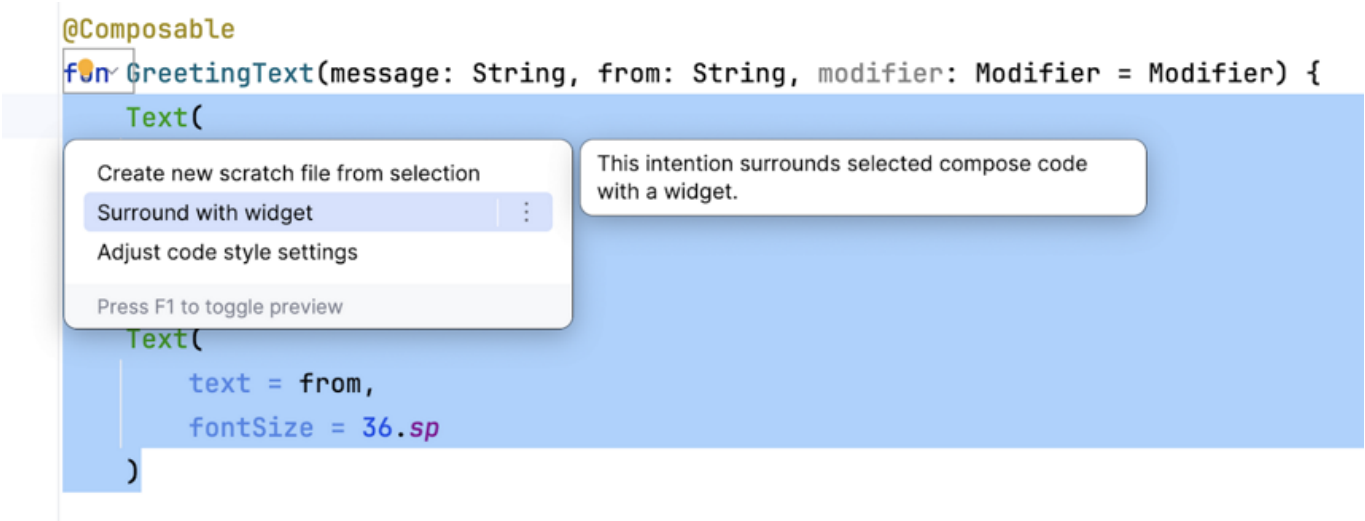
Поскольку параметр `content` является последним в сигнатуре функции, и вы передаете его значение в виде лямбда-выражения (ничего страшного, если вы не знаете, что такое лямбда, просто ознакомьтесь с синтаксисом), вы можете удалить параметр content и круглые скобки следующим образом:

```
Row {
    Text("Some text")
    Text("Some more text")
    Text("Last text")
}
```

Расположите текстовые элементы в ряд

В этом задании вы расположите текстовые элементы в вашем приложении в ряд, чтобы избежать наложения.

В файле MainActivity.kt прокрутите страницу до функции GreetingText(). Добавьте композит `Row` вокруг текстовых элементов так, чтобы он показывал строку с двумя текстовыми элементами. Выберите два текстовых элемента, щелкните на лампочке. Выберите `Surround with widget > Surround with Row`.



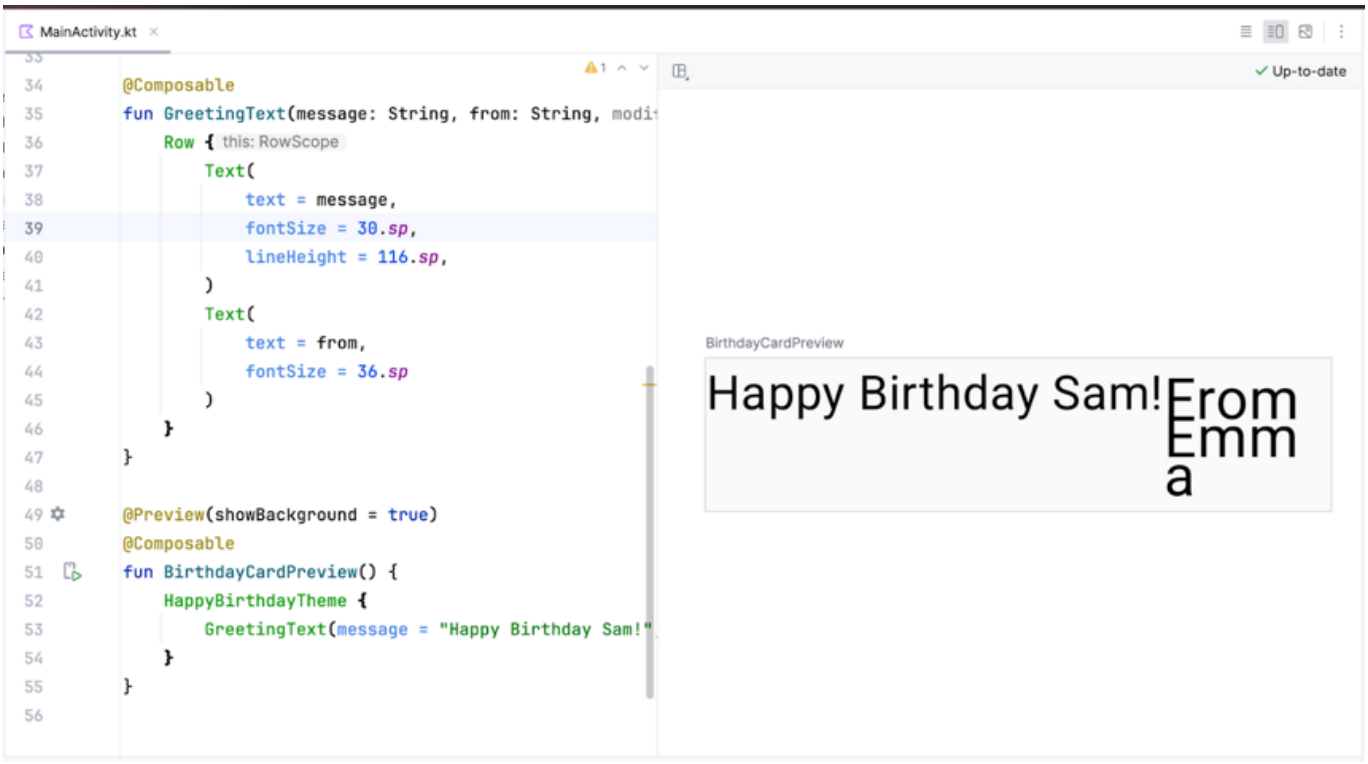
Теперь функция должна выглядеть так, как показано в этом фрагменте кода:

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Row {
        Text(
            text = message,
            fontSize = 100.sp,
            lineHeight = 116.sp,
        )
```

```
            Text(
                text = from,
                fontSize = 36.sp
            )
        }
    }
```

Android Studio автоматически импортирует функцию `Row`. Прокрутите страницу вверх и обратите внимание на раздел импорта. Импорт `androidx.compose.foundation.layout.Row` должен был быть добавлен.

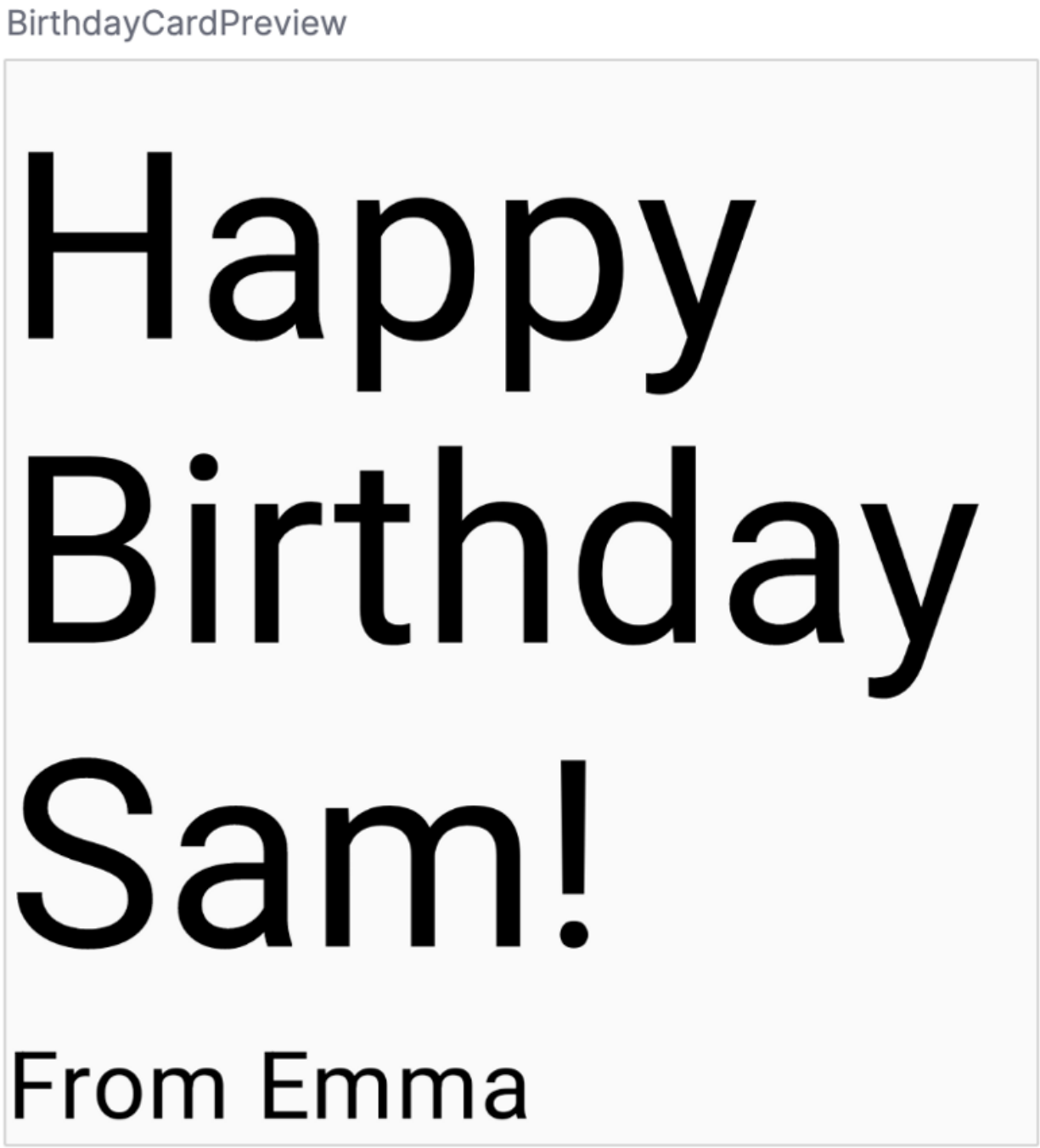
Обратите внимание на обновленный предварительный просмотр в панели Design. Временно измените размер шрифта для сообщения о дне рождения на 30.sp.



Теперь предварительный просмотр выглядит гораздо лучше, так как нет перекрытия. Однако это не то, что вы хотите, потому что не хватает места для подписи. В следующем задании вы расположите текстовые элементы в столбце, чтобы решить эту проблему.

Расположите текстовые элементы в столбце

В этом задании вам предстоит изменить функцию GreetingText(), чтобы расположить текстовые элементы в столбце. Предварительный просмотр должен выглядеть как на этом скриншоте:



Теперь, когда вы попробовали сделать это самостоятельно, не стесняйтесь сверять свой код с кодом решения в этом фрагменте:

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Column {
        Text(
            text = message,
            fontSize = 100.sp,
            lineHeight = 116.sp,
        )
        Text(
            text = from,
```

```
            fontSize = 36.sp
        )
    }
}
```

Обратите внимание на автоматически импортируемый Android Studio пакет:

```
import androidx.compose.foundation.layout.Column
```

> Вспомните, что в составных элементах нужно передавать параметр модификатора дочернему элементу. Это означает, что вам нужно передать параметр модификатора композитному элементу `Column`.

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Column(modifier = modifier) {
        Text(
            text = message,
            fontSize = 100.sp,
            lineHeight = 116.sp,
        )
        Text(
            text = from,
            fontSize = 36.sp
        )
    }
}
```

Добавьте приветствие в приложение

Когда вы будете довольны предварительным просмотром, настанет время добавить композицию в ваше приложение на устройстве или эмуляторе.

В файле MainActivity.kt прокрутите страницу до функции `onCreate()`. Вызовите функцию GreetingText() из блока Surface. Передайте функции GreetingText() свое поздравление с днем рождения и подпись.

Завершенная функция onCreate() должна выглядеть так, как показано в этом фрагменте кода:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HappyBirthdayTheme {
                // A surface container using the 'background' color from the theme
                Surface(
```

```
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    GreetingText(message = "Happy Birthday Sam!", from = "From
Emma")
                }
            }
        }
    }
}
```

Создайте и запустите свое приложение в эмуляторе.


12:00
Happy Birthday Sam!
From Emma

Выравнивание приветствия по центру

Чтобы выровнять приветствие по центру экрана, добавьте параметр `verticalArrangement` и установите его в `Arrangement.Center`.

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Column(
        verticalArrangement = Arrangement.Center,
        modifier = modifier
    ) {
        // ...
    }
}
```

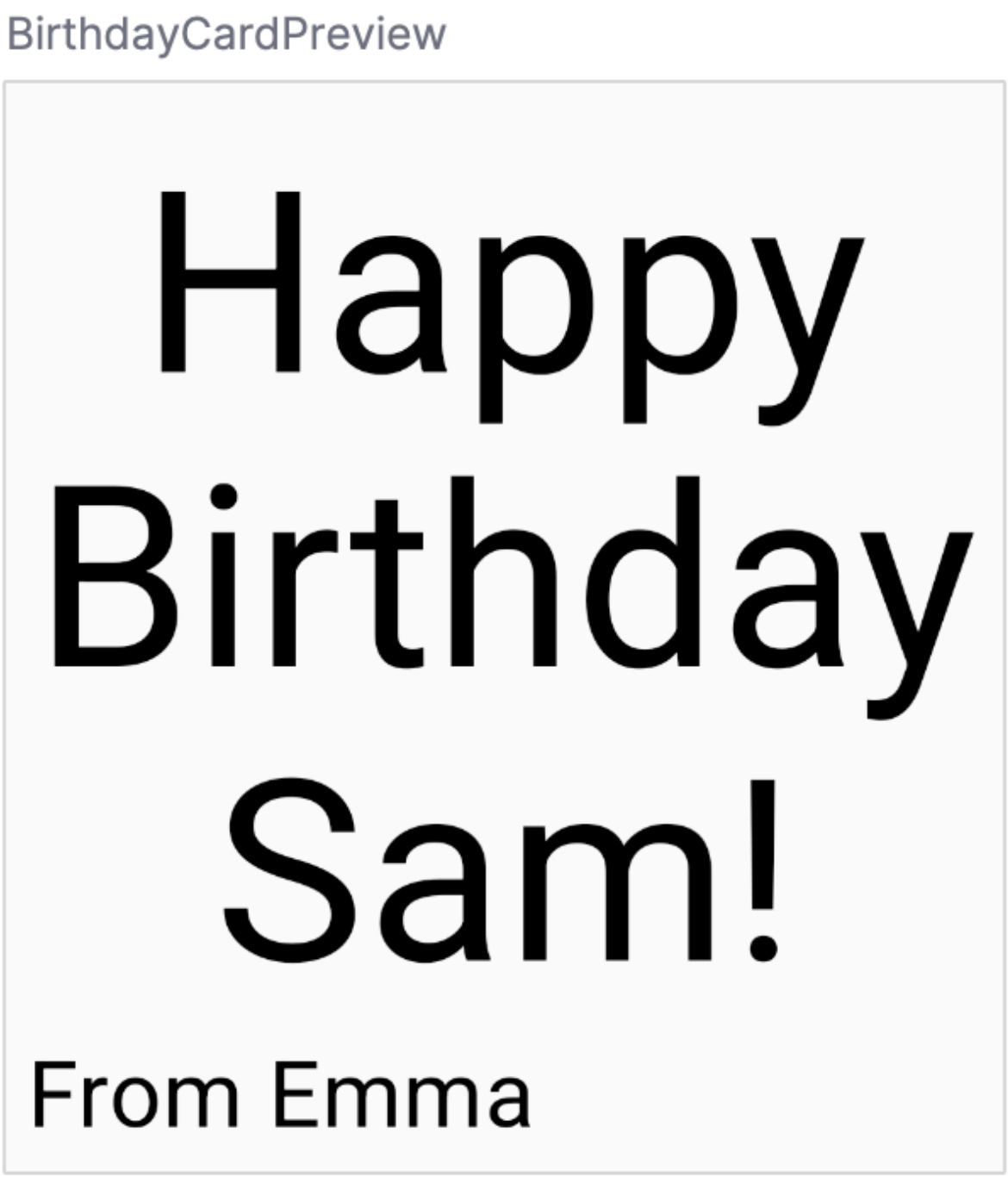
Добавьте 8.dp подкладки вокруг колонки.

> Хорошей практикой является использование отступов с шагом 4.dp.

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Column(
        verticalArrangement = Arrangement.Center,
        modifier = modifier.padding(8.dp)
    ) {
        // ...
    }
}
```

Чтобы еще больше украсить ваше приложение, выровняйте текст приветствия по центру с помощью `textAlign`.

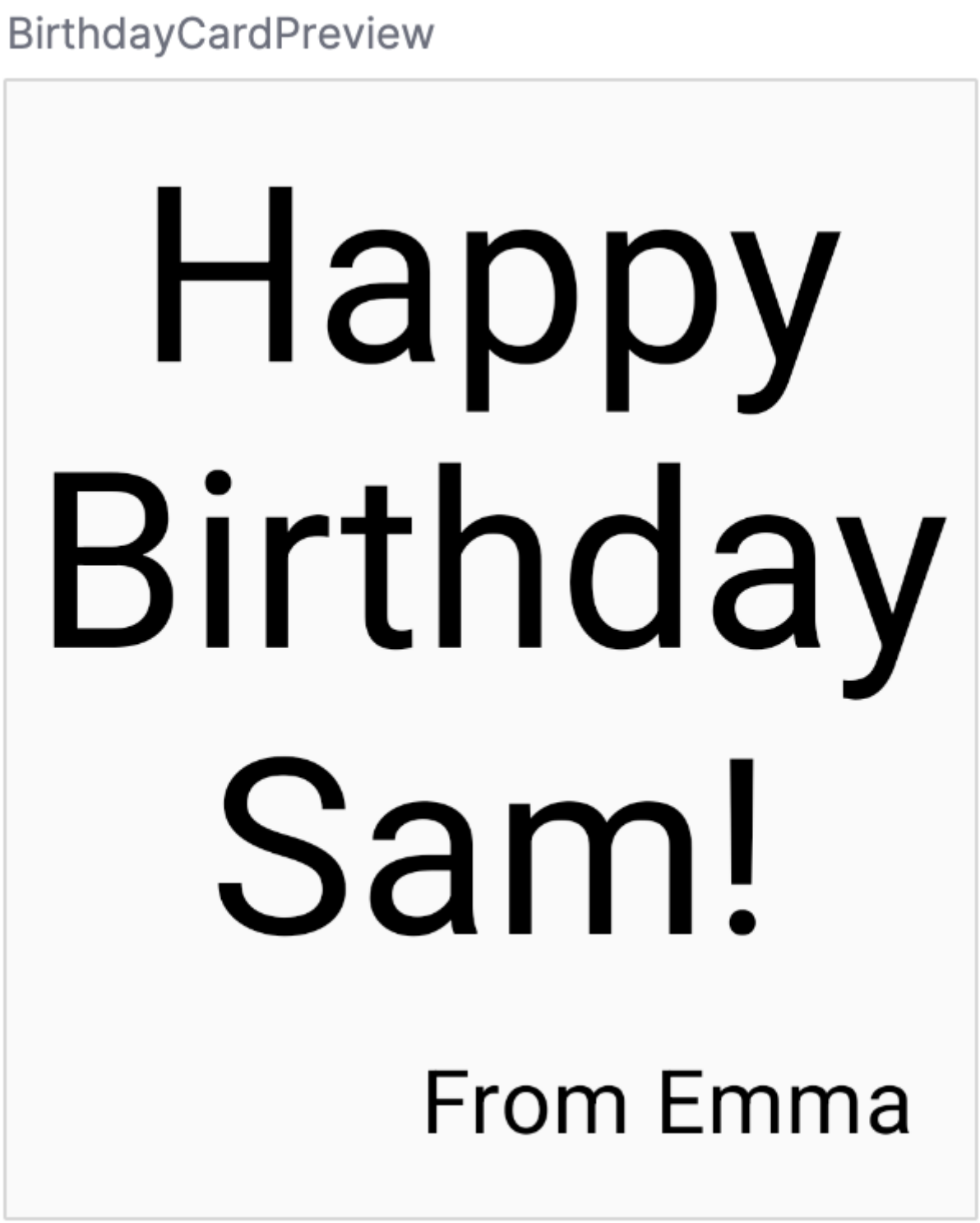
```
Text(
    text = message,
    fontSize = 100.sp,
    lineHeight = 116.sp,
    textAlign = TextAlign.Center
)
```



На скриншоте выше только приветствие выровнено по центру благодаря параметру `textAlign`. Подпись «From Emma» имеет выравнивание по умолчанию - по левому краю.

Добавьте `padding` к подписи и выровняйте ее по правому краю.

```
Text(
    text = from,
    fontSize = 36.sp,
    modifier = Modifier
        .padding(16.dp)
        .align(alignment = Alignment.End)
)
```



Применяйте передовую практику

> Хорошей практикой является передача атрибута(ов) модификатора вместе с модификатором из родительского композита. Обновите параметр модификатора в GreetingText() следующим образом:

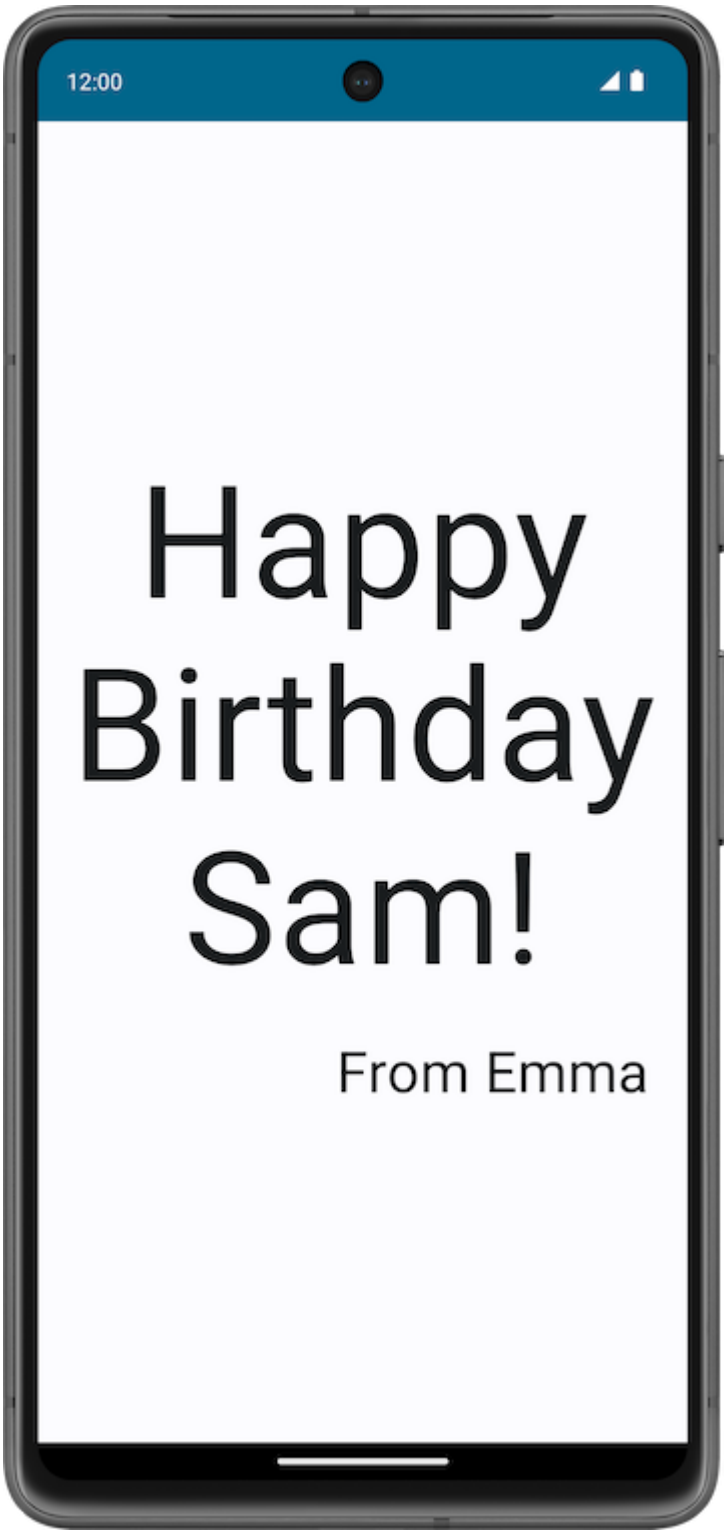
onCreate()

```
Surface(
    //...
) {
    GreetingText(
        message = "Happy Birthday Sam!",
        from = "From Emma",
        modifier = Modifier.padding(8.dp)
    )
}
```

GreetingText()

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Column(
        verticalArrangement = Arrangement.Center,
        modifier = modifier
    ) {
        // ...
    }
}
```

Создайте и запустите свое приложение в эмуляторе, чтобы увидеть конечный результат.



Получите код решения

Завершенный файл MainActivity.kt:

```
package com.example.happybirthday

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Arrangement
```

```
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.happybirthday.ui.theme.HappyBirthdayTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HappyBirthdayTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    GreetingText(
                        message = "Happy Birthday Sam!",
                        from = "From Emma",
                        modifier = Modifier.padding(8.dp)
                    )
                }
            }
        }
    }
}

@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Column(
        verticalArrangement = Arrangement.Center,
        modifier = modifier
    ) {
        Text(
            text = message,
            fontSize = 100.sp,
            lineHeight = 116.sp,
            textAlign = TextAlign.Center
        )
        Text(
            text = from,
            fontSize = 36.sp,
            modifier = Modifier
                .padding(16.dp)
                .align(alignment = Alignment.End)
```

```
        )
    }
}

@Preview(showBackground = true)
@Composable
fun BirthdayCardPreview() {
    HappyBirthdayTheme {
        GreetingText(message = "Happy Birthday Sam!", from = "From Emma")
    }
}
```

Заключение

- Вы создали свое приложение «С днем рождения».
- Позже вы добавите в приложение картинку и измените выравнивание текстовых элементов, чтобы украсить его.

Резюме

- Jetpack Compose - это современный набор инструментов для создания пользовательского интерфейса Android. Jetpack Compose упрощает и ускоряет разработку пользовательского интерфейса на Android благодаря меньшему количеству кода, мощным инструментам и интуитивно понятным API на языке Kotlin.
- Пользовательский интерфейс (UI) приложения - это то, что вы видите на экране: текст, изображения, кнопки и многие другие типы элементов.
- Композитные функции - это основной строительный блок Compose. Композитная функция - это функция, которая описывает некоторую часть вашего пользовательского интерфейса.
- Функция Composable аннотируется аннотацией `@Composable`; эта аннотация сообщает компилятору Compose, что данная функция предназначена для преобразования данных в пользовательский интерфейс.
- Три основных стандартных элемента макета в Compose - это `Column`, `Row` и `Box`. Они представляют собой функции Composable, которые принимают содержимое Composable, поэтому вы можете размещать внутри них элементы. Например, каждый дочерний элемент в `Row` будет располагаться горизонтально рядом друг с другом.