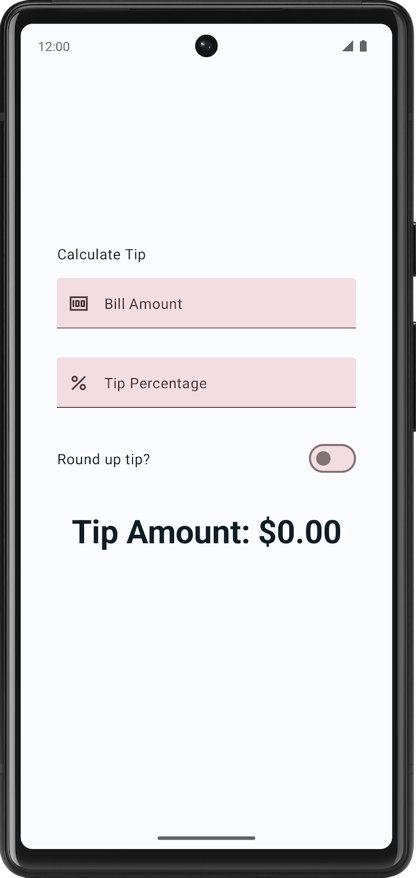
**Лабораторная работа: калькулятор чаевых (продолжение)**

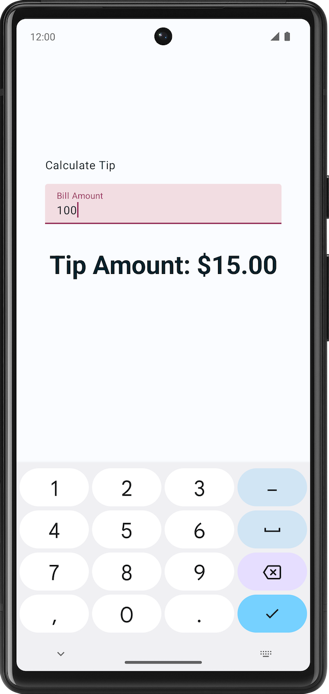
В этой лабораторной вы используете код решения из предыдущей лабораторной работы для создания интерактивного калькулятора чаевых, который может автоматически рассчитывать и округлять сумму чаевых при вводе суммы счета и процента чаевых. Вы можете увидеть окончательное приложение на этом изображении:



Начинаем с приложения Tip Time из предыдущей лаборатории [Intro для состояния в Compose](https://developer.android.com/codelabs/basic-android-kotlin-compose-using-state), которое предоставляет пользовательский интерфейс, необходимый для расчета чаевых с фиксированным процентом чаевых. Текстовое поле **Сумма счета** позволяет пользователю ввести стоимость услуги. Приложение рассчитывает и отображает сумму чаевых в составном виде Text.

**Запустите приложение Tip Time**

1. Откройте проект Tip Time в Android Studio и запустите приложение на эмуляторе или устройстве.
2. Введите сумму счета. Приложение автоматически рассчитывает и отображает сумму чаевых.



В текущей реализации процент чаевых жестко запрограммирован на уровне 15%. В этой кодовой лаборатории вы расширяете эту функцию с помощью текстового поля, которое позволяет приложению рассчитывать процент чаевых и округлять сумму чаевых.

Добавьте необходимые строковые ресурсы

1. На вкладке **«Проект»** нажмите **«res» > «values» > «strings.xml»**.
2. Между тегами <resources> файла strings.xml добавьте следующие строковые ресурсы:

<string name="how\_was\_the\_service">Tip Percentage</string>  
<string name="round\_up\_tip">Round up tip?</string>

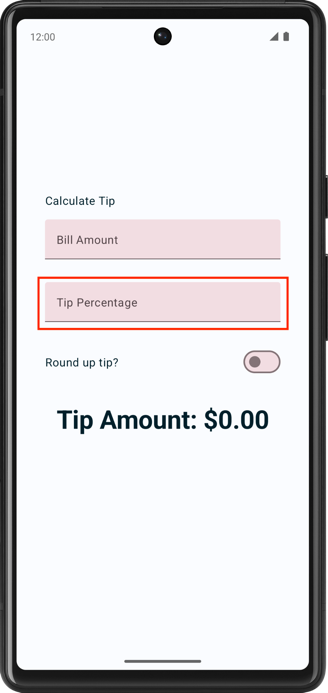
Файл strings.xml должен выглядеть как этот фрагмент кода, который включает строки из предыдущей кодовой лаборатории:

strings.xml

<resources>  
    <string name="app\_name">Tip Time</string>  
    <string name="calculate\_tip">Calculate Tip</string>  
    <string name="bill\_amount">Bill Amount</string>  
**<string name="how\_was\_the\_service">Tip Percentage</string>  
    <string name="round\_up\_tip">Round up tip?</string>**    <string name="tip\_amount">Tip Amount: %s</string>  
</resources>

[4. Добавьте текстовое поле с процентом чаевых.](https://developer.android.com/codelabs/basic-android-kotlin-compose-calculate-tip?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-calculate-tip#3)

Клиент может захотеть оставить больше или меньше чаевых в зависимости от качества предоставленной услуги и других причин. Чтобы учесть это, приложение должно позволять пользователю рассчитывать индивидуальные чаевые. В этом разделе вы добавляете текстовое поле, в котором пользователь может ввести процент чаевых, как вы можете видеть на этом изображении:



В вашем приложении уже есть составное текстовое поле **«Сумма счета»** EditNumberField(), которое представляет собой составную функцию без сохранения состояния. В предыдущей лабораторной вы передали состояние amountInput из составного объекта EditNumberField() в составной TipTimeLayout(), что сделало  составной объект EditNumberField() без состояния.

Чтобы добавить текстовое поле, вы можете повторно использовать тот же составной элемент EditNumberField(), но с другой меткой. Чтобы внести это изменение, вам нужно передать метку в качестве параметра, а не жестко запрограммировать ее внутри составной функции EditNumberField().

Сделайте составную функцию EditNumberField() многоразовой:

1. В  файле MainActivity.kt в   параметрах компонуемой функции EditNumberField() добавьте ресурс label типа Int:

@Composable  
fun EditNumberField(  
**label: Int,**    value: String,  
    onValueChanged: (String) -> Unit,  
    modifier: Modifier = Modifier  
)

**Примечание.** Лучше всего предоставлять [Modifier](https://developer.android.com/reference/kotlin/androidx/compose/ui/Modifier) параметр по умолчанию для всех составных функций, что повышает возможность повторного использования. Его следует добавить в качестве первого необязательного параметра после всех обязательных параметров.

1. В теле функции замените жестко запрограммированный идентификатор строкового ресурса параметром label:

@Composable  
fun EditNumberField(  
    //...  
) {  
     TextField(  
         //...  
         label = { Text(stringResource(label)) },  
         //...  
     )  
}

1. Чтобы обозначить, что параметр label, как ожидается, будет ссылкой на строковый ресурс, добавьте к параметру функции аннотацию @StringRes:

@Composable  
fun EditNumberField(  
    **@StringRes** label: Int,  
    value: String,  
    onValueChanged: (String) -> Unit,  
    modifier: Modifier = Modifier  
)

**Примечание.** Примером ссылки на строковый ресурс является файл R.string.bill\_amount.

1. Импортируйте следующее:

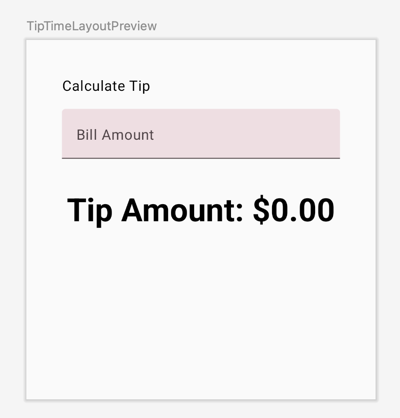
import androidx.annotation.StringRes

**Примечание.** Аннотация @StringRes - это типобезопасный способ использования строковых ресурсов. Это указывает на то, что передаваемое целое число является строковым ресурсом из файла values/strings.xml. Эти аннотации полезны разработчикам, работающим над вашим кодом, а также инструментам проверки кода, таким как [lint,](https://developer.android.com/studio/write/lint) в Android Studio. Вы узнаете больше о lint в будущей лаборатории кода.

1. В вызове функции TipTimeLayout() составной функции EditNumberField() установите label параметр строкового ресурса R.string.bill\_amount:

EditNumberField(  
**label = R.string.bill\_amount**,  
    value = amountInput,  
    onValueChanged = { amountInput = it },  
    modifier = Modifier.padding(bottom = 32.dp).fillMaxWidth()  
)

1. На панели **предварительного просмотра** не должно быть никаких визуальных изменений.

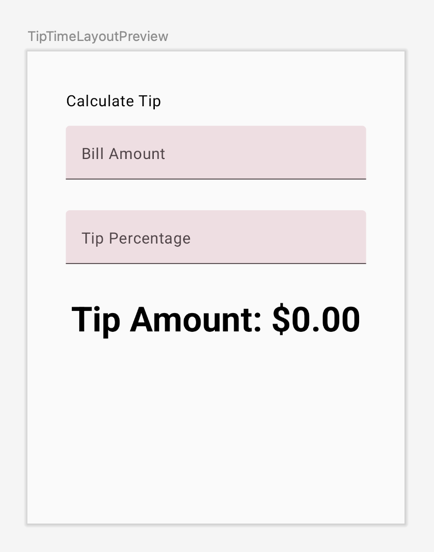


1. В составной функции TipTimeLayout() после вызова функции EditNumberField() добавьте еще одно текстовое поле для пользовательского процента чаевых. Вызовите составную функцию EditNumberField() со следующими параметрами:

EditNumberField(  
    label = R.string.how\_was\_the\_service,  
    value = "",  
    onValueChanged = { },  
    modifier = Modifier.padding(bottom = 32.dp).fillMaxWidth()  
)

Это добавляет еще одно текстовое поле для пользовательского процента чаевых.

1. В предварительном просмотре приложения теперь отображается текстовое поле **«Процент чаевых»**, как вы можете видеть на этом изображении:



1. В верхней части составной функции TipTimeLayout() добавьте свойство var, вызывающее переменную состояния tipInput добавленного текстового поля. Используйте для инициализации переменной mutableStateOf("") и окружения вызова функцией remember:

var tipInput by remember { mutableStateOf("") }

1. В новом вызове функции установите именованный параметр в переменную, а затем обновите переменную в лямбда-выражении: *EditNumberField*() valuetipInputtipInputonValueChanged

EditNumberField(

    label = R.string.how\_was\_the\_service,

    value = tipInput,

    onValueChanged = { tipInput = it },

    modifier = Modifier.padding(bottom = 32.dp).fillMaxWidth()

)

1. В функции TipTimeLayout() после определения переменной tipInput. Определите val имя tipPercent, которое преобразует переменную tipInput в тип Double. Используйте оператор Элвиса и верните 0, если значение равно null. Это значение может быть null, если текстовое поле пусто.

val tipPercent = tipInput.toDoubleOrNull() ?: 0.0

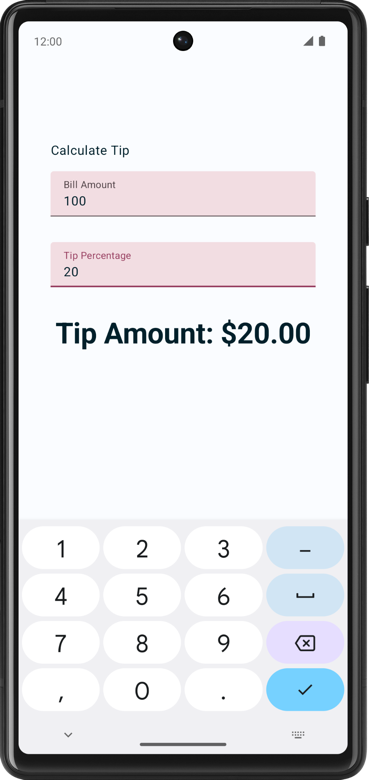
1. В функции TipTimeLayout() обновите calculateTip() вызов функции, передайте переменную tipPercent в качестве второго параметра:

val tip = calculateTip(amount, tipPercent)

Код функции TipTimeLayout()теперь должен выглядеть следующим образом:

@Composable  
fun TipTimeLayout() {  
    var amountInput by remember { mutableStateOf("") }  
    var tipInput by remember { mutableStateOf("") }  
    val amount = amountInput.toDoubleOrNull() ?: 0.0  
    val tipPercent = tipInput.toDoubleOrNull() ?: 0.0  
  
    val tip = calculateTip(amount, tipPercent)  
    Column(  
        modifier = Modifier.padding(40.dp),  
        horizontalAlignment = Alignment.CenterHorizontally,  
        verticalArrangement = Arrangement.Center  
    ) {  
        Text(  
            text = stringResource(R.string.calculate\_tip),  
            modifier = Modifier  
                .padding(bottom = 16.dp)  
                .align(alignment = Alignment.Start)  
        )  
        EditNumberField(  
            label = R.string.bill\_amount,  
            value = amountInput,  
            onValueChanged = { amountInput = it },  
            modifier = Modifier  
                .padding(bottom = 32.dp)  
                .fillMaxWidth()  
        )  
        EditNumberField(  
            label = R.string.how\_was\_the\_service,  
            value = tipInput,  
            onValueChanged = { tipInput = it },  
            modifier = Modifier  
                .padding(bottom = 32.dp)  
                .fillMaxWidth()  
        )  
        Text(  
            text = stringResource(R.string.tip\_amount, tip),  
            style = MaterialTheme.typography.displaySmall  
        )  
        Spacer(modifier = Modifier.height(150.dp))  
    }  
}

1. Запустите приложение на эмуляторе или устройстве, а затем введите сумму счета и процент чаевых. Правильно ли приложение рассчитывает сумму чаевых?



[5. Установите кнопку действия](https://developer.android.com/codelabs/basic-android-kotlin-compose-calculate-tip?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-calculate-tip#4)

В предыдущей лаборатории кода вы узнали, как использовать KeyboardOptions класс для установки типа клавиатуры. В этом разделе вы узнаете, как установить кнопку действия клавиатуры с тем же расширением KeyboardOptions. Кнопка действия клавиатуры — это кнопка на конце клавиатуры. Некоторые примеры вы можете увидеть в этой таблице:

|  |  |
| --- | --- |
| **Свойство** | **Кнопка действия на клавиатуре** |
| [ImeAction.Search](https://developer.android.com/reference/kotlin/androidx/compose/ui/text/input/ImeAction#Search()) Используется, когда пользователь хочет выполнить поиск. | Изображение представляет собой значок поиска для выполнения поиска. |
| [ImeAction.Send](https://developer.android.com/reference/kotlin/androidx/compose/ui/text/input/ImeAction#Send()) Используется, когда пользователь хочет отправить текст в поле ввода. | Изображение представляет собой значок «Отправить» для отправки текста в поле ввода. |
| [ImeAction.Go](https://developer.android.com/reference/kotlin/androidx/compose/ui/text/input/ImeAction#Go()) Используется, когда пользователь хочет перейти к цели текста во входных данных. | Изображение представляет собой значок «Перейти» для перехода к цели текста во входных данных. |

В этой задаче вы устанавливаете две разные кнопки действий для текстовых полей:

* **Кнопка «Следующее** действие» для текстового поля **«Сумма счета»**, которая указывает, что пользователь завершил текущий ввод и хочет перейти к следующему текстовому полю.
* Кнопка действия «Готово» для текстового **поля «Процент чаевых»**, которая указывает, что пользователь завершил ввод данных.

Примеры клавиатур с этими кнопками действий вы можете увидеть на этих изображениях:

|  |  |
| --- | --- |
|  |  |

Добавьте параметры клавиатуры:

1. При вызове EditNumberField() функции TextField() передайте KeyboardOptions конструктору imeAction именованный аргумент, которому присвоено ImeAction.Next значение. Используйте эту функцию [KeyboardOptions.Default.copy()](https://developer.android.com/reference/kotlin/androidx/compose/foundation/text/KeyboardOptions#copy(androidx.compose.ui.text.input.KeyboardCapitalization,kotlin.Boolean,androidx.compose.ui.text.input.KeyboardType,androidx.compose.ui.text.input.ImeAction)), чтобы убедиться, что вы используете другие параметры по умолчанию.

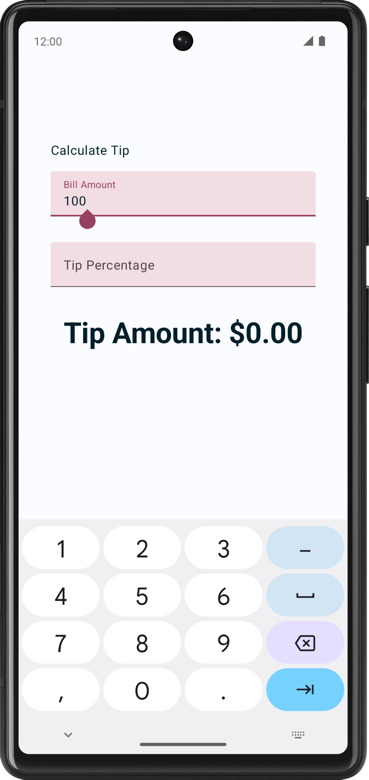
import androidx.compose.ui.text.input.ImeAction

@Composable

fun EditNumberField(

    //...  
) {  
    TextField(  
        //...  
        keyboardOptions = KeyboardOptions.Default.copy(  
            keyboardType = KeyboardType.Number,  
            imeAction = ImeAction.Next  
        )  
    )  
}

1. Запустите приложение на эмуляторе или устройстве. На клавиатуре теперь отображается кнопка **«Следующее** действие», как вы можете видеть на этом изображении:



Обратите внимание, что на клавиатуре отображается та же кнопка **«Следующее** действие», когда выбрано текстовое поле **«Процент чаевых» .** Однако вам нужны две разные кнопки действий для текстовых полей. Вы исправите эту проблему в ближайшее время.

1. Изучите функцию EditNumberField(). Параметр keyboardOptions в функции TextField() жестко запрограммирован. Чтобы создать различные кнопки действий для текстовых полей, вам необходимо передать объект KeyboardOptions в качестве аргумента, что вы и сделаете на следующем шаге.

// No need to copy, just examine the code.  
fun EditNumberField(  
    @StringRes label: Int,  
    value: String,  
    onValueChanged: (String) -> Unit,  
    modifier: Modifier = Modifier  
) {  
    TextField(  
        //...  
        keyboardOptions = KeyboardOptions.Default.copy(  
           keyboardType = KeyboardType.Number,  
           imeAction = ImeAction.Next  
        )  
    )  
}

1. В определении функции EditNumberField() добавьте параметр типа keyboardOptions type KeyboardOptions. В теле функции присвойте его именованному параметру TextField() функции keyboardOptions:

@Composable  
fun EditNumberField(  
    @StringRes label: Int,  
    keyboardOptions: KeyboardOptions,  
    // ...  
){  
    TextField(  
        //...  
        keyboardOptions = keyboardOptions  
    )  
}

1. В функции TipTimeLayout() обновите первый вызов функции EditNumberField(), передайте именованный параметр keyboardOptions в текстовое поле **«Сумма счета»** :

EditNumberField(  
    label = R.string.bill\_amount,  
    keyboardOptions = KeyboardOptions.Default.copy(  
        keyboardType = KeyboardType.Number,  
        imeAction = ImeAction.Next  
    ),  
    // ...  
)

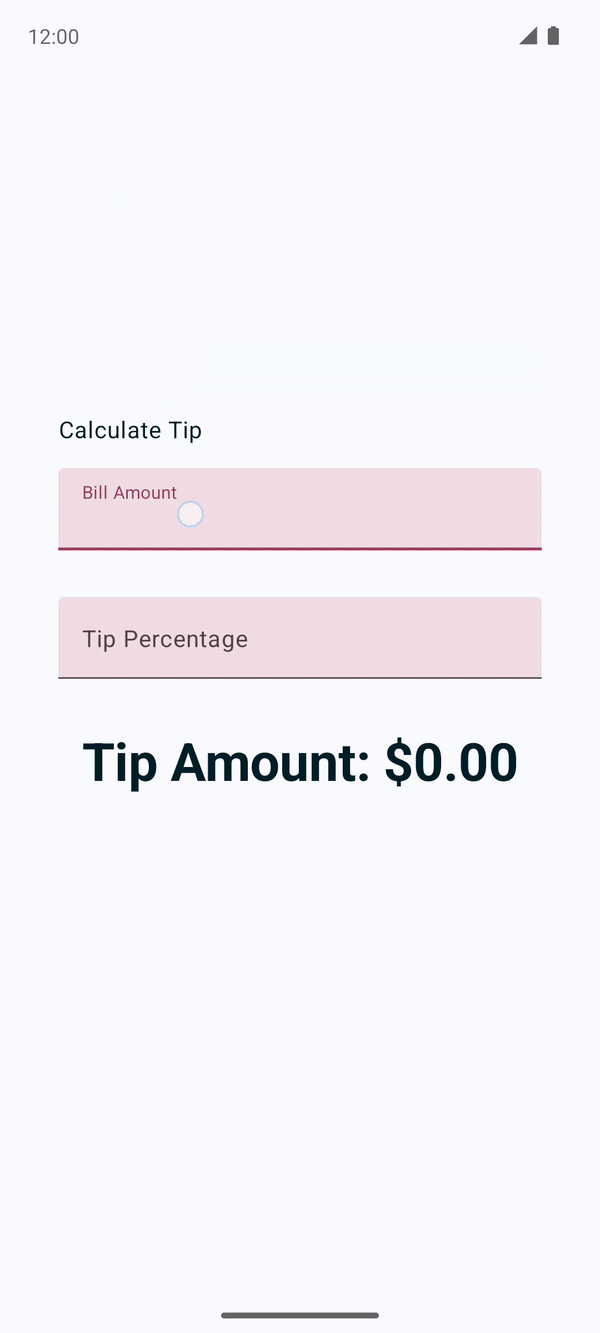
1. Во втором вызове функции EditNumberField() измените в текстовом поле **«Процент чаевых»** imeAction на ImeAction.Done. Ваша функция должна выглядеть как этот фрагмент кода:

EditNumberField(  
    label = R.string.how\_was\_the\_service,  
    keyboardOptions = KeyboardOptions.Default.copy(  
        keyboardType = KeyboardType.Number,  
        imeAction = ImeAction.Done  
    ),  
    // ...  
)

1. Запустите приложение. Он отображает кнопки действий **«Далее»** и **«Готово»** , как вы можете видеть на этих изображениях:

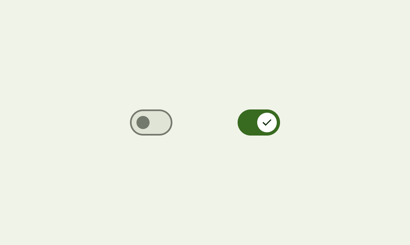
|  |  |
| --- | --- |
|  |  |

1. Введите любую сумму счета и нажмите кнопку **«Следующее** действие», затем введите любой процент чаевых и нажмите кнопку **«Готово** ». Клавиатура закрывается.

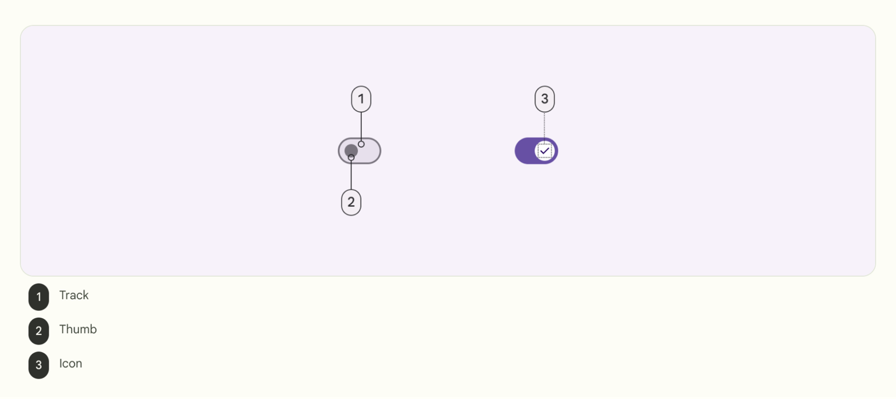


[6. Добавьте переключатель](https://developer.android.com/codelabs/basic-android-kotlin-compose-calculate-tip?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-calculate-tip#5)

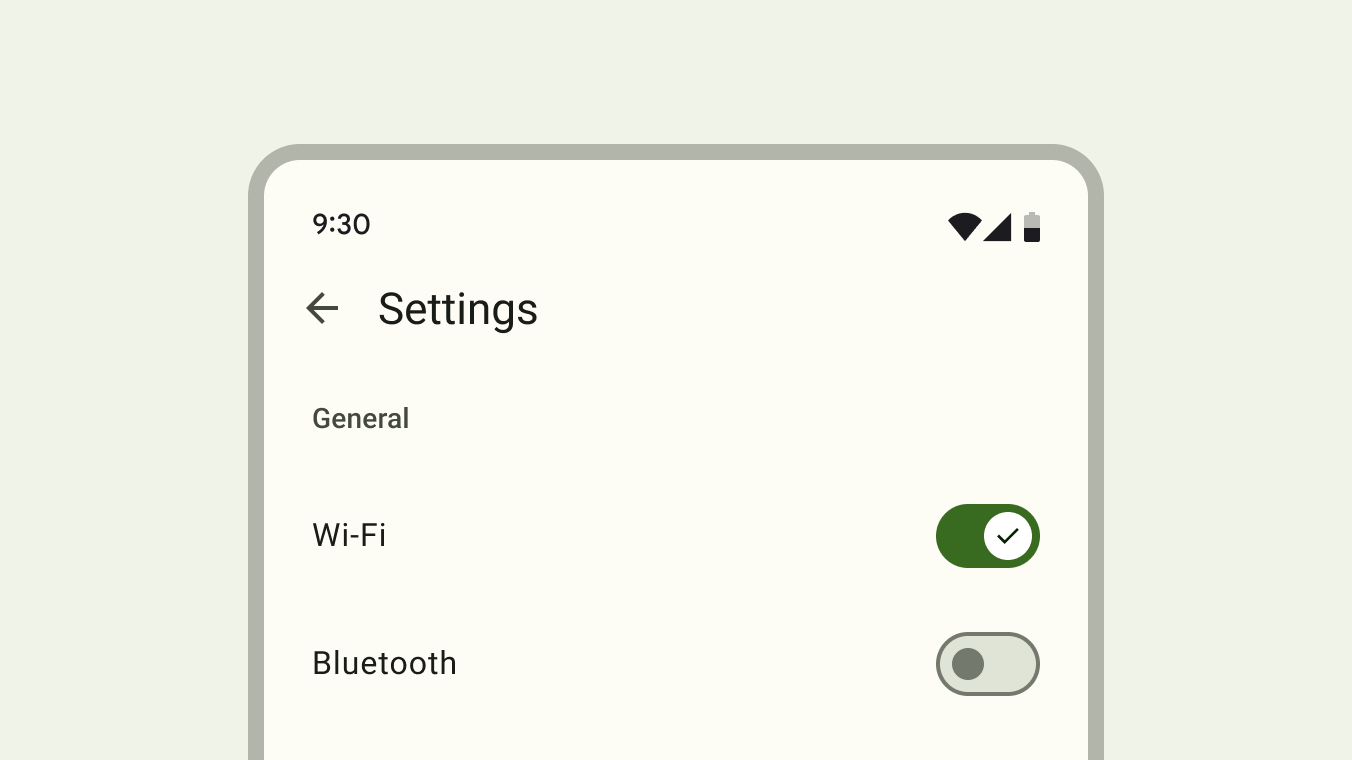
Переключатель включает или выключает состояние отдельного элемента.



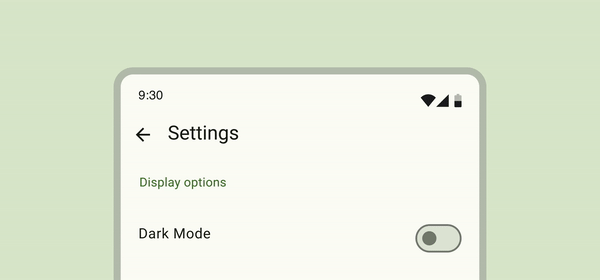
У переключателя есть два состояния, которые позволяют пользователю выбирать между двумя вариантами. Переключатель состоит из дорожки, ползунка и дополнительного значка, как вы можете видеть на этих изображениях:



Переключатель — это элемент управления выбором, который можно использовать для ввода решений или объявления предпочтений, например настроек, которые вы можете видеть на этом изображении:

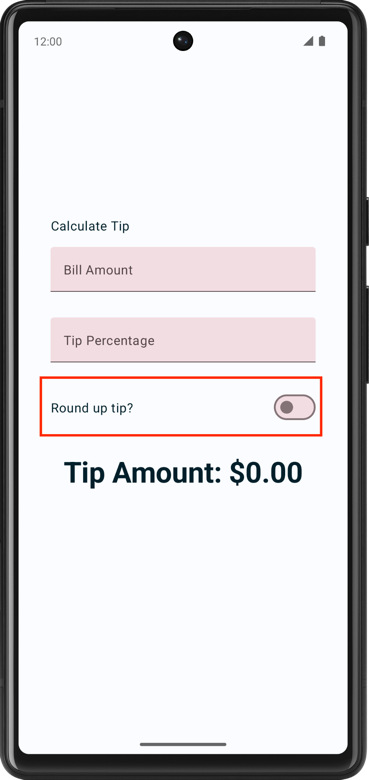


Пользователь может перетаскивать *большой палец* вперед и назад, чтобы выбрать выбранную опцию, или просто нажать переключатель для переключения. В этом GIF-изображении вы можете увидеть еще один пример переключения, в котором настройка визуальных параметров переключается на **темный режим** :



Дополнительные сведения о переключателях см. в документации [по переключателям](https://m3.material.io/components/switch/overview) .

Вы используете  компоновку Switch, чтобы пользователь мог выбрать, округлять ли чаевые до ближайшего целого числа, как вы можете видеть на этом изображении:



Добавьте строку для составных элементов Text и Switch:

1. После функции EditNumberField() добавьте составную функцию RoundTheTipRow(), а затем передайте значение по умолчанию в качестве аргументов Modifier, аналогичных функции EditNumberField():

@Composable  
fun RoundTheTipRow(modifier: Modifier = Modifier) {  
}

1. Реализуйте функцию RoundTheTipRow(), добавьте компонуемый макет Row со следующим modifier,  чтобы установить максимальную ширину дочерних элементов на экране, центрировать выравнивание и обеспечить размер 48dp:

Row(  
   modifier = modifier  
       .fillMaxWidth()  
       .size(48.dp),  
   verticalAlignment = Alignment.CenterVertically  
) {  
}

1. Импортируйте следующее:

import androidx.compose.foundation.layout.Row  
import androidx.compose.foundation.layout.size

1. В лямбда-блок компонуемого макета Row добавьте компонуемый объект Text, который использует   строковый ресурс R.string.round\_up\_tip для отображения  строки Round up tip?:

Text(text = stringResource(R.string.round\_up\_tip))

1. После составного объекта Text добавьте составной объект Switch и передайте именованный параметр checked, задав для него значение roundUp, а именованный параметр onCheckedChange задайте для него значение onRoundUpChanged.

Switch(  
    checked = roundUp,  
    onCheckedChange = onRoundUpChanged,  
)

Эта таблица содержит информацию об этих параметрах, которые являются теми же параметрами, которые вы определили для функции RoundTheTipRow():

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| checked | Проверен ли переключатель. Это состояние компонуемого Switch. |
| onCheckedChange | Обратный вызов, который будет вызываться при нажатии переключателя. |

1. Импортируйте следующее:

import androidx.compose.material3.Switch

1. В функцию RoundTheTipRow()добавьте параметр roundup типа Boolean и лямбда-функцию onRoundUpChanged, которая принимает Boolean и ничего не возвращает:

@Composable  
fun RoundTheTipRow(  
    roundUp: Boolean,  
    onRoundUpChanged: (Boolean) -> Unit,  
    modifier: Modifier = Modifier  
)

Это повышает состояние переключателя.

1. В составной элемент Switch добавьте modifier, чтобы выровнять составной элемент Switch по краю экрана:

       Switch(  
           modifier = modifier  
               .fillMaxWidth()  
               .wrapContentWidth(Alignment.End),  
           //...  
       )

1. Импортируйте следующее:

import androidx.compose.foundation.layout.wrapContentWidth

1. В функцию TipTimeLayout() добавьте переменную var для состояния компонуемого объекта Switch. Создайте var переменную с именем roundUp, установите mutableStateOf() и укажите в качестве начального значения false. Окружите remember { }.

fun TipTimeLayout() {  
    //...  
    **var roundUp by remember { mutableStateOf(false) }**  
  
    //...  
    Column(  
        ...  
    ) {  
      //...  
   }  
}

Это переменная для компонуемого состояния Switch, а значением по умолчанию будет значение false.

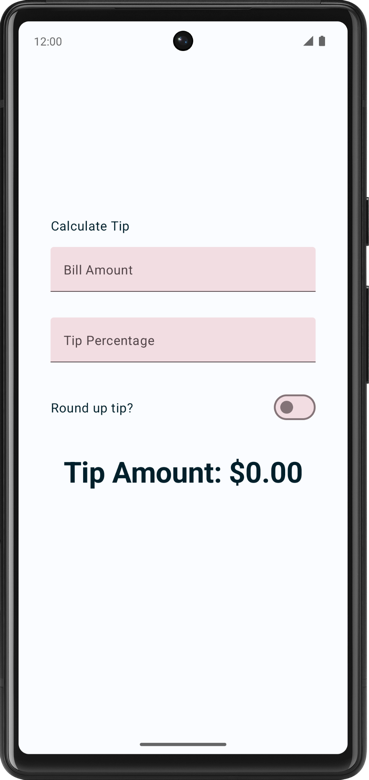
1. В блоке Column функций  TipTimeLayout() после текстового поля **«Процент чаевых»**. Вызовите  функцию RoundTheTipRow() со следующими аргументами: именованный параметр roundUp имеет значение roundUp и именованный параметр onRoundUpChanged имеет значение обратного вызова лямбда, который обновляет значение roundUp:

@Composable

fun TipTimeLayout() {  
    //...  
    Column(  
        ...  
    ) {  
        Text(  
            ...  
        )  
        Spacer(...)  
        EditNumberField(  
            ...  
        )  
        EditNumberField(  
            ...  
        )  
        RoundTheTipRow(  
             roundUp = roundUp,  
             onRoundUpChanged = { roundUp = it },  
             modifier = Modifier.padding(bottom = 32.dp)  
         )  
        Text(  
            ...  
        )  
    }  
}

Отобразится **подсказка "Округлить вверх"?**ряд.

1. Запустите приложение. Приложение отображает **подсказку «Округлить вверх»?**переключать.



1. Введите сумму счета и процент чаевых, а затем выберите **Округлить чаевые в большую сторону?**переключать. Сумма чаевых не округляется, поскольку вам все равно необходимо обновить функцию calculateTip(), что вы и сделаете в следующем разделе.

Обновите функцию calculateTip(), чтобы закруглить кончик.

Измените calculateTip() функцию, чтобы она принимала Boolean переменную для [округления](https://www.collinsdictionary.com/us/dictionary/english/round-up#:~:text=If%20you%20round%20an%20amount,or%20tip%20up%20to%2010%25.) кончика до ближайшего целого числа:

1. Чтобы округлить подсказку, функция calculateTip() должна знать состояние переключателя, который представляет собой Boolean. В функцию calculateTip() добавьте параметр roundUp типа Boolean:

private fun calculateTip(  
    amount: Double,  
    tipPercent: Double = 15.0,  
    roundUp: Boolean  
): String {   
    //...  
}

1. В функции calculateTip() перед оператором return добавьте условие if(), проверяющее значение roundUp. Если roundUp есть true, определите переменную tip и присвойте ей значение функции kotlin.math.*ceil*(), а затем передайте в функцию в качестве аргумента tip:

if (roundUp) {  
    tip = kotlin.math.ceil(tip)  
}

Android Studio показывает вам ошибку - val cannot be reassigned ошибка, вам нужно изменить tip переменную на var переменную:

var tip = tipPercent / 100 \* amount

**Примечание.** Функция [kotlin.math.ceil(x)](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.math/ceil.html) округляет заданное целое число в большую сторону. Например, оно округляет 10.65 до 11.00. Эта функция может принимать число Double или Float число.

Завершенная функция calculateTip() должна выглядеть следующим образом:

private fun calculateTip(amount: Double, tipPercent: Double = 15.0, roundUp: Boolean): String {  
    var tip = tipPercent / 100 \* amount  
    if (roundUp) {  
        tip = kotlin.math.ceil(tip)  
    }  
    return NumberFormat.getCurrencyInstance().format(tip)  
}

1. В функции TipTimeLayout() обновите вызов функции calculateTip(), а затем передайте параметр roundUp:

val tip = calculateTip(amount, tipPercent, roundUp)

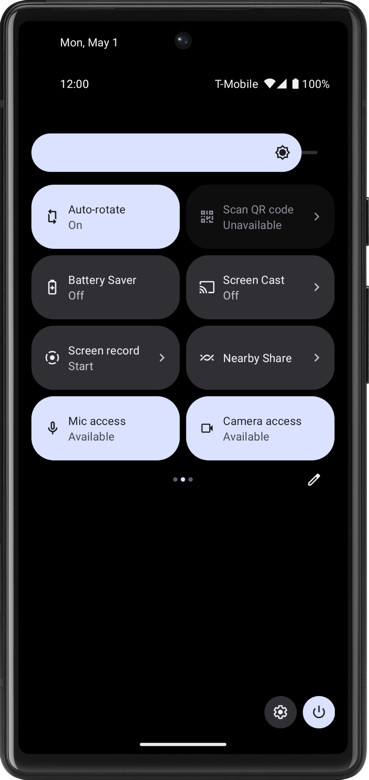
1. Запустите приложение. Теперь сумма чаевых округляется в большую сторону, как вы можете видеть на этих изображениях:

|  |  |
| --- | --- |
|  |  |

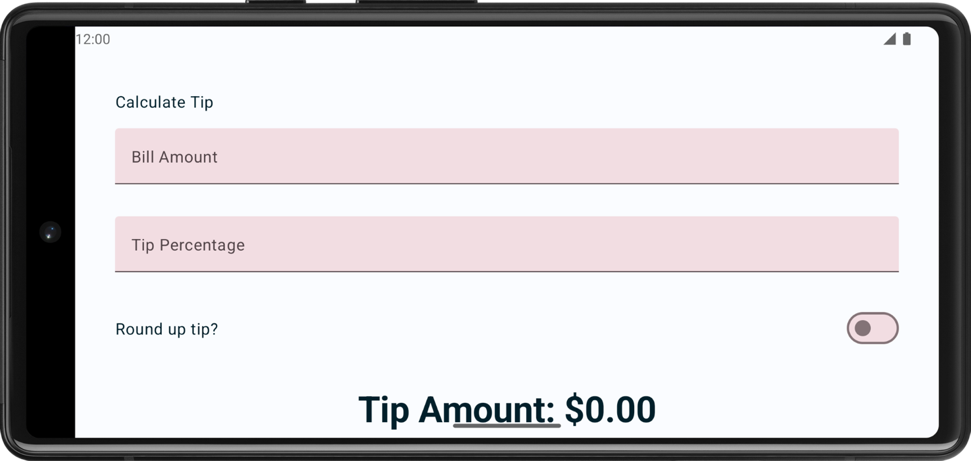
[7. Добавьте поддержку альбомной ориентации.](https://developer.android.com/codelabs/basic-android-kotlin-compose-calculate-tip?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-calculate-tip#6)

Устройства Android выпускаются в различных форм-факторах: телефоны, планшеты, складные устройства и устройства ChromeOS, которые имеют широкий диапазон размеров экрана. Ваше приложение должно поддерживать как книжную, так и альбомную ориентацию.

1. Проверьте свое приложение в ландшафтном режиме, включите автоматический поворот.



1. Поверните эмулятор или устройство влево. Обратите внимание, что вы не видите сумму чаевых. Чтобы решить эту проблему, вам понадобится вертикальная полоса прокрутки, которая поможет вам прокручивать экран приложения.



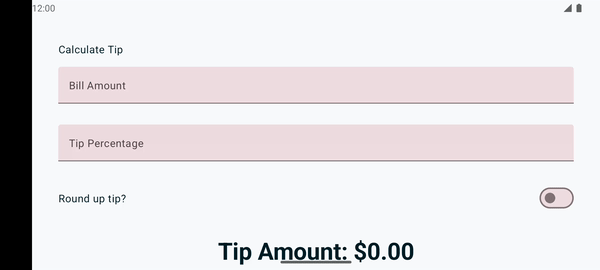
1. Добавьте .verticalScroll(rememberScrollState()) к модификатору, чтобы столбец мог прокручиваться вертикально. Создает rememberScrollState() и автоматически запоминает состояние прокрутки.

@Composable  
fun TipTimeLayout() {  
    // ...  
    Column(  
        modifier = Modifier  
            .padding(40.dp)  
            .verticalScroll(rememberScrollState()),  
        horizontalAlignment = Alignment.CenterHorizontally,  
        verticalArrangement = Arrangement.Center  
    ) {  
        //...  
    }  
}

1. Импортируйте следующее:

import androidx.compose.foundation.rememberScrollState  
import androidx.compose.foundation.verticalScroll

1. Запустите приложение еще раз. Попробуйте прокрутить в ландшафтном режиме!

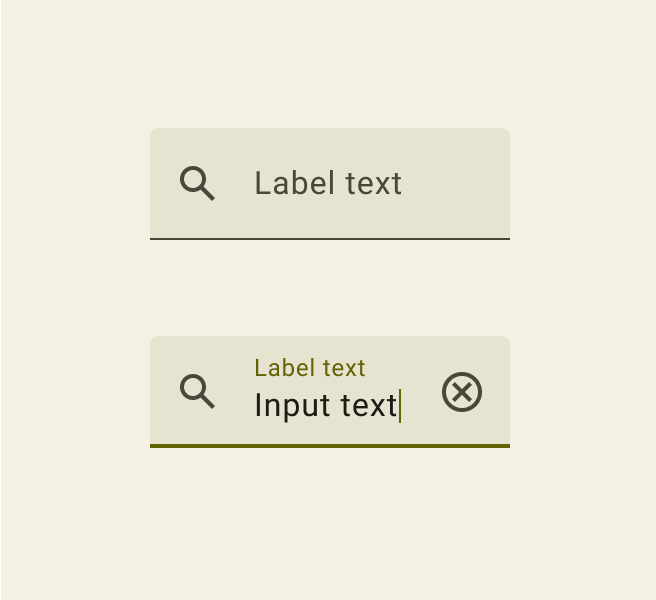


[8. Добавьте ведущий значок в текстовые поля (необязательно).](https://developer.android.com/codelabs/basic-android-kotlin-compose-calculate-tip?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-calculate-tip#7)

Значки могут сделать текстовое поле более привлекательным и предоставить дополнительную информацию о текстовом поле. Значки можно использовать для передачи информации о назначении текстового поля, например, какой тип данных ожидается или какой ввод требуется. Например, значок телефона рядом с текстовым полем может указывать на то, что пользователь должен ввести номер телефона.

Значки можно использовать для управления вводом пользователя, предоставляя визуальные подсказки о том, что ожидается. Например, значок календаря рядом с текстовым полем может указывать на то, что пользователь должен ввести дату.

Ниже приведен пример текстового поля со значком поиска, указывающим на необходимость ввода условия поиска.



Добавьте еще один параметр leadingIcon типа Int в составной объект EditNumberField(). Прокомментируйте его с помощью @DrawableRes.

@Composable  
fun EditNumberField(  
    @StringRes label: Int,  
    @DrawableRes leadingIcon: Int,  
    keyboardOptions: KeyboardOptions,  
    value: String,  
    onValueChanged: (String) -> Unit,  
    modifier: Modifier = Modifier  
)

1. Импортируйте следующее:

import androidx.annotation.DrawableRes  
import androidx.compose.material3.Icon

1. Добавьте ведущий значок в текстовое поле. Принимает составной объект leadingIcon, вы передаете следующий составной объект Icon.

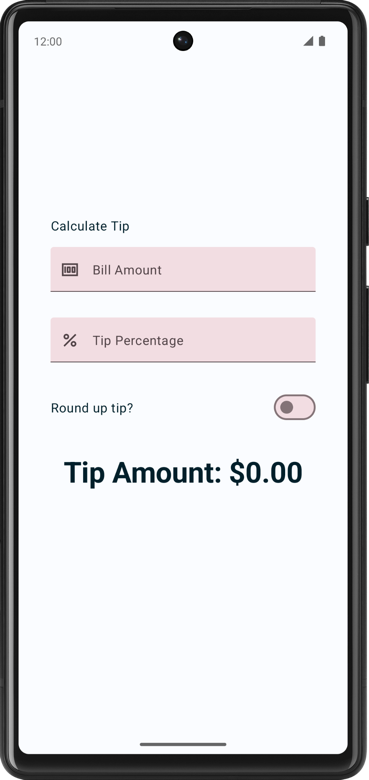
TextField(

    value = value,  
    leadingIcon = { Icon(painter = painterResource(id = leadingIcon), null) },  
    //...  
)

1. Передайте ведущий значок в текстовые поля. Для вашего удобства значки уже присутствуют в стартовом коде.

EditNumberField(  
    label = R.string.bill\_amount,  
    leadingIcon = R.drawable.money,  
    // Other arguments  
)  
EditNumberField(  
    label = R.string.how\_was\_the\_service,  
    leadingIcon = R.drawable.percent,  
    // Other arguments  
)

1. Запустите приложение.



Поздравляем! Теперь в вашем приложении есть возможность рассчитывать индивидуальные советы.