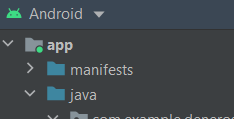
# Calcular a gorjeta

Escolha a visualização **Android** na lista suspensa, caso ela ainda não esteja selecionada.



# [Vinculação de visualizações](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator#2)

Por conveniência, o Android também oferece um recurso chamado [*vinculação de visualizações*](https://developer.android.com/topic/libraries/view-binding?hl=pt-br). Com um pouco mais de trabalho no início, a vinculação de visualizações faz com que seja muito mais fácil e rápido chamar métodos nas visualizações da IU. Será necessário ativar a vinculação de visualizações do app no Gradle e fazer algumas mudanças no código.

# Ativar a vinculação de visualizações

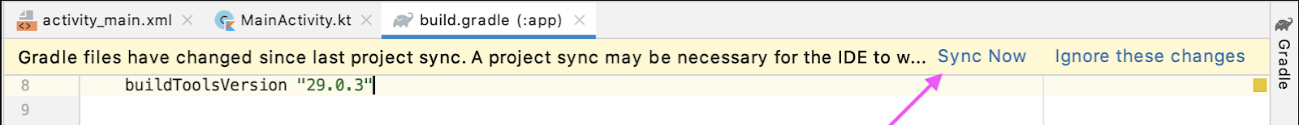
1. Abra o arquivo build.gradle do app (**Gradle Scripts > build.gradle (Module: Tip\_Time.app)**)
2. Na seção android, adicione as seguintes linhas:

buildFeatures {

viewBinding = true

}

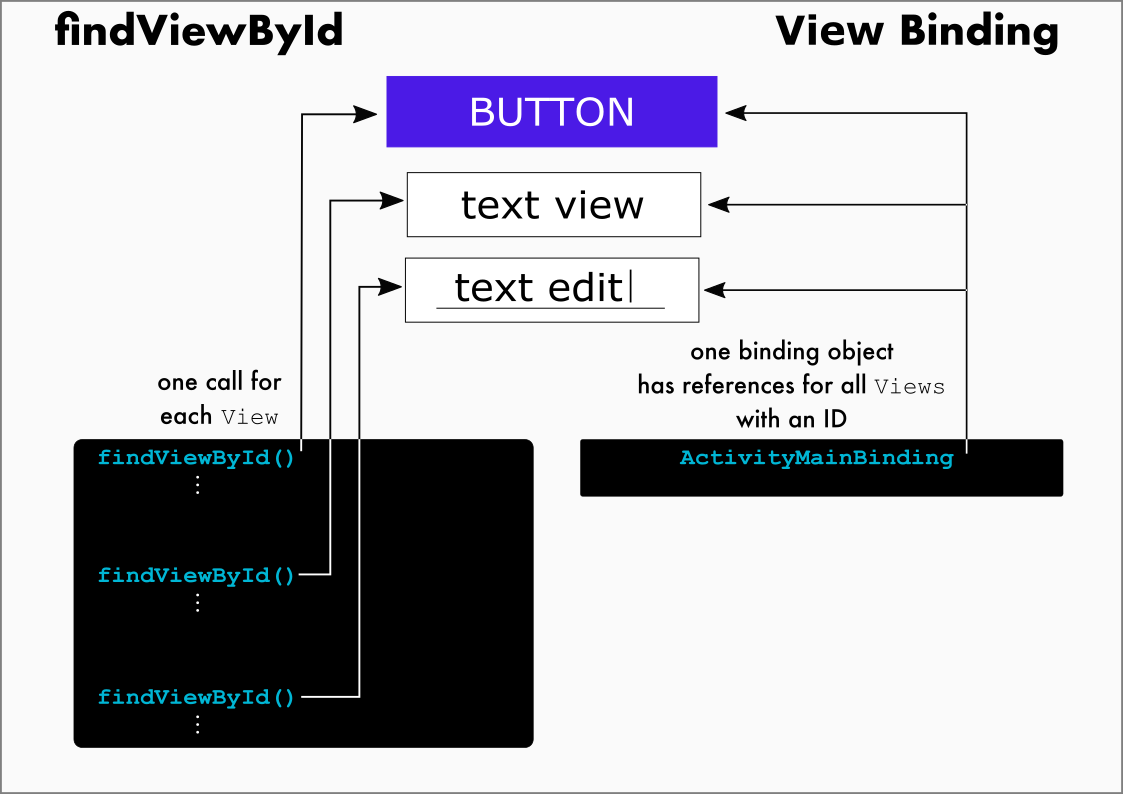
1. A mensagem **Gradle files have changed since last project sync**, que informa que os arquivos mudaram desde a última sincronização, deve aparecer.
2. Pressione **Sync Now**.



Após alguns instantes, você verá uma mensagem na parte inferior da janela do Android Studio, **Gradle sync finished**. Feche o arquivo build.gradle, se quiser.

Inicializar o objeto de vinculação

Nos codelabs anteriores, você viu o método onCreate() na classe MainActivity. Esse método é um dos primeiros a serem chamados quando o app é iniciado e a MainActivity é inicializada. Em vez de chamar findViewById() para cada View no seu app, você criará e inicializará um objeto de vinculação apenas uma vez.



1. Abra MainActivity.kt (**app > java > com.example.tiptime > MainActivity**).
2. Substitua todo o código da classe MainActivity por este código para configurar a MainActivity para usar a vinculação de visualização:

class MainActivity : AppCompatActivity() {  
  
    lateinit var binding: ActivityMainBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
    }  
}

1. Esta linha declara uma variável de nível superior na classe para o objeto de vinculação. Ela é definida nesse nível porque será usada em vários métodos da classe MainActivity.

lateinit var binding: ActivityMainBinding

A palavra-chave lateinit é nova. Ela é uma promessa de que seu código inicializará a variável antes de usá-la. Se você não fizer isso, o app falhará.

1. Essa linha inicializa o objeto binding que você usará para acessar as Views no layout activity\_main.xml.

binding = ActivityMainBinding.inflate(layoutInflater)

1. Defina a visualização de conteúdo da atividade. Em vez de transmitir o ID de recurso do layout, R.layout.activity\_main, esse código especifica a raiz da hierarquia de visualizações no app, binding.root.

setContentView(binding.root)

É possível que você se lembre da ideia de visualizações mães e filhas. A raiz se conecta a todas elas.

Agora, quando você precisar de uma referência a uma View no app, poderá acessá-la no objeto binding, em vez de chamar o método findViewById(). O objeto binding define automaticamente as referências para cada View do app que tenha um ID. Usar a vinculação de visualizações é tão mais conciso que muitas vezes você não precisará criar uma variável para armazenar a referência de uma View, bastará usá-la diretamente com o objeto de vinculação.

// Old way with findViewById()  
val myButton: Button = findViewById(R.id.my\_button)  
myButton.text = "A button"  
  
// Better way with view binding  
val myButton: Button = binding.myButton  
myButton.text = "A button"  
  
// Best way with view binding and no extra variable  
binding.myButton.text = "A button"

Não é demais?

**Observação**: o nome da classe de vinculação é gerado pela conversão do nome do arquivo XML em letras concatenadas e da adição da palavra "Binding" ao final. Da mesma forma, a referência para cada visualização é gerada removendo sublinhados e convertendo o nome da visualização para letras concatenadas. Por exemplo, **activity\_main.xml** torna-se **ActivityMainBinding**, e você pode acessar **@id/text\_view** como **binding.textView**.

[4. Calcular a gorjeta](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator#3)

O cálculo da gorjeta começará quando o usuário tocar no botão **Calculate**. Isso envolve verificar a IU para ver o custo do serviço e o percentual de gorjeta que o usuário quer deixar. Com essas informações, você calcula o valor total cobrado pelo serviço e exibe o valor da gorjeta.

Adicionar listeners de clique ao botão

O primeiro passo é adicionar um listener de clique para especificar o que o botão **Calculate** fará quando o usuário tocar nele.

1. Em MainActivity.kt no método onCreate(), após a chamada de setContentView(), defina um listener de clique no botão **Calculate** e faça com que ele chame o método calculateTip().

binding.calculateButton.setOnClickListener{ calculateTip() }

1. Ainda na classe MainActivity, mas fora do método onCreate(), adicione um método auxiliar com o nome calculateTip().

fun calculateTip() {  
  
}

É aqui que você adicionará o código para verificar a IU e calcular a gorjeta.

MainActivity.kt

class MainActivity : AppCompatActivity() {  
  
    lateinit var binding: ActivityMainBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
        binding.calculateButton.setOnClickListener{ calculateTip() }  
    }  
  
    fun calculateTip() {  
  
    }  
}

Acessar o custo do serviço

Para calcular a gorjeta, a primeira coisa de que você precisa é o custo do serviço. O texto é armazenado no EditText, mas você precisa que ele seja um número para poder usá-lo em cálculos. Você pode se lembrar do tipo Int de outros codelabs, mas um Int só pode conter números inteiros. Para usar um número decimal no app, use o tipo de dados Double em vez de Int. Leia mais sobre [tipos de dados numéricos em Kotlin](https://kotlinlang.org/docs/reference/basic-types.html#numbers) (em inglês) na documentação. O Kotlin fornece um método para converter uma String em um Double, conhecido como toDouble().

1. Primeiro, acesse o texto do custo do serviço. No método calculateTip(), acesse o atributo de texto EditText **Cost of Service**e atribua-o a uma variável com o nome stringInTextField. Lembre-se de que é possível acessar o elemento de IU usando o objeto binding e fazendo referência ao elemento da IU com base no nome do ID de recurso em letras concatenadas.

val stringInTextField = binding.costOfService.text

Observe o .text no final. A primeira parte, binding.costOfService, faz referência ao elemento da IU para o custo do serviço. Adicionar .text no final instrui o app a usar o resultado (um objeto EditText) e acessar a propriedade text dele. Isso é conhecido como *encadeamento* e é um padrão muito comum no Kotlin.

1. Depois, converta o texto em número decimal. Chame o método toDouble() em stringInTextField e armazene-o em uma variável com o nome cost.

val cost = stringInTextField.toDouble()

No entanto, isso não funcionará. toDouble() precisa ser chamado em uma String. Perceba que o atributo text de um EditText é do tipo Editable, porque representa um texto que pode ser modificado. Felizmente, é possível converter um Editable em uma String chamando o método toString() nele.

1. Chame toString() em binding.costOfService.text para convertê-lo em uma String:

val stringInTextField = binding.costOfService.text.toString()

Agora o método stringInTextField.toDouble() funcionará.

Neste momento, o método calculateTip() será semelhante a:

fun calculateTip() {  
    val stringInTextField = binding.costOfService.text.toString()  
    val cost = stringInTextField.toDouble()  
}

Acessar a porcentagem da gorjeta

Até o momento, você tem o custo do serviço. Agora você precisa da porcentagem de gorjeta, que o usuário selecionou em um RadioGroup de RadioButtons.

1. Em calculateTip(), acesse o atributo checkedRadioButtonId do RadioGroup tipOptions e atribua-o a uma variável com o nome selectedId.

val selectedId = binding.tipOptions.checkedRadioButtonId

Agora você sabe que uma das opções entre R.id.option\_twenty\_percent, R.id.option\_eighteen\_percent ou R.id.fifteen\_percent foi o RadioButton selecionado, mas precisa da porcentagem correspondente. É possível escrever uma série de instruções if/else, mas é muito mais fácil usar uma expressão when.

1. Adicione as linhas abaixo para acessar a porcentagem da gorjeta.

val tipPercentage = when (selectedId) {  
    R.id.option\_twenty\_percent -> 0.20  
    R.id.option\_eighteen\_percent -> 0.18  
    else -> 0.15  
}

Neste momento, o método calculateTip() será semelhante a:

fun calculateTip() {  
    val stringInTextField = binding.costOfService.text.toString()  
    val cost = stringInTextField.toDouble()  
    val selectedId = binding.tipOptions.checkedRadioButtonId  
    val tipPercentage = when (selectedId) {  
        R.id.option\_twenty\_percent -> 0.20  
        R.id.option\_eighteen\_percent -> 0.18  
        else -> 0.15  
    }  
}

Calcular a gorjeta e arredondar para cima

Agora que você tem o custo do serviço e o percentual de gorjeta, calcular a gorjeta é simples: multiplique o custo pela porcentagem da gorjeta. Gorjeta = custo do serviço \* porcentagem da gorjeta. Esse valor pode ser arredondado para cima.

1. No método calculateTip(), depois do outro código que você adicionou, multiplique a tipPercentage pelo cost e atribua o resultado a uma variável com o nome tip.

var tip = tipPercentage \* cost

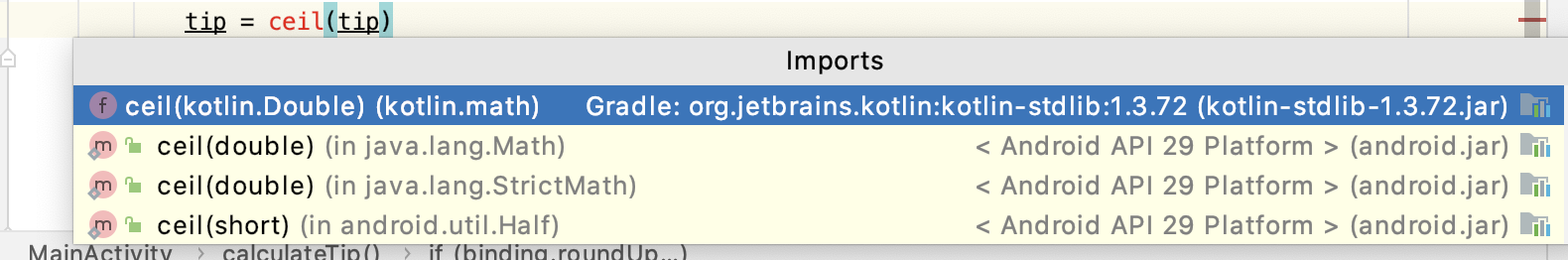
Observe que var é usado em vez de val. Isso ocorre porque pode ser necessário arredondar o valor se o usuário tiver selecionado essa opção. Assim, o valor poderá mudar.

    For a `Switch` element, you can check the `isChecked` attribute to see if the switch is "on".

1. Atribua o atributo isChecked do interruptor de arredondamento a uma variável com o nome roundUp.

val roundUp = binding.roundUpSwitch.isChecked

Arredondar significa ajustar um número decimal para cima ou para baixo para que seja o valor inteiro mais próximo. Porém, nesse caso, você só quer arredondar para cima ou encontrar o teto. Para isso, use a função ceil(). Há várias funções com esse nome, mas a que você quer usar está definida em kotlin.math. É possível adicionar uma instrução import, mas, neste caso, é mais simples apenas informar ao Android Studio a função que você quer usando kotlin.math.ceil().



Se você quisesse usar várias funções matemáticas, seria mais fácil adicionar uma instrução import.

1. Adicione uma instrução if que atribui o teto da gorjeta à variável tip se roundUp for verdadeiro.

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

Neste momento, o método calculateTip() será semelhante a:

fun calculateTip() {  
    val stringInTextField = binding.costOfService.text.toString()  
    val cost = stringInTextField.toDouble()  
    val selectedId = binding.tipOptions.checkedRadioButtonId  
    val tipPercentage = when (selectedId) {  
        R.id.option\_twenty\_percent -> 0.20  
        R.id.option\_eighteen\_percent -> 0.18  
        else -> 0.15  
    }  
    var tip = tipPercentage \* cost  
    val roundUp = binding.roundUpSwitch.isChecked  
    if (roundUp) {  
        tip = kotlin.math.ceil(tip)  
    }  
}

Formatar a gorjeta

Seu app está quase pronto. Você calculou a gorjeta, então só precisa formatá-la e exibi-la.

Como é de se esperar, o Kotlin fornece métodos para formatar diferentes tipos de número. Mas o valor da gorjeta é um pouco diferente: ele representa um valor de moeda. Cada país usa uma moeda diferente e tem regras diferentes para formatar números decimais. Por exemplo, em dólares americanos, 1234.56 seria formatado como $1,234.56, mas em euros, o formato seria €1.234,56. Felizmente, o framework do Android fornece métodos para formatar números como moedas, de modo que você não precisa conhecer todas as possibilidades. O sistema formata automaticamente a moeda com base no idioma e em outras configurações escolhidas pelo usuário no smartphone. Leia mais sobre [NumberFormat](https://developer.android.com/reference/java/text/NumberFormat?hl=pt-br) na documentação do desenvolvedor Android.

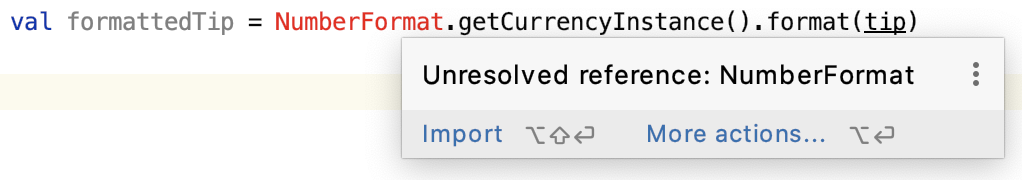
1. Em calculateTip(), após o outro código, chame NumberFormat.getCurrencyInstance().

NumberFormat.getCurrencyInstance()

Isso fornece um formatador numérico que pode ser usado para formatar os números como moedas.

1. Usando o formatador de números, encadeie uma chamada para o método format() com a tip e atribua o resultado a uma variável com o nome formattedTip.

val formattedTip = NumberFormat.getCurrencyInstance().format(tip)

1. Observe que NumberFormat é exibido em vermelho. Isso ocorre porque o Android Studio não consegue descobrir automaticamente qual versão de NumberFormat você quer usar.
2. Passe o cursor sobre NumberFormat e escolha **Import** no pop-up que será exibido. 
3. Na lista de possíveis importações, escolha **NumberFormat (java.text)**. O Android Studio adicionará uma instrução import na parte superior do arquivo MainActivity e o NumberFormat não estará mais vermelho.

Exibir a gorjeta

Agora é hora de exibir a gorjeta no elemento TextView do valor de gorjeta do app. Você pode atribuir formattedTip ao atributo text, mas seria bom marcar o que o valor representa. Nos EUA, em inglês, você pode exibir **Tip Amount: $12.34**, mas em outros idiomas o número pode precisar ser exibido no início ou no meio da string. O framework do Android fornece um mecanismo para isso chamado *parâmetros de string*. Assim, alguém que traduzir seu app poderá mudar onde o número aparece, se necessário.

1. Abra strings.xml (**app > res > values > strings.xml**)
2. Mude a string tip\_amount de Tip Amount para Tip Amount: %s.

<string name="tip\_amount">Tip Amount: %s</string>

O %s é onde a moeda formatada será inserida.

1. Agora, defina o texto do tipResult. De volta ao método calculateTip() em MainActivity.kt, chame getString(R.string.tip\_amount, formattedTip) e o atribua ao atributo text do resultado da gorjeta da TextView.

binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)

Neste momento, o método calculateTip() será semelhante a:

fun calculateTip() {  
    val stringInTextField = binding.costOfService.text.toString()  
    val cost = stringInTextField.toDouble()  
    val selectedId = binding.tipOptions.checkedRadioButtonId  
    val tipPercentage = when (selectedId) {  
        R.id.option\_twenty\_percent -> 0.20  
        R.id.option\_eighteen\_percent -> 0.18  
        else -> 0.15  
    }  
    var tip = tipPercentage \* cost  
    val roundUp = binding.roundUpSwitch.isChecked  
    if (roundUp) {  
        tip = kotlin.math.ceil(tip)  
    }  
    val formattedTip = NumberFormat.getCurrencyInstance().format(tip)  
    binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)  
}

Você está quase lá. Ao desenvolver seu app e exibir a visualização, é útil ter um marcador para essa TextView.

1. Abra activity\_main.xml (**app > res > layout > activity\_main.xml**)
2. Localize a TextView do tip\_result.
3. Remova a linha com o atributo android:text.

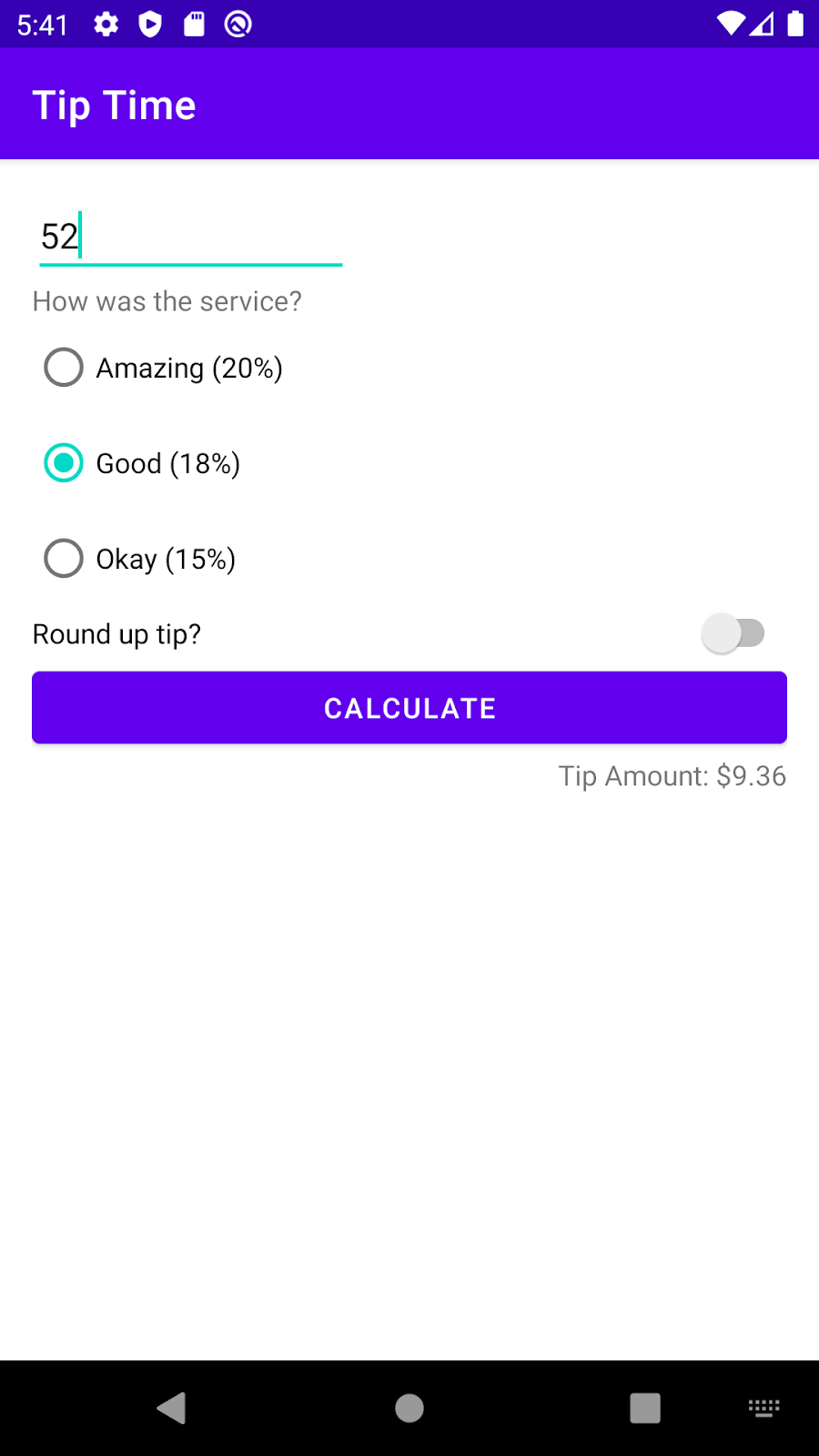
android:text="@string/tip\_amount"

1. Adicione uma linha para o atributo tools:text definido como Tip Amount: $10.

tools:text="Tip Amount: $10"

Como este é apenas um marcador, não é necessário extrair a string para um recurso. Ela não aparecerá quando você executar o app.

1. Observe que o texto das ferramentas é exibido no **Layout Editor**.
2. Execute o app. Insira um valor para o custo, selecione algumas opções e pressione o botão **Calculate**.



Parabéns, o app está funcionando! Se você não estiver recebendo o valor da gorjeta correto, volte para a etapa 1 desta seção e verifique se fez todas as mudanças necessárias no código.

[5. Testar e depurar](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator#4)

Você executou o app em várias etapas para garantir que ele faça o que você quer, mas agora é hora de fazer outros testes.

Por enquanto, pense em como as informações passam pelo app no método calculateTip() e o que poderia dar errado em cada etapa.

Por exemplo, o que aconteceria nessa linha:

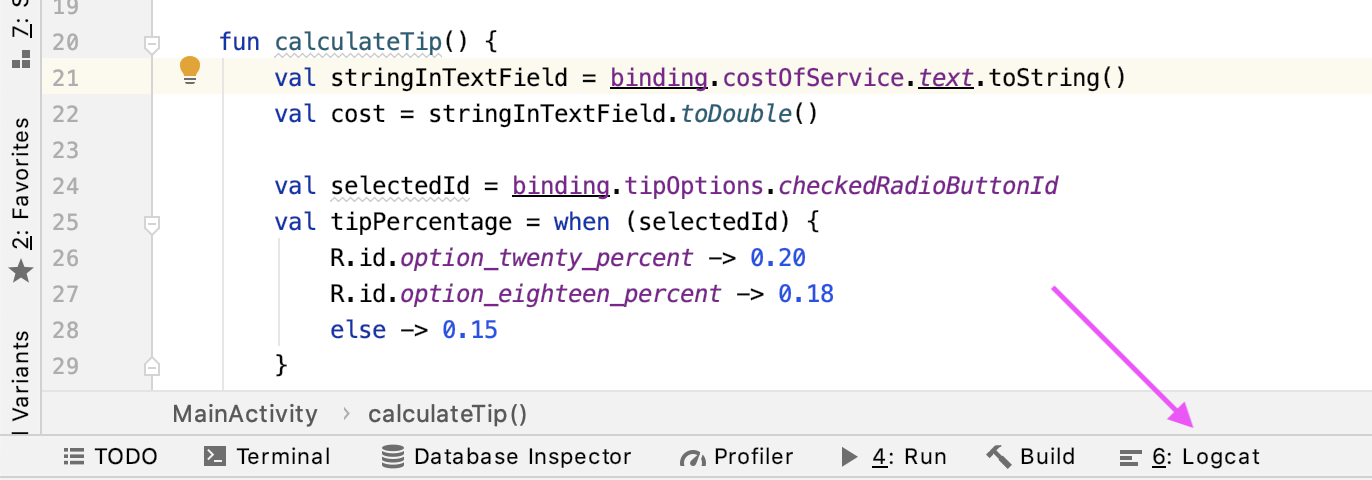
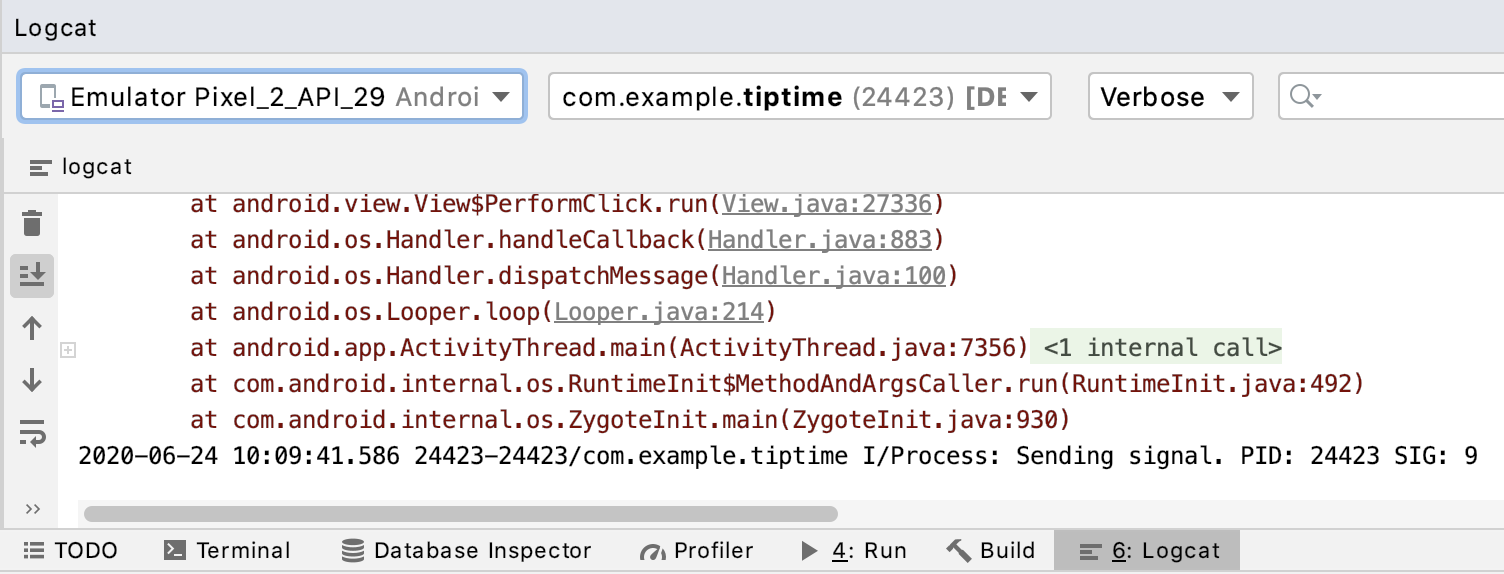
val cost = stringInTextField.toDouble()

se stringInTextField não representasse um número? O que aconteceria se o usuário não tivesse inserido texto e o campo stringInTextField estivesse vazio?

1. Execute o app no emulador, mas, em vez de usar **Run > Run 'app'**, use **Run > Debug 'app'**.
2. Ao tocar em **Calculate**, tente usar diferentes combinações de custo, valor da gorjeta, arredondando para cima ou não, e verifique se você recebeu o resultado esperado para cada caso.
3. Agora tente excluir todo o texto no campo **Cost of Service** e toque em **Calculate**. Ops! O programa falhou.

Depurar a falha

O primeiro passo para lidar com um bug é descobrir o que aconteceu. O Android Studio mantém um [registro](https://developer.android.com/studio/debug/am-logcat?hl=pt-br) do que está acontecendo no sistema, que você pode usar para descobrir o que deu errado.

1. Pressione o botão **Logcat** na parte inferior do Android Studio ou selecione **View > Tool Windows > Logcat** nos menus. 
2. A janela **Logcat** será exibida na parte inferior do Android Studio, preenchida com um texto estranho.  O texto é um *stack trace*, uma lista dos métodos que estavam sendo chamados quando a falha ocorreu.
3. Role para cima no **Logcat** até encontrar uma linha que inclua o texto FATAL EXCEPTION.

2020-06-24 10:09:41.564 24423-24423/com.example.tiptime E/AndroidRuntime: FATAL EXCEPTION: main  
    Process: com.example.tiptime, PID: 24423  
    java.lang.NumberFormatException: empty String  
        at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1842)  
        at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:110)  
        at java.lang.Double.parseDouble(Double.java:538)  
        at com.example.tiptime.MainActivity.calculateTip(MainActivity.kt:22)  
        at com.example.tiptime.MainActivity$onCreate$1.onClick(MainActivity.kt:17)

1. Deslize para baixo até encontrar a linha com NumberFormatException.

java.lang.NumberFormatException: empty String

À direita, está escrito empty String. O tipo da exceção informa que está relacionado a um formato de número e o restante mostra a base do problema: uma String vazia foi encontrada quando deveria ser uma String com um valor.

1. Continue descendo para ver as chamadas para parseDouble().
2. Abaixo dessas chamadas, encontre a linha com calculateTip. Observe que ela também inclui sua classe MainActivity.

at com.example.tiptime.MainActivity.calculateTip(MainActivity.kt:20)

1. Observe atentamente essa linha e você verá exatamente onde a chamada foi feita no código, linha 20 em MainActivity.kt. Se você digitou seu código de outra maneira, o número pode ser diferente. Essa linha converte a String em um Double e atribui o resultado à variável cost.

val cost = stringInTextField.toDouble()

1. Na documentação do Kotlin, procure o método toDouble() que funciona em uma String. O método tem o nome [String.toDouble()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/to-double.html) (link em inglês).
2. A página exibirá "Exceptions: NumberFormatException, se a string não for uma representação válida de um número".

Uma *exceção* é a maneira como o sistema informa que há um problema. Nesse caso, o problema é que o método toDouble() não conseguiu converter a String vazia em um Double. Mesmo que o EditText tenha inputType=numberDecimal, ainda é possível inserir alguns valores que toDouble() não consegue processar, como uma string vazia.

Saiba mais sobre o valor nulo

Chamar o método toDouble() em uma string vazia ou em uma string que não representa um número decimal válido não funciona. Felizmente, o Kotlin também oferece o método toDoubleOrNull() que gerencia esses problemas. Ele retorna um número decimal, se puder, ou retorna null se houver um problema.

*Null* é um valor especial que significa "sem valor". Ele é diferente de um Double com valor 0.0 ou uma String vazia com zero caractere, "". Null significa que não há valor, nenhum Double ou nenhuma String. Muitos métodos que esperam um valor podem não saber como processar null e serão interrompidos, o que significa que o app falhará. Por isso, o Kotlin tenta limitar onde null é usado. Você aprenderá mais sobre isso nas próximas lições.

Seu app pode verificar se null é retornado pelo método toDoubleOrNull() e fazer coisas de forma diferente nesse caso para que o app não falhe.

1. Em calculateTip(), mude a linha que declara a variável cost para chamar toDoubleOrNull(), em vez de toDouble().

val cost = stringInTextField.toDoubleOrNull()

1. Após essa linha, adicione uma instrução para verificar se cost é null e, em caso afirmativo, para retornar do método. A instrução return significa sair do método sem executar o restante das instruções. Se o método precisar retornar um valor, especifique-o com uma instrução return com uma expressão.

if (cost == null) {  
    return  
}

1. Execute o app novamente.
2. Sem adicionar texto ao campo **Cost of Service**, toque em **Calculate**. Desta vez, seu app não falhou. Bom trabalho! Você encontrou e corrigiu o bug.

Gerenciar outro caso

Nem todos os bugs causam falhas no app. Às vezes, os resultados podem ser confusos para o usuário.

Veja outro caso. O que acontecerá se o usuário:

1. inserir um valor válido para o custo;
2. tocar em **Calculate** para calcular a gorjeta;
3. excluir o custo;
4. tocar em **Calculate** novamente?

Na primeira vez, a gorjeta será calculada e exibida conforme o esperado. Na segunda vez, o método calculateTip() retornará antes devido à verificação que você acabou de adicionar, mas o app ainda mostrará o valor da gorjeta anterior. Isso pode confundir o usuário. Então, adicione algum código para limpar o valor da gorjeta se houver algum problema.

1. Para confirmar o problema, digite um custo válido, toque em **Calculate**. Em seguida, exclua o texto e toque em **Calculate** novamente. O primeiro valor da gorjeta ainda será exibido.
2. Na expressão if adicionada, antes da instrução return, adicione uma linha para definir o atributo text do tipResult como uma string vazia.

if (cost == null) {  
    binding.tipResult.text = ""  
    return  
}

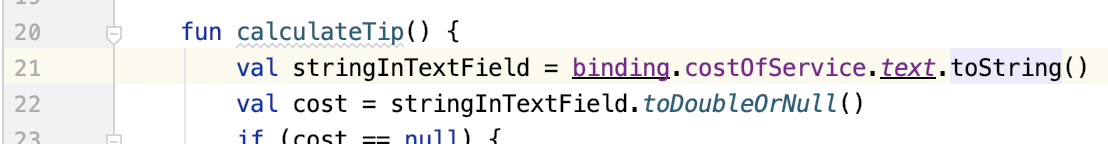
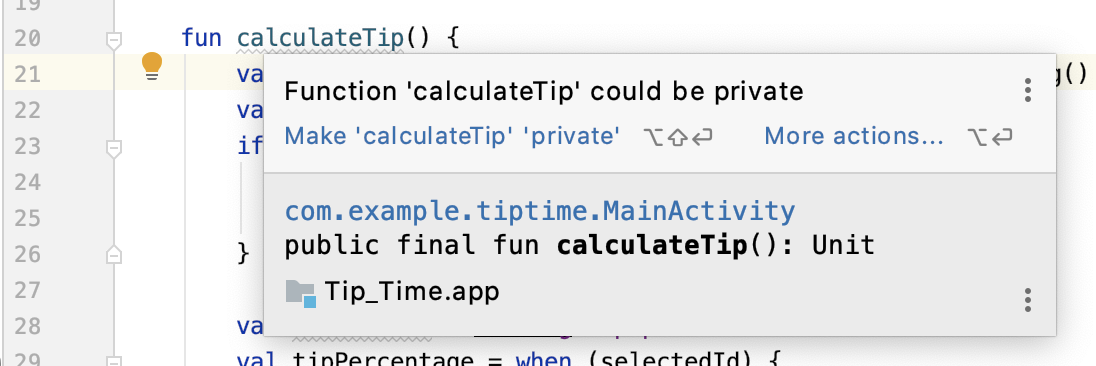
Isso limpará o valor da gorjeta antes de retornar do método calculateTip().

1. Execute o app novamente e repita o processo acima. O primeiro valor da gorjeta desaparecerá quando você tocar em **Calculate** pela segunda vez.

Parabéns! Você criou um app de calculadora de gorjetas funcional para Android e resolveu alguns possíveis problemas.

[6. Adotar boas práticas de programação](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator#5)

Sua calculadora de gorjetas funciona agora, mas você pode melhorar o código e facilitar a utilização no futuro adotando práticas de programação adequadas.

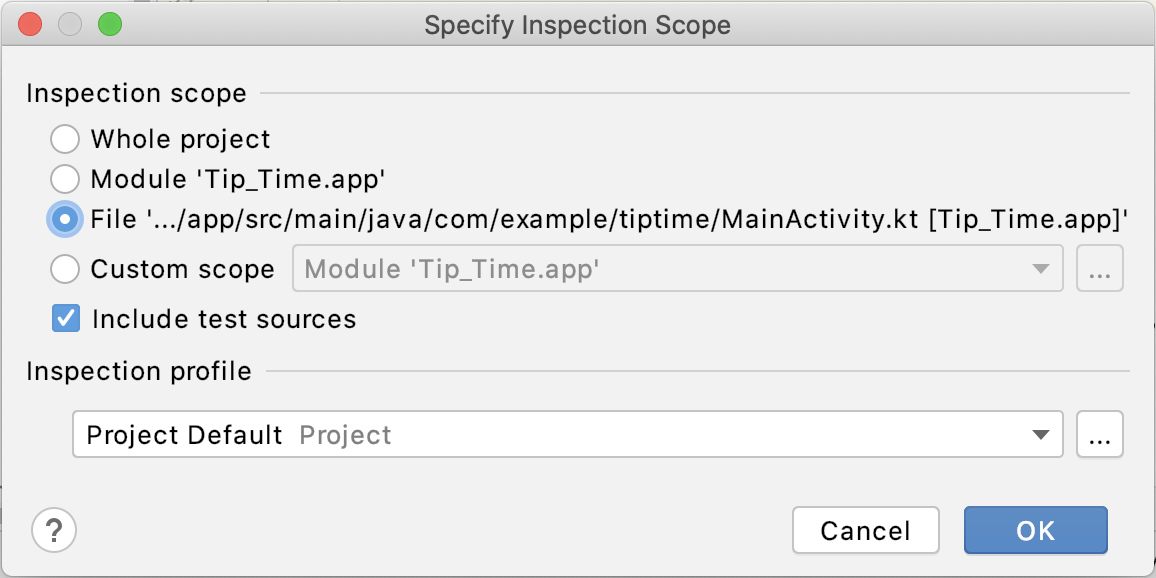
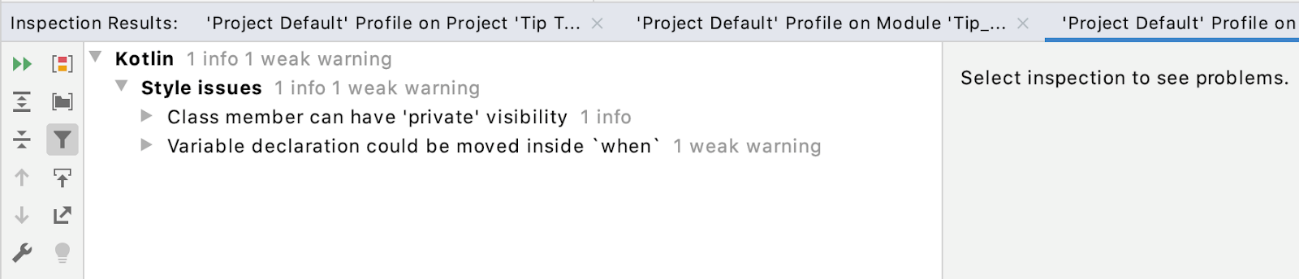
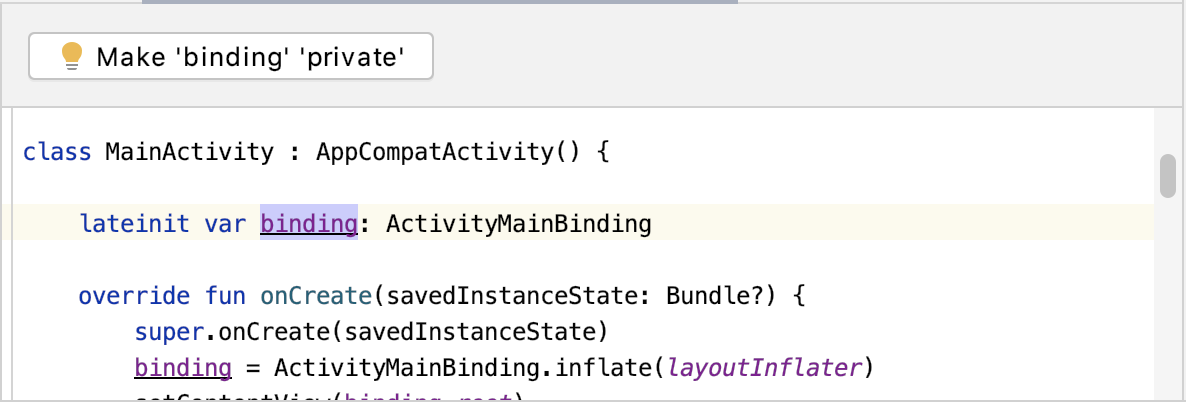
1. Abra MainActivity.kt (**app > java > com.example.tiptime > MainActivity**).
2. Veja o início do método calculateTip(). Ele está sublinhado com uma linha ondulada cinza. 
3. Passe o cursor sobre calculateTip(). Você verá a mensagem **Function 'calculateTip' could be private** com uma sugestão abaixo para que você faça com que a função seja particular: **Make 'calculateTip' 'private'**. 

Lembre-se de codelabs anteriores em que private significa que o método ou a variável ficam visíveis apenas para o código dessa classe, neste caso, a MainActivity. Não há motivo para que o código fora da MainActivity chame calculateTip(). Você pode tornar essa função private com segurança.

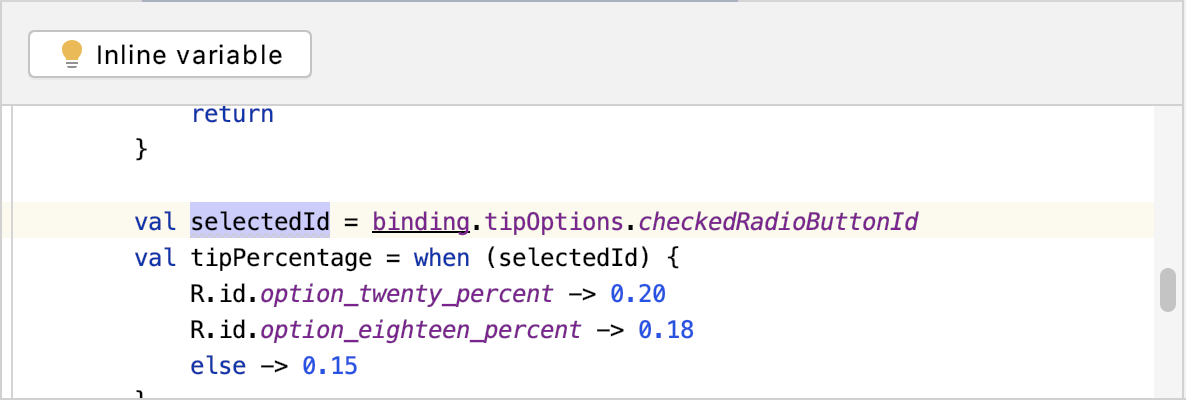
1. Escolha **Make ‘calculateTip' ‘private'** ou adicione a palavra-chave private antes de fun calculateTip(). A linha cinza em calculateTip() desaparecerá.

Inspecionar o código

A linha cinza é muito fina e pode passar despercebida. Você pode procurar essas linhas em todo o arquivo, mas há uma maneira mais simples de garantir que todas elas sejam encontradas.

1. Com o arquivo MainActivity.kt ainda aberto, escolha **Analyze > Inspect Code…** nos menus. Uma caixa de diálogo **Specify Inspection Scope** será exibida. 
2. Escolha a opção que começa com **File** e pressione **OK**. Isso limitará a inspeção apenas ao arquivo MainActivity.kt.
3. Uma janela com os **Inspection Results** será exibida na parte inferior.
4. Clique nos triângulos cinza ao lado de **Kotlin** e, ao lado de **Style issues**, até ver duas mensagens. A primeira diz **Class member can have 'private visibility'**. 
5. Clique nos triângulos cinza até ver a mensagem **Property ‘binding' could be private** e clique na mensagem. O Android Studio exibirá a parte do código da MainActivity e destacará a variável binding. 
6. Pressione o botão **Make ‘binding' ‘private'**. O Android Studio removerá o problema dos **Inspection Results**.
7. Se você verificar binding no seu código, verá que o Android Studio adicionou a palavra-chave private antes da declaração.

private lateinit var binding: ActivityMainBinding

1. Clique nos triângulos cinza nos resultados até ver a mensagem **Variable declaration could be inlined**. O Android Studio exibirá novamente a parte do código, mas, desta vez, destacará a variável selectedId. 
2. Se você analisar seu código, verá que o selectedId é usado apenas duas vezes: primeiro, na linha destacada em que o valor de tipOptions.checkedRadioButtonId é atribuído a ele e na próxima linha, na expressão when.
3. Pressione o botão **Inline variable**. O Android Studio substituirá o selectedId na expressão when pelo valor atribuído na linha anterior. Depois, ele removerá a linha anterior completamente, porque ela não é mais necessária.

val tipPercentage = when (binding.tipOptions.checkedRadioButtonId) {  
    R.id.option\_twenty\_percent -> 0.20  
    R.id.option\_eighteen\_percent -> 0.18  
    else -> 0.15  
}

Isso é bem legal! Seu código tem uma linha e uma variável a menos.

Como remover variáveis desnecessárias

O Android Studio não tem mais resultados da inspeção. No entanto, se você olhar seu código atentamente, verá um padrão semelhante ao que acabou de mudar: a variável roundUp é atribuída em uma linha, usada na seguinte e não é usada em nenhum outro lugar.

1. Copie a expressão à direita de = da linha em que roundUp é atribuído.

val roundUp = binding.roundUpSwitch.isChecked

1. Substitua roundUp na próxima linha pela expressão que você acabou de copiar, binding.roundUpSwitch.isChecked.

if (binding.roundUpSwitch.isChecked) {  
    tip = kotlin.math.ceil(tip)  
}

1. Exclua a linha com roundUp porque ela não é mais necessária.

Você fez o mesmo que o Android Studio ajudou a fazer com a variável selectedId. Novamente, o código tem uma linha e uma variável a menos. Essas mudanças são pequenas, mas ajudam a tornar seu código mais conciso e legível.

(Opcional) Eliminar o código repetitivo

Quando seu app estiver em execução corretamente, você poderá procurar outras oportunidades de limpar o código e torná-lo mais conciso. Por exemplo, quando você não insere um valor no custo do serviço, o app atualiza o tipResult para ser uma string vazia "". Quando houver um valor, você usará NumberFormat para formatá-lo. Essa funcionalidade pode ser aplicada em outro lugar no app, por exemplo, para exibir uma gorjeta de 0.0 em vez de uma string vazia.

Para reduzir a duplicação de código muito semelhante, extraia essas duas linhas de código para funções próprias. Essa função auxiliar pode receber como entrada um valor de gorjeta, como Double, formatá-la e atualizar a TextView do tipResult na tela.

1. Identifique o código duplicado no arquivo MainActivity.kt. Essas linhas de código podem ser usadas várias vezes na função calculateTip(), uma vez para o caso do valor 0.0 e outra para o caso geral.

val formattedTip = NumberFormat.getCurrencyInstance().format(0.0)  
binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)

1. Mova o código duplicado para a própria função. Uma mudança no código é usar uma gorjeta como parâmetro para que o código funcione em vários lugares.

private fun displayTip(tip : Double) {  
   val formattedTip = NumberFormat.getCurrencyInstance().format(tip)  
   binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)  
}

1. Atualize a função calculateTip() para usar a função auxiliar displayTip() e verifique o valor 0.0 também.

MainActivity.kt

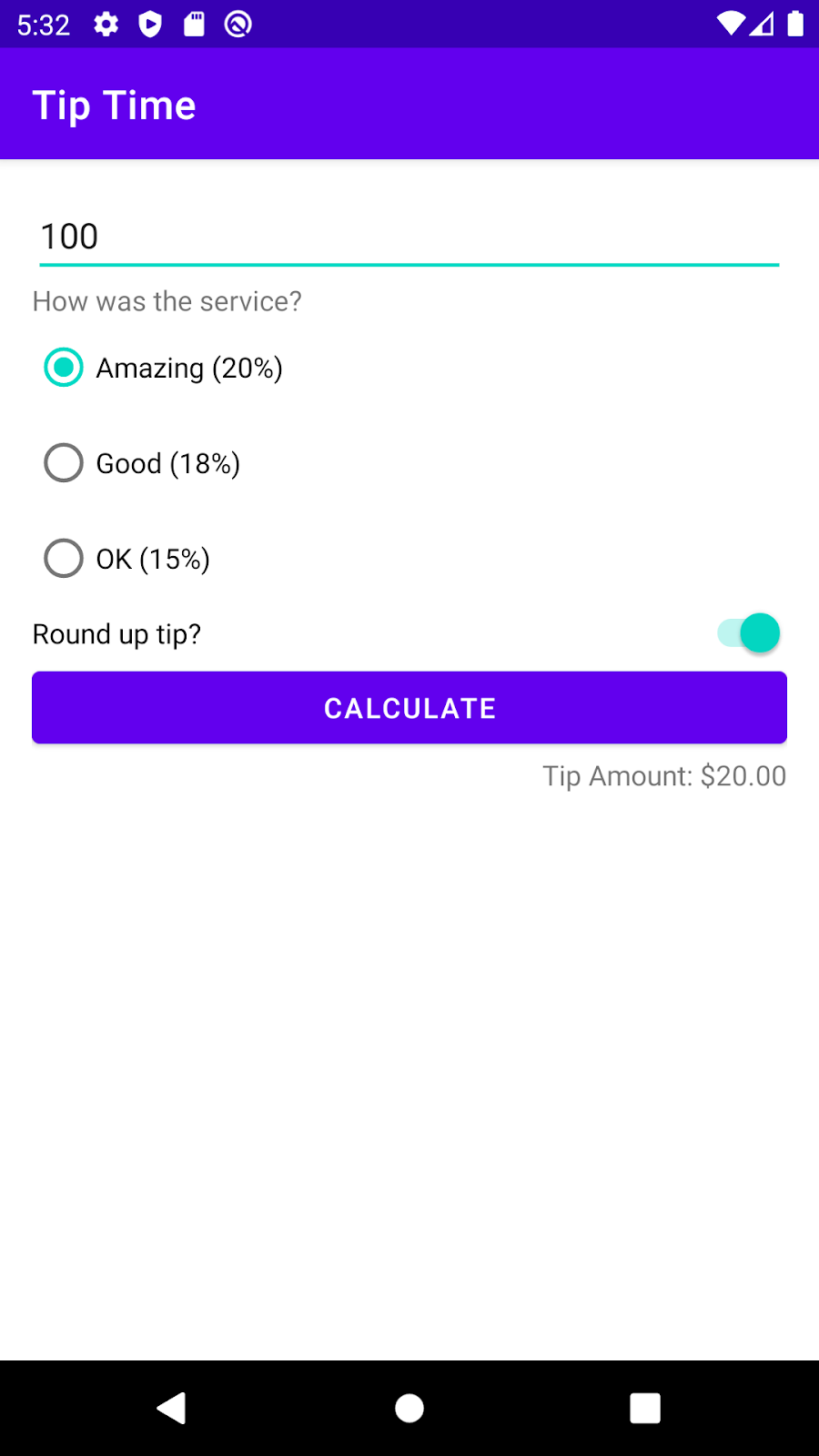
private fun calculateTip() {  
    ...  
  
        // If the cost is null or 0, then display 0 tip and exit this function early.  
        if (cost == null || cost == 0.0) {  
            displayTip(0.0)  
            return  
        }  
  
    ...  
    val roundUp = binding.roundUpSwitch.isChecked  
    if (roundUp) {  
        tip = kotlin.math.ceil(tip)  
    }  
  
    // Display the formatted tip value on screen  
    displayTip(tip)  
}

Observação

Mesmo que o app esteja funcionando agora, ele ainda não está pronto para produção. Você precisa fazer mais testes. E precisa adicionar algumas melhorias visuais e seguir as diretrizes do Material Design. Você também aprenderá a mudar o tema e o ícone do app nos codelabs a seguir.

## [7. Código da solução](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator#6)

Veja abaixo o código da solução para este codelab.



MainActivity.kt

(observação na primeira linha: substitua o nome do pacote, se o seu for diferente de com.example.tiptime)

package com.example.tiptime  
  
import android.os.Bundle  
import androidx.appcompat.app.AppCompatActivity  
import com.example.tiptime.databinding.ActivityMainBinding  
import java.text.NumberFormat  
  
class MainActivity : AppCompatActivity() {  
  
   private lateinit var binding: ActivityMainBinding  
  
   override fun onCreate(savedInstanceState: Bundle?) {  
       super.onCreate(savedInstanceState)  
  
       binding = ActivityMainBinding.inflate(layoutInflater)  
       setContentView(binding.root)  
  
       binding.calculateButton.setOnClickListener { calculateTip() }  
   }  
  
   private fun calculateTip() {  
       val stringInTextField = binding.costOfService.text.toString()  
       val cost = stringInTextField.toDoubleOrNull()  
       if (cost == null) {  
           binding.tipResult.text = ""  
           return  
       }  
  
       val tipPercentage = when (binding.tipOptions.checkedRadioButtonId) {  
           R.id.option\_twenty\_percent -> 0.20  
           R.id.option\_eighteen\_percent -> 0.18  
           else -> 0.15  
       }  
  
       var tip = tipPercentage \* cost  
       if (binding.roundUpSwitch.isChecked) {  
           tip = kotlin.math.ceil(tip)  
       }  
  
       val formattedTip = NumberFormat.getCurrencyInstance().format(tip)  
       binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)  
   }  
}

Modifique strings.xml

<string name="tip\_amount">Tip Amount: %s</string>

Modifique activity\_main.xml

...  
  
<TextView  
   android:id="@+id/tip\_result"  
   ...  
   tools:text="Tip Amount: $10" />  
  
...

Modifique o build.gradle do módulo do app

android {  
    ...  
  
    buildFeatures {  
        viewBinding = true  
    }  
    ...  
}

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator)

[8. Resumo](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator#7)

* A vinculação de visualizações permite criar códigos com mais facilidade, para que interajam com os elementos da IU do seu app.
* O tipo de dados Double no Kotlin pode armazenar um número decimal.
* Use o atributo checkRadioButtonId de um RadioGroup para descobrir qual RadioButton foi selecionado.
* Use NumberFormat.getCurrencyInstance() para que um formatador seja usado para formatar números como moeda.
* Use parâmetros de string como %s para criar strings dinâmicas que ainda podem ser facilmente traduzidas para outros idiomas.
* Testes são importantes.
* Você pode usar o **Logcat** no Android Studio para resolver problemas como falhas do app.
* Um stack trace mostra uma lista de métodos que foram chamados. Isso pode ser útil se o código gerar uma exceção.
* Exceções indicam um problema que o código não esperava.
* Null significa "sem valor".
* Nem todos os códigos podem manipular valores null, portanto, tenha cuidado ao usá-los.
* Use **Analyze > Inspect Code** para ver sugestões que podem melhorar o código.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator)

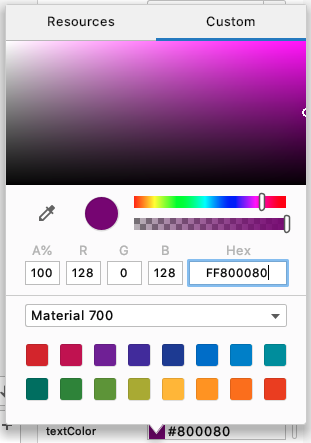
## [2. Design e cor](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#1)

## Material Design

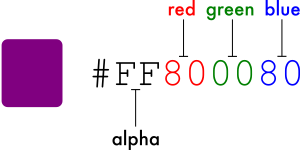
O [Material Design](https://material.io/design/introduction) (link em inglês) é inspirado no mundo físico e nas texturas dele, incluindo a forma como os objetos refletem a luz e projetam sombras. Ele apresenta diretrizes sobre como criar a IU do app de uma forma legível, atraente e consistente. Com o [Material Theming](https://material.io/design/material-theming/overview.html#material-theming) (link em inglês), você pode adaptar o Material Design ao app seguindo as orientações de personalização de cores, tipografia e formas. O Material Design vem com um tema de referência, que pode ser usado sem mudanças. Você pode personalizá-lo o quanto quiser para que o Material Design se adapte ao seu app.

## Um pouco sobre cores

As cores estão por toda parte ao nosso redor, no mundo real e no mundo digital. A primeira coisa que você precisa saber é que nem todos os usuários veem as cores da mesma maneira. Então, é importante escolher cores para o app de modo que ele possa ser usado facilmente pelos usuários. Escolher cores com contraste suficiente é apenas uma etapa para [criar apps mais acessíveis](https://developer.android.com/guide/topics/ui/accessibility?hl=pt-br).



Uma [Cor](https://developer.android.com/reference/kotlin/android/graphics/Color?hl=pt-br) pode ser representada por três números hexadecimais, #00-#FF (0-255), representando os componentes vermelho, verde e azul (RGB) dessa cor. Quanto maior o número, maior a proporção da cor específica.



Uma cor também pode ser definida incluindo um valor alfa #00-#FF, que representa a transparência (#00 = 0% = totalmente transparente, #FF = 100% = totalmente opaco). Quando incluído, o valor alfa é o primeiro de quatro números hexadecimais (ARGB). Se não houver um valor alfa, ele será considerado #FF = 100% opaco.

No entanto, não é necessário gerar os números hexadecimais manualmente. Há ferramentas disponíveis para ajudar você a escolher as cores que gerarão os números.

Alguns exemplos que você pode ter visto no arquivo colors.xml do seu app Android incluem 100% preto (R=#00, G=#00, B=#00) e 100% branco (R=#FF, G=#FF, B=#FF):

<color name="black">#FF000000</color>  
<color name="white">#FFFFFFFF</color>

[3. Temas](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#2)

Um *estilo* pode especificar atributos para uma View, como cor e tamanho da fonte, cor do segundo plano e muito mais.

O *tema* é uma coleção de estilos aplicados a um app, atividade ou hierarquia de visualização inteiros, não apenas a uma View individual. Quando você aplica um tema a um app, atividade, visualização individual ou em grupo, o tema é aplicado a esse elemento e a todos os filhos deles. Os temas também podem aplicar estilos a elementos que não são de visualização, como a barra de status e o segundo plano da janela.

Criar um projeto Empty Activity

Se você estiver usando seu próprio app, pule esta seção. Se precisar de um app para usar como exemplo, siga estas etapas para criar um app Empty Activity.

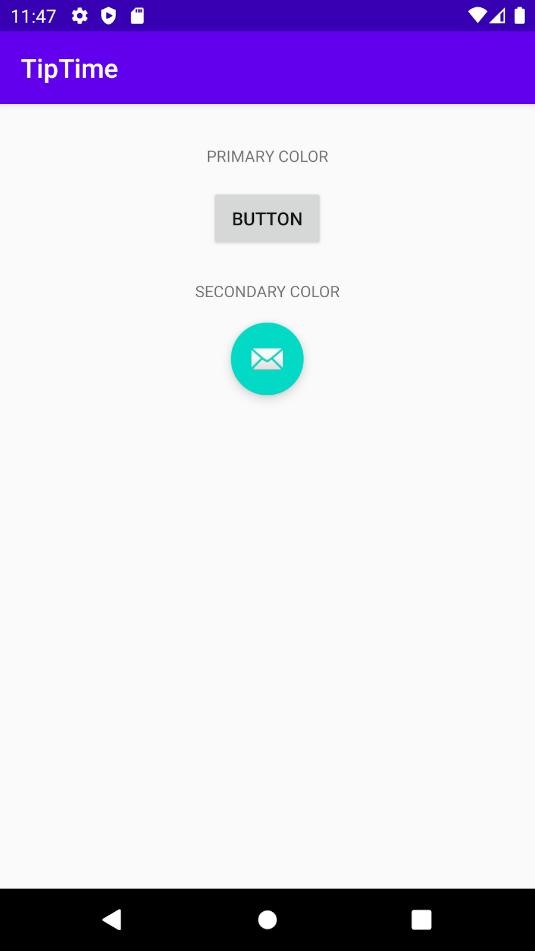
1. Abra o Android Studio.
2. Crie um novo projeto Kotlin usando o modelo **Empty Activity**.
3. Use o nome "TipTime". Como alternativa, você pode manter o nome padrão, "My Application", se não estiver participando de outros codelabs.
4. Selecione uma API de nível 19 no mínimo (KitKat).
5. Depois que o Android Studio concluir a criação do app, abra o arquivo activity\_main.xml (**app > res > layout > activity\_main.xml**).
6. Se necessário, alterne para a visualização **Code**.
7. Substitua todo o texto por este XML:

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:layout\_margin="16dp"  
    android:orientation="vertical"  
    tools:context=".MainActivity">  
  
    <TextView  
        android:layout\_width="wrap\_content"  
        android:layout\_height="48dp"  
        android:layout\_gravity="center\_horizontal"  
        android:gravity="center\_vertical"  
        android:text="@string/primary\_color"  
        android:textAllCaps="true"  
        android:textSize="12sp" />  
  
    <Button  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="center\_horizontal"  
        android:text="@string/button" />  
  
    <TextView  
        android:layout\_width="wrap\_content"  
        android:layout\_height="48dp"  
        android:layout\_gravity="center\_horizontal"  
        android:layout\_marginTop="8dp"  
        android:gravity="center\_vertical"  
        android:text="@string/secondary\_color"  
        android:textAllCaps="true"  
        android:textSize="12sp" />  
  
    <com.google.android.material.floatingactionbutton.FloatingActionButton  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="center\_horizontal"  
        android:contentDescription="@string/email\_icon"  
        app:srcCompat="@android:drawable/ic\_dialog\_email" />  
  
</LinearLayout>

1. Abra o arquivo strings.xml (**app > res > values > strings.xml**).
2. Substitua todo o texto por este XML:

<resources>  
    <string name="app\_name">TipTime</string>  
    <string name="primary\_color">Primary color</string>  
    <string name="button">Button</string>  
    <string name="secondary\_color">Secondary color</string>  
    <string name="email\_icon">email icon</string>  
</resources>

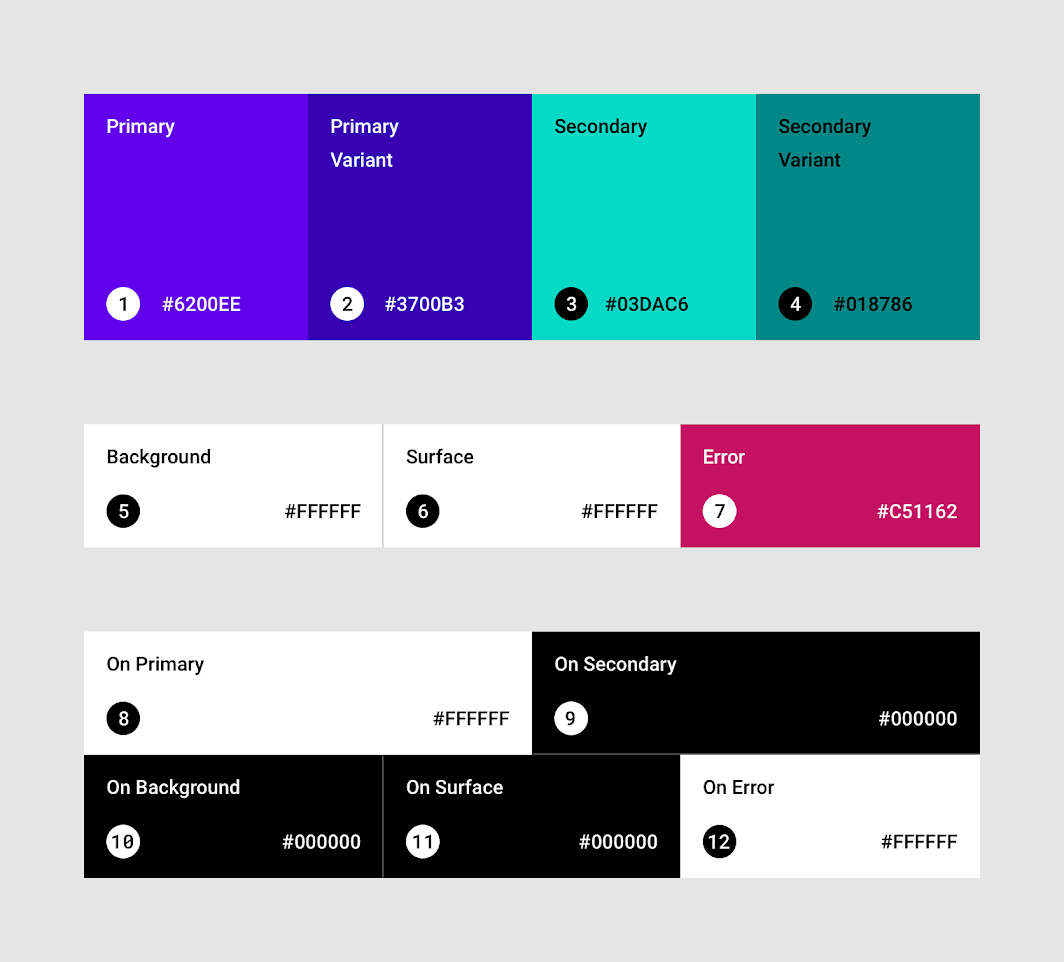
1. Execute o app. Ele ficará parecido com a captura de tela abaixo.



O app inclui uma TextView e um Button para permitir que você veja as opções de cor em um app Android real. A cor do botão mudará para a cor principal do tema nas próximas etapas.

Saiba mais sobre cores de tema

Cada parte da IU dos apps Android usa cores diferentes. Para ajudar você a usar as cores de forma significativa no app e aplicá-las consistentemente, o sistema de tema agrupa as cores em [12 atributos nomeados](https://material.io/design/material-theming/implementing-your-theme.html#color) relacionados à cor que será usada pelo texto, pelos ícones e muito mais. Seu tema não precisa especificar todos eles. Você escolherá as cores primárias e secundárias, além das cores do texto e dos ícones desenhados com essas cores.



As cores "ativadas" são usadas para textos e ícones desenhados nas diferentes superfícies.

|  |  |  |
| --- | --- | --- |
| **MULHERES** | **Nome** | **Atributo do tema** |
| 1 | Primário | colorPrimary |
| 2 | Variante Primária | colorPrimaryVariant |
| 3 | Secundário | colorSecondary |
| 4 | Variante Secundária | colorSecondaryVariant |
| 5 | Fundo | colorBackground |
| 6 | Superfície | colorSurface |
| 7 | Erro | colorError |
| 8 | No Primário | colorOnPrimary |
| 9 | No Secundário | colorOnSecondary |
| 10 | No fundo | colorOnBackground |
| 11 | Na Superfície | colorOnSurface |
| 12 | Com Erro | colorOnError |

Veja as cores definidas no tema padrão.

1. No Android Studio, abra o arquivo themes.xml (**app > res > values > themes.xml**).
2. O nome do tema, Theme.TipTime, é baseado no nome do app.

<style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">

1. Observe que a linha do XML também especifica um tema pai, Theme.MaterialComponents.DayNight.DarkActionBar. DayNight é um tema predefinido na biblioteca de componentes do Material Design. DarkActionBar significa que a barra de ações usa uma cor escura. Assim como uma classe herda os atributos da classe mãe, um tema herda os atributos do tema pai.

**Observação:**os atributos de cor de tema que não estiverem definidos em um tema usarão a cor do tema pai.

1. Analise os itens no arquivo e observe que os nomes são semelhantes aos do diagrama acima: colorPrimary, colorSecondary e assim por diante. themes.xml

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/purple\_500</item>  
        <item name="colorPrimaryDark">@color/purple\_700</item>  
        <item name="colorOnPrimary">@color/white</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/teal\_200</item>  
        <item name="colorSecondaryVariant">@color/teal\_700</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

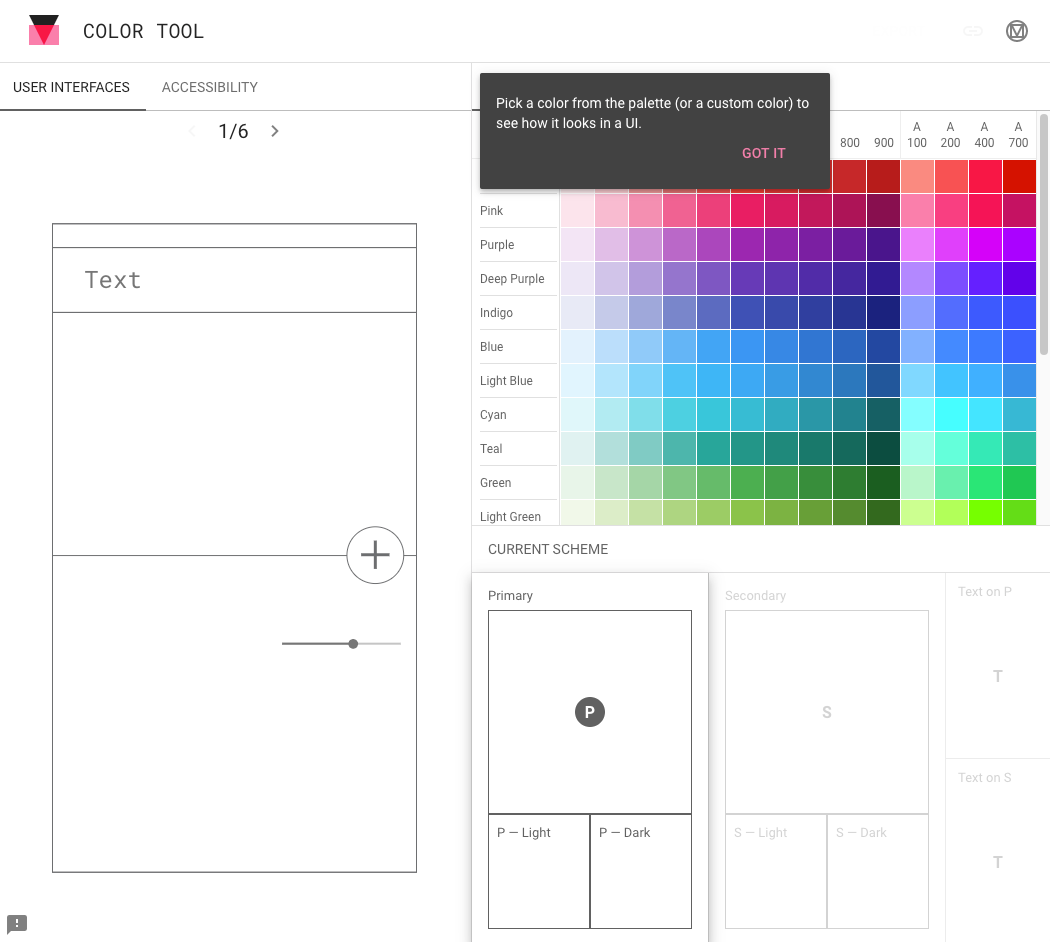
Nem todos os atributos de tema de cor são definidos. As cores que não forem definidas herdarão a cor do tema pai.

1. O Android Studio desenha uma cor pequena de exemplo na margem esquerda. 
2. Por fim, as cores são especificadas como recursos de cor, por exemplo, @color/purple\_500, em vez de usar um valor RGB diretamente.
3. Abra o arquivo colors.xml (**app > res > values > colors.xml**). Você verá os valores hexadecimais de cada recurso de cor. Lembre-se de que o #FF à esquerda é o valor alfa, o que significa que a cor é 100% opaca. colors.xml

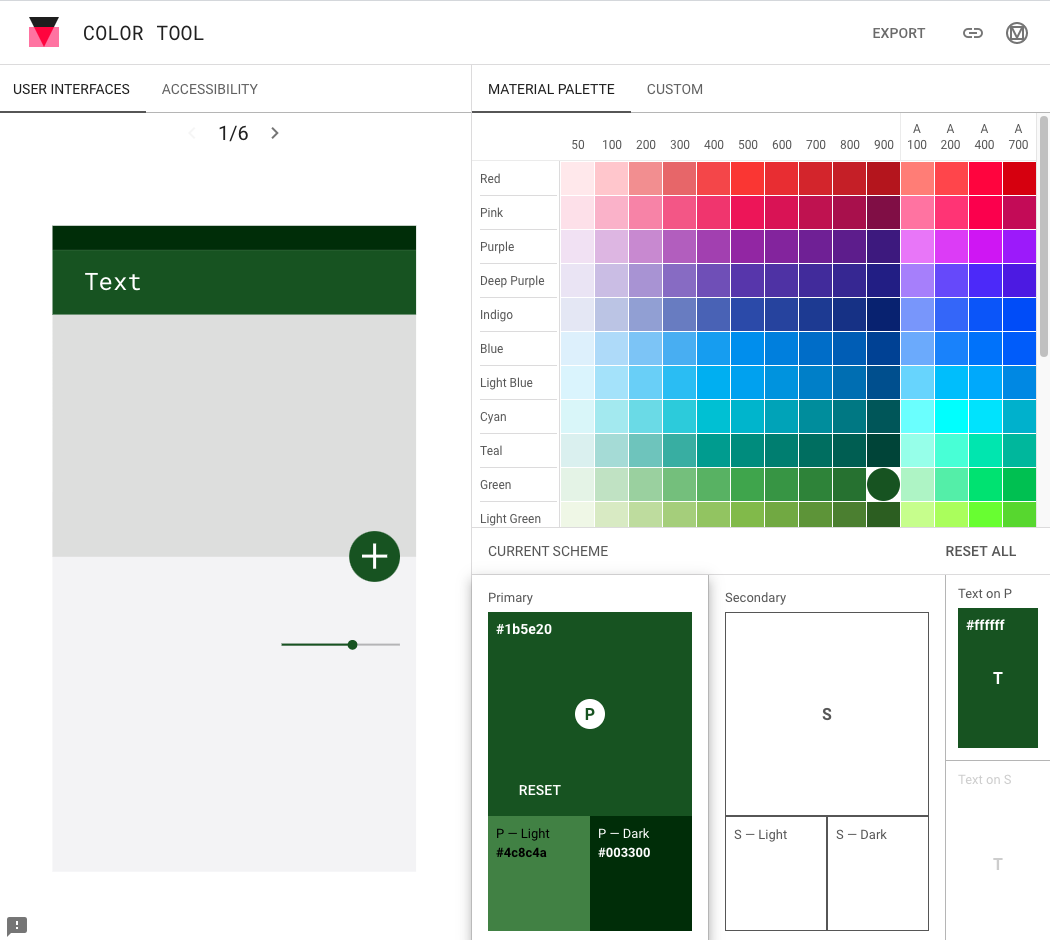
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="purple\_200">#FFBB86FC</color>  
    <color name="purple\_500">#FF6200EE</color>  
    <color name="purple\_700">#FF3700B3</color>  
    <color name="teal\_200">#FF03DAC5</color>  
    <color name="teal\_700">#FF018786</color>  
    <color name="black">#FF000000</color>  
    <color name="white">#FFFFFFFF</color>  
</resources>

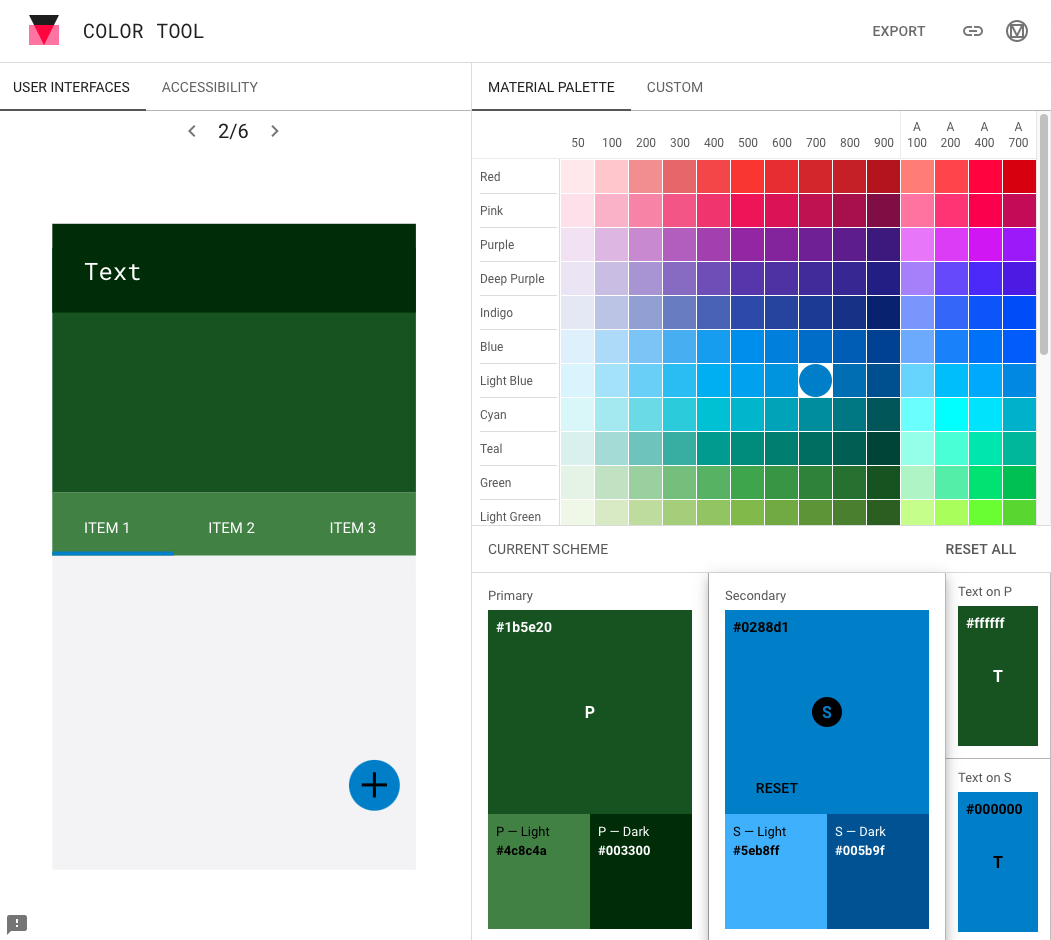
[4. Escolher cores para o tema do app](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#3)

Agora que você já conhece um pouco sobre os atributos do tema, é hora de escolher algumas cores. A maneira mais fácil de fazer isso é com o [Color Tool](https://material.io/resources/color/#!/?view.left=0&view.right=0), um app baseado na Web fornecido pela equipe do Material Design. A ferramenta oferece uma paleta de cores predefinidas e permite ver facilmente como elas serão exibidas quando usadas em diferentes elementos da IU.

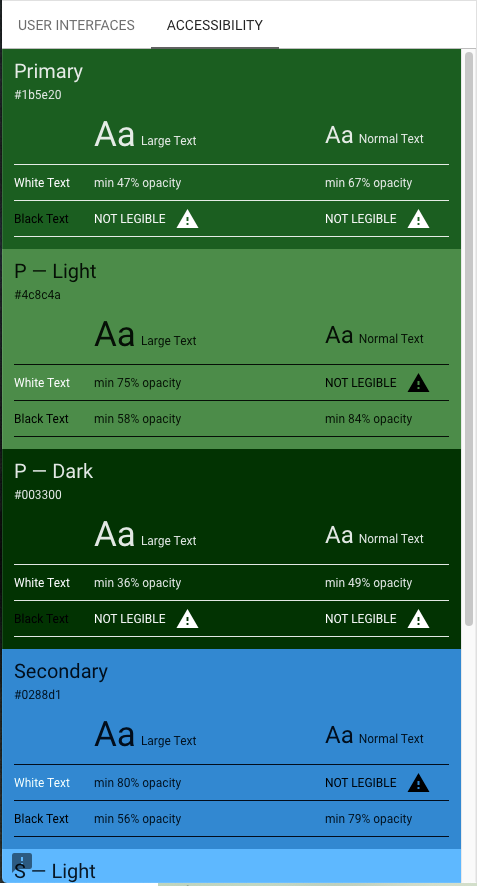


Escolher algumas cores

1. Comece selecionando uma cor principal na seção **Primary** (colorPrimary), por exemplo, **Green 900**. A ferramenta de cor mostrará a aparência em um modelo do app, além de selecionar as variantes **Light** e **Dark**. 
2. Toque na seção **Secondary** e escolha uma cor secundária (colorSecondary) de sua preferência, por exemplo, **Light Blue 700**. A cor mostra o que será exibido no modelo do app e seleciona novamente as variantes **Light** e **Dark**.
3. O modelo do app inclui seis simulações de "telas". Veja como suas opções de cor serão exibidas nas diferentes telas tocando nas setas acima do modelo.



1. A ferramenta de cores também fornece a guia **Accessibility** para informar se as cores têm contraste suficiente para serem lidas quando usadas com texto preto ou branco. Para tornar seu app mais acessível, é preciso garantir que o contraste de cores seja alto o suficiente: 4,5:1 ou maior para textos pequenos e 3,0:1 ou maior para textos grandes. Saiba mais sobre [contraste de cores](https://support.google.com/accessibility/android/answer/7158390?hl=pt-br).



1. Para primaryColorVariant e secondaryColorVariant, você pode escolher a variante clara ou escura sugerida.

**Observação:** também é possível usar o [gerador de paletas do Material Design](https://material.io/design/color/the-color-system.html#tools-for-picking-colors) para escolher uma cor secundária. Quando você escolher uma cor principal, a ferramenta sugerirá cores que sejam [complementares](https://en.wikipedia.org/wiki/Complementary_colors), [análogas](https://en.wikipedia.org/wiki/Analogous_colors) ou [triádicas](https://en.wikipedia.org/wiki/Color_scheme#Triadic).

Adicionar as cores ao app

A definição de recursos de cores facilita a reutilização consistente das mesmas cores em partes diferentes do app.

1. No Android Studio, abra o arquivo colors.xml (**app > res > values > colors.xml**).
2. Após as cores existentes, defina um recurso de cor chamado green usando o valor selecionado acima, #1B5E20.

<color name="green">#1B5E20</color>

1. Continue definindo recursos para as outras cores. A maioria dos valores é gerada pela ferramenta de cores. Observe que os valores para green\_light e blue\_light são diferentes daqueles mostrados pela ferramenta. Você vai usá-los em uma etapa futura.

|  |  |
| --- | --- |
| green | #1B5E20 |
| green\_dark | #003300 |
| green\_light | #A5D6A7 |
| blue | #0288D1 |
| blue\_dark | #005B9F |
| blue\_light | #81D4FA |

Como já existem recursos de cor definidos para preto e branco, você não precisa defini-los.

O arquivo colors.xml do app agora terá esta aparência:

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="purple\_200">#FFBB86FC</color>  
    <color name="purple\_500">#FF6200EE</color>  
    <color name="purple\_700">#FF3700B3</color>  
    <color name="teal\_200">#FF03DAC5</color>  
    <color name="teal\_700">#FF018786</color>  
    <color name="black">#FF000000</color>  
    <color name="white">#FFFFFFFF</color>  
  
    <color name="green">#1B5E20</color>  
    <color name="green\_dark">#003300</color>  
    <color name="green\_light">#A5D6A7</color>  
    <color name="blue">#0288D1</color>  
    <color name="blue\_dark">#005B9F</color>  
    <color name="blue\_light">#81D4FA</color>  
</resources>

Usar as cores no tema

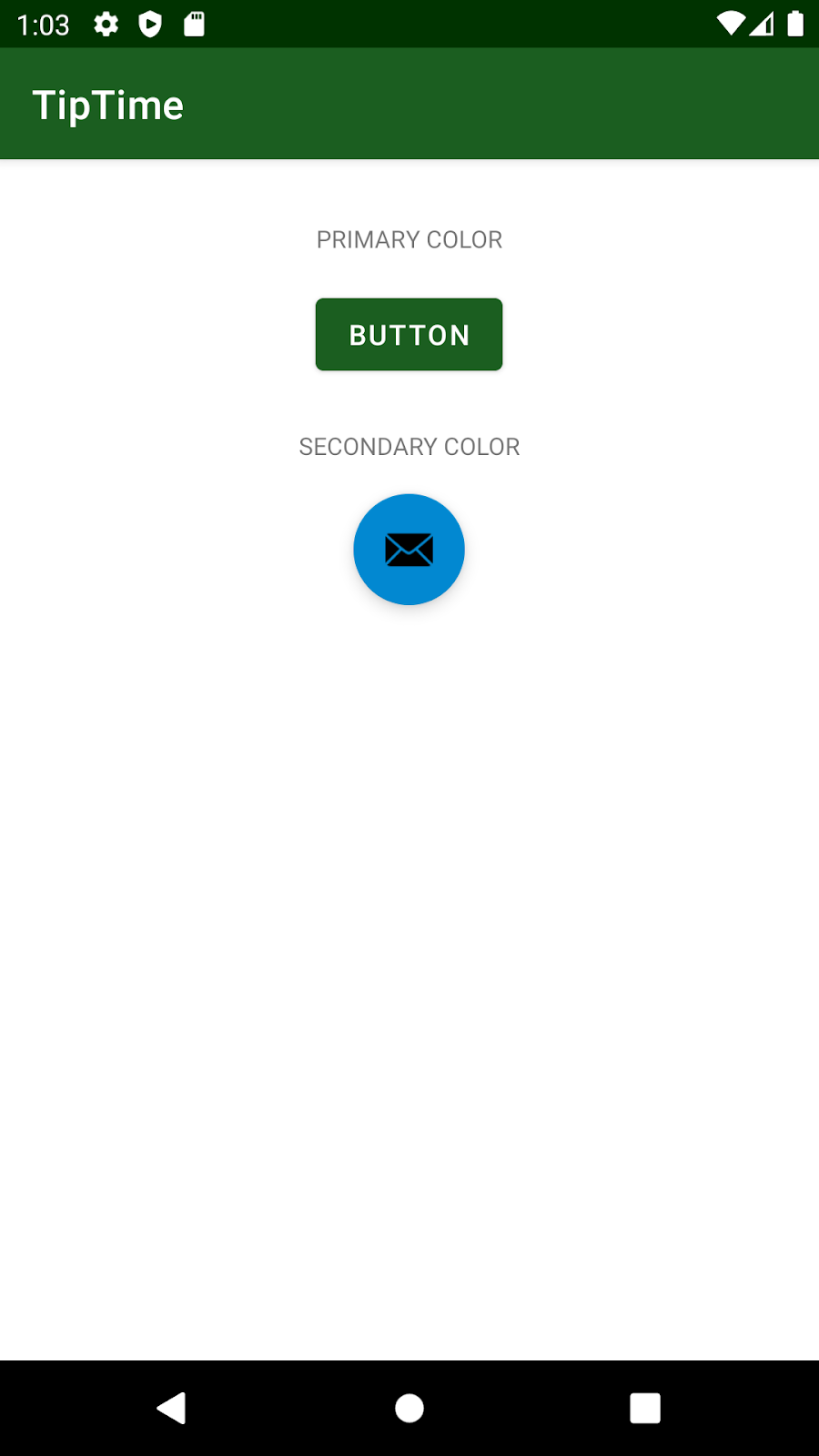
Agora que você definiu nomes para as cores selecionadas, é hora de usá-las no seu tema.

1. Abra o arquivo themes.xml (**app > res > values > themes > themes.xml**).
2. Mude a colorPrimary para a cor principal que você selecionou, @color/green.
3. Mude colorPrimaryVariant para @color/green\_dark.
4. Mude colorSecondary para @color/blue.
5. Mude colorSecondaryVariant para @color/blue\_dark.
6. Verifique se as opções **Text on P** e **Text on S** ainda estão definidas como branco (#FFFFFF) e preto (#000000). Se você estiver usando a ferramenta de cores por conta própria e selecionar outras cores, talvez seja necessário definir mais recursos de cor.

Quando terminar, o tema ficará assim:

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green</item>  
        <item name="colorPrimaryVariant">@color/green\_dark</item>  
        <item name="colorOnPrimary">@color/white</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue</item>  
        <item name="colorSecondaryVariant">@color/blue\_dark</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

1. Execute seu app no emulador ou em um dispositivo e veja o visual do novo tema no app.



## [5. Tema escuro](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#4)

O modelo de app incluiu um tema claro padrão e também uma variante de [tema escuro](https://developer.android.com/guide/topics/ui/look-and-feel/darktheme?hl=pt-br). Um tema escuro usa cores mais escuras e discretas e:

* Pode reduzir significativamente o consumo de energia (dependendo da tecnologia da tela do dispositivo).
* Melhora a visibilidade para usuários com problemas de visão e que tenham sensibilidade ao brilho da luz.
* Facilita o uso do dispositivo em um ambiente com pouca luz.

## Como escolher cores para o tema escuro

As cores de um tema escuro precisam ser legíveis. Os temas escuros usam uma cor de superfície escura com tons de cores limitados. Para garantir a legibilidade, as cores primárias geralmente são versões menos saturadas das cores primárias do tema claro.

Para oferecer mais flexibilidade e usabilidade em um tema escuro, recomendamos usar tons mais claros (200 a 50) em um tema escuro, em vez do tema de cores padrão (tons de saturação de 900 a 500). Anteriormente, você escolheu verde 200 e azul claro 200 como as cores claras. Para o app, você usará as cores claras como as principais e as cores primárias como as variantes.

## Atualizar a versão escura do tema

1. Abra o arquivo themes.xml (night)( **app> res> values> themes> themes.xml (night)** ).

**Observação:** este arquivo **themes.xml** é diferente do arquivo **themes.xml** anterior. Ele contém a versão escura do tema. Os recursos neste arquivo serão usados quando o **tema escuro** estiver ativado no dispositivo.

1. Mude a colorPrimary para a variante clara da cor primária que você selecionou, @color/green\_light.
2. Mude colorPrimaryVariant para @color/green.
3. Mude colorSecondary para @color/blue\_light.
4. Mude colorSecondaryVariant para @color/blue\_light.

Quando você terminar, seu arquivo themes.xml (night) terá esta aparência:

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Application theme for dark theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green\_light</item>  
        <item name="colorPrimaryVariant">@color/green</item>  
        <item name="colorOnPrimary">@color/black</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue\_light</item>  
        <item name="colorSecondaryVariant">@color/blue\_light</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

1. Agora, as cores originais definidas no arquivo colors.xml, por exemplo, purple\_200, não são mais usadas; então você pode excluí-las.

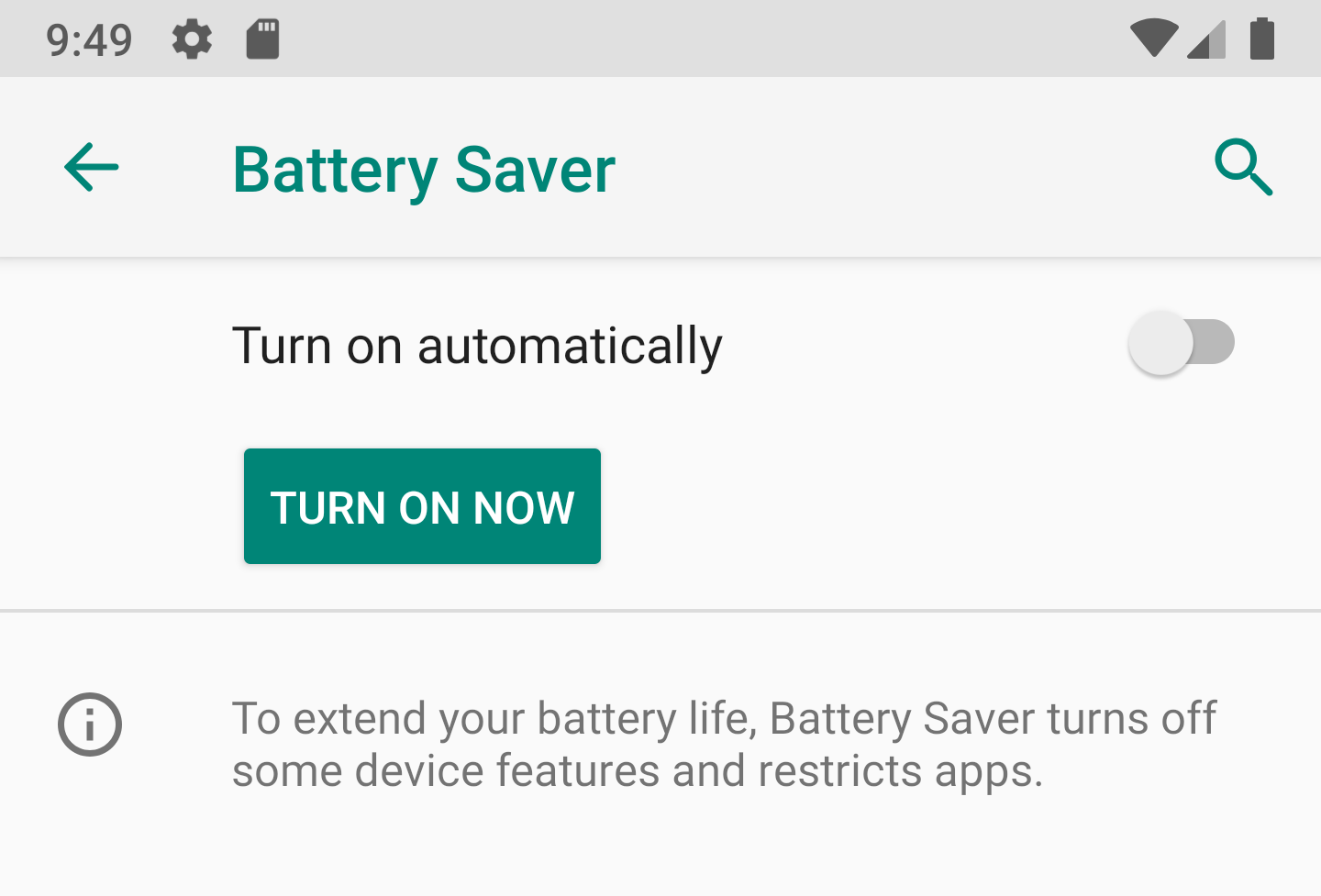
## Testar o tema escuro

Você pode ver como seu tema será exibido no modo escuro ativando-o no dispositivo.

**Observação**: o tema escuro requer um dispositivo ou emulador com a API 28 (Android 9) ou a versão 29 (Android 10) ou mais recente.

### **Para a API 28 (Android 9)**

1. Execute o app novamente.
2. Acesse o app **Config.**.
3. Na seção **Bateria**, localize **Economia de bateria**.

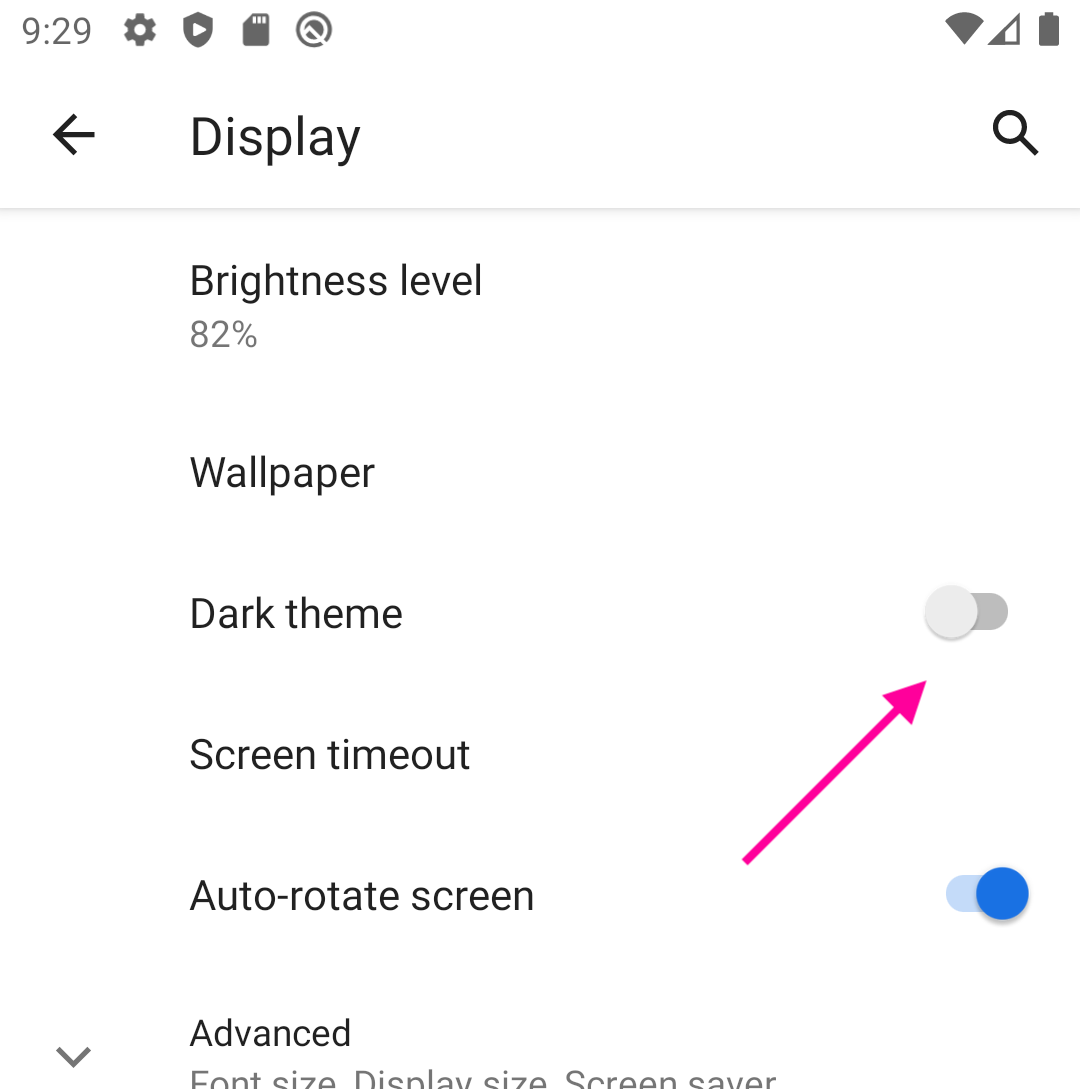


1. Pressione **Ativar agora**.

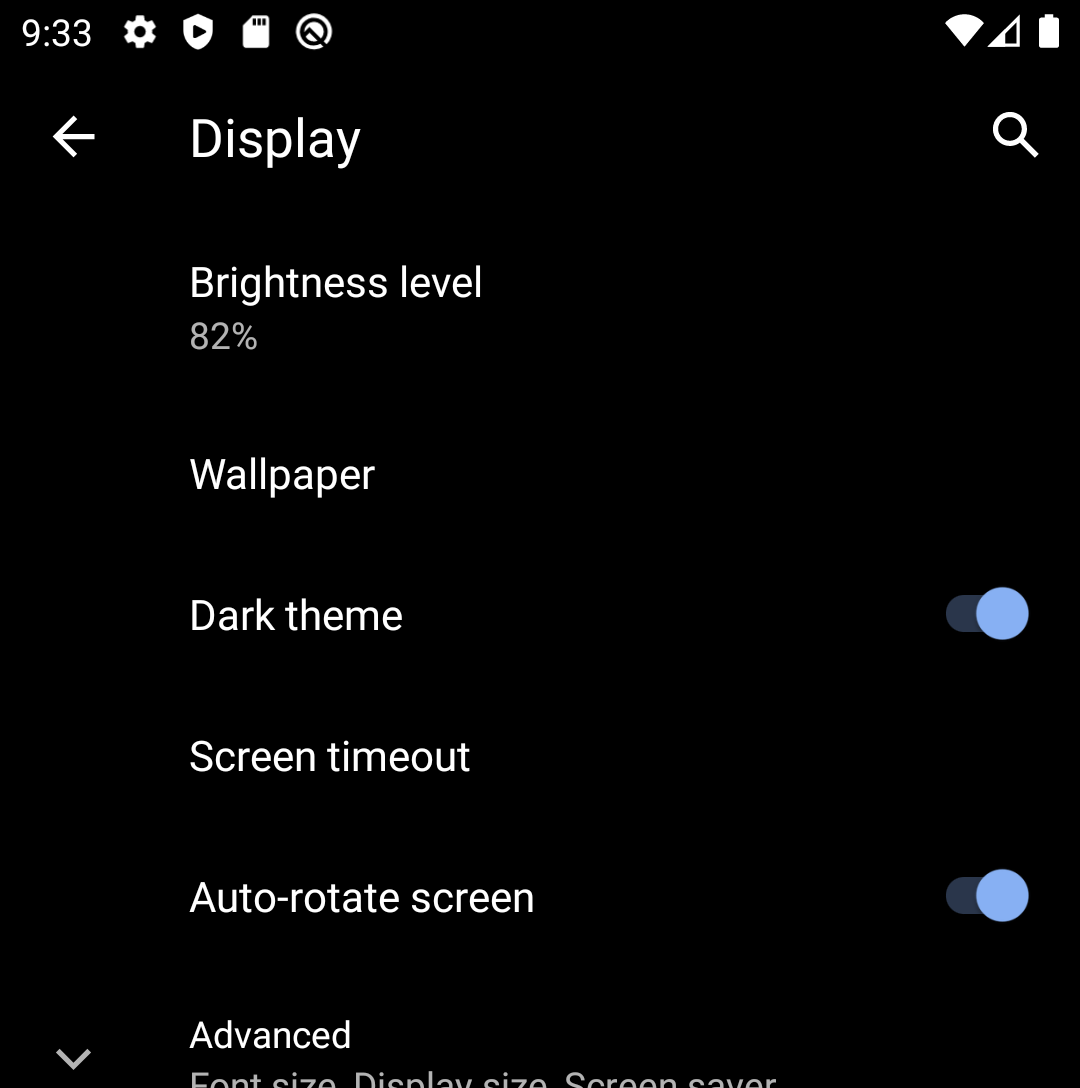
Conclua as etapas abaixo.

### **Para a API 29 (Android 10) ou mais recente**

1. Execute o app novamente.
2. Acesse o app **Config.**.
3. Na seção **Tela**, localize a chave **Tema escuro**.

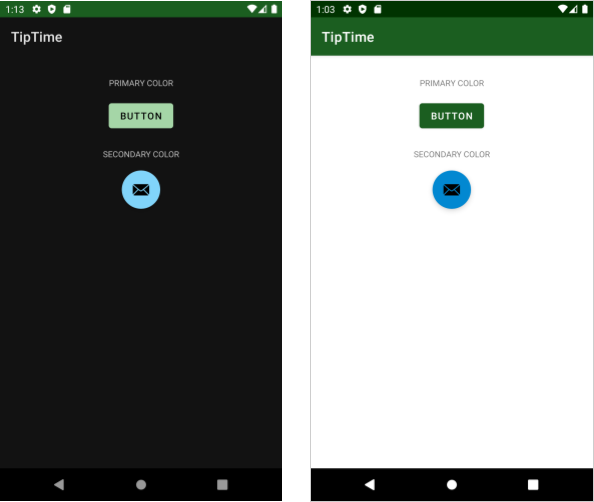


1. Mude a posição da chave **Tema escuro** para "Ativado". O dispositivo passará a usar o modo noturno.



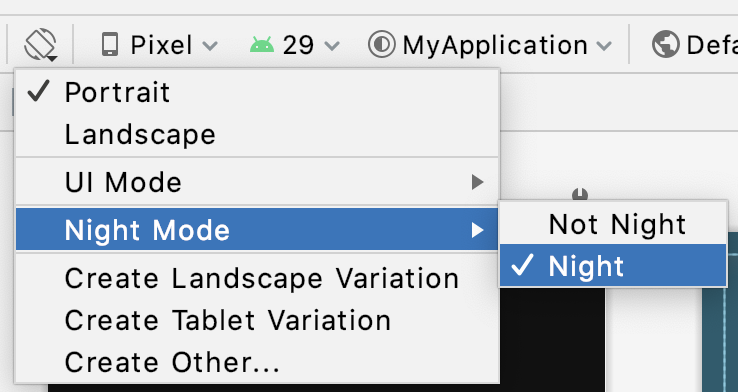
### **Para qualquer API**

1. Volte para seu app e veja as diferenças.



A mudança mais óbvia é que o segundo plano está mais escuro e com texto claro, em vez de um segundo plano claro com texto escuro. Além disso, as cores usadas para os botões são menos vibrantes no tema escuro do que no tema claro.

Parabéns! Você criou um novo tema para o app com um tema claro e um escuro.

**Observação:** você também pode ver como seu tema será exibido usando o **Design Editor** do Android Studio. O menu **Orientation for Preview** contém configurações para **Night** ou **Not Night**. 

## [6. Código da solução](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#5)

Este codelab se concentrou na personalização das cores do tema, mas há muitos atributos que podem ser personalizados, incluindo texto, forma, estilo de botões e muito mais. Confira este artigo para ver outras formas de personalizar um tema de app. [Estilo do Android: atributos de tema comuns](https://medium.com/androiddevelopers/android-styling-common-theme-attributes-8f7c50c9eaba) (em inglês).

Veja abaixo o código da solução para este codelab.

colors.xml( **app> res> values> colors.xml** )

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="black">#FF000000</color>  
    <color name="white">#FFFFFFFF</color>  
    <color name="green">#1B5E20</color>  
    <color name="green\_dark">#003300</color>  
    <color name="green\_light">#A5D6A7</color>  
    <color name="blue">#0288D1</color>  
    <color name="blue\_dark">#005B9F</color>  
    <color name="blue\_light">#81D4FA</color>  
</resources>

values/themes.xml( **app> res> valores> temas> themes.xml** )

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green</item>  
        <item name="colorPrimaryVariant">@color/green\_dark</item>  
        <item name="colorOnPrimary">@color/white</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue</item>  
        <item name="colorSecondaryVariant">@color/blue\_dark</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

values-night/themes.xml( **app> res> valores> temas> themes.xml (noite)** )

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Application theme for dark theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green\_light</item>  
        <item name="colorPrimaryVariant">@color/green</item>  
        <item name="colorOnPrimary">@color/black</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue\_light</item>  
        <item name="colorSecondaryVariant">@color/blue\_light</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

Mudar o tema do app

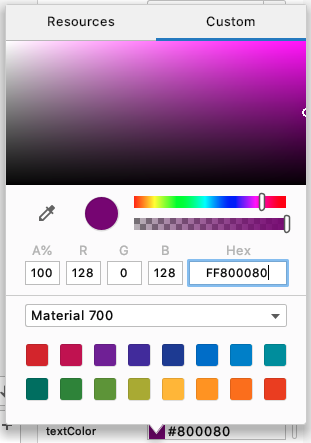
## [2. Design e cor](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#1)

## Material Design

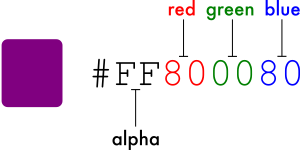
O [Material Design](https://material.io/design/introduction) (link em inglês) é inspirado no mundo físico e nas texturas dele, incluindo a forma como os objetos refletem a luz e projetam sombras. Ele apresenta diretrizes sobre como criar a IU do app de uma forma legível, atraente e consistente. Com o [Material Theming](https://material.io/design/material-theming/overview.html#material-theming) (link em inglês), você pode adaptar o Material Design ao app seguindo as orientações de personalização de cores, tipografia e formas. O Material Design vem com um tema de referência, que pode ser usado sem mudanças. Você pode personalizá-lo o quanto quiser para que o Material Design se adapte ao seu app.

## Um pouco sobre cores

As cores estão por toda parte ao nosso redor, no mundo real e no mundo digital. A primeira coisa que você precisa saber é que nem todos os usuários veem as cores da mesma maneira. Então, é importante escolher cores para o app de modo que ele possa ser usado facilmente pelos usuários. Escolher cores com contraste suficiente é apenas uma etapa para [criar apps mais acessíveis](https://developer.android.com/guide/topics/ui/accessibility?hl=pt-br).



Uma [Cor](https://developer.android.com/reference/kotlin/android/graphics/Color?hl=pt-br) pode ser representada por três números hexadecimais, #00-#FF (0-255), representando os componentes vermelho, verde e azul (RGB) dessa cor. Quanto maior o número, maior a proporção da cor específica.



Uma cor também pode ser definida incluindo um valor alfa #00-#FF, que representa a transparência (#00 = 0% = totalmente transparente, #FF = 100% = totalmente opaco). Quando incluído, o valor alfa é o primeiro de quatro números hexadecimais (ARGB). Se não houver um valor alfa, ele será considerado #FF = 100% opaco.

No entanto, não é necessário gerar os números hexadecimais manualmente. Há ferramentas disponíveis para ajudar você a escolher as cores que gerarão os números.

Alguns exemplos que você pode ter visto no arquivo colors.xml do seu app Android incluem 100% preto (R=#00, G=#00, B=#00) e 100% branco (R=#FF, G=#FF, B=#FF):

<color name="black">#FF000000</color>  
<color name="white">#FFFFFFFF</color>

[3. Temas](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#2)

Um *estilo* pode especificar atributos para uma View, como cor e tamanho da fonte, cor do segundo plano e muito mais.

O *tema* é uma coleção de estilos aplicados a um app, atividade ou hierarquia de visualização inteiros, não apenas a uma View individual. Quando você aplica um tema a um app, atividade, visualização individual ou em grupo, o tema é aplicado a esse elemento e a todos os filhos deles. Os temas também podem aplicar estilos a elementos que não são de visualização, como a barra de status e o segundo plano da janela.

Criar um projeto Empty Activity

Se você estiver usando seu próprio app, pule esta seção. Se precisar de um app para usar como exemplo, siga estas etapas para criar um app Empty Activity.

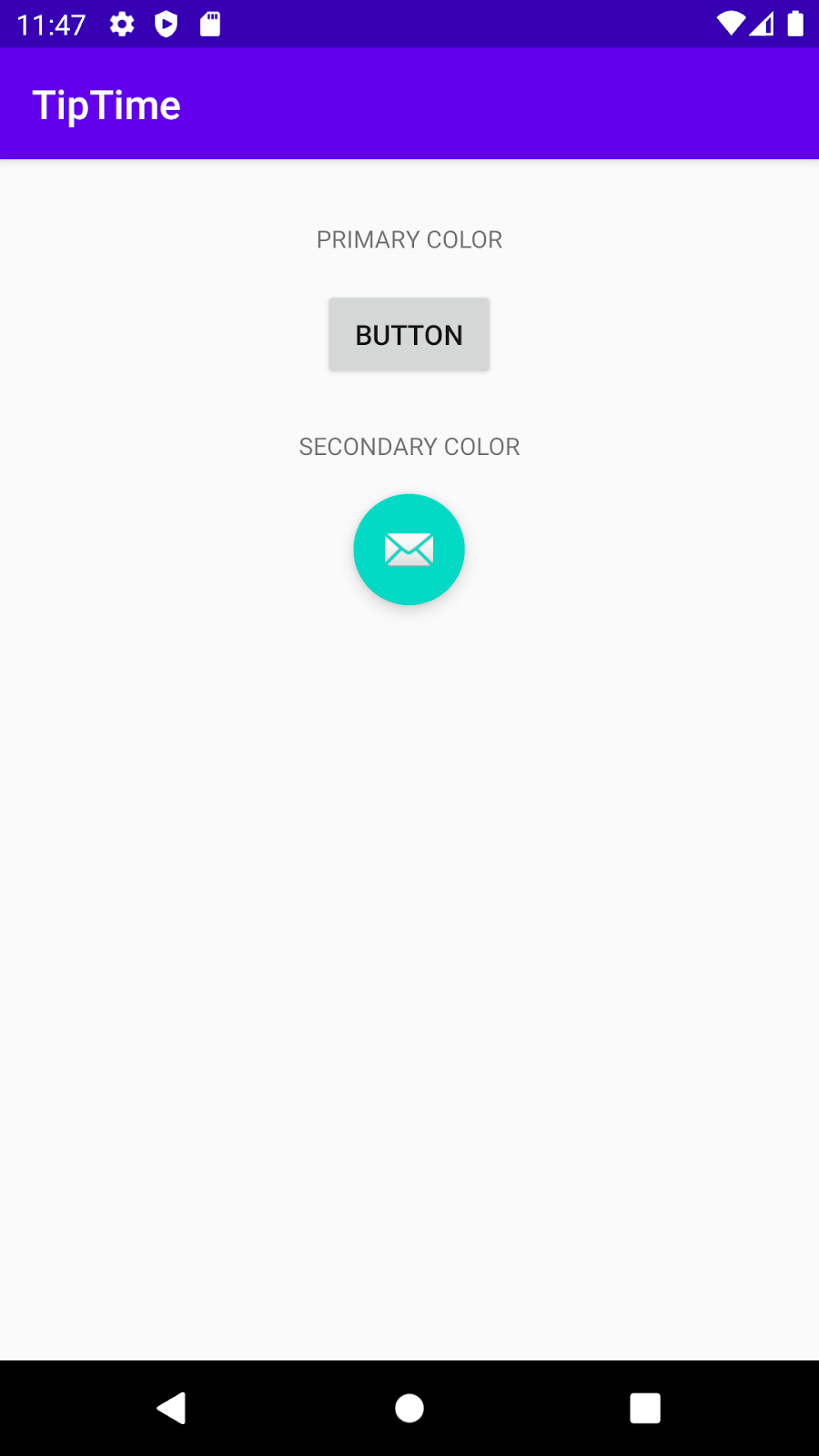
1. Abra o Android Studio.
2. Crie um novo projeto Kotlin usando o modelo **Empty Activity**.
3. Use o nome "TipTime". Como alternativa, você pode manter o nome padrão, "My Application", se não estiver participando de outros codelabs.
4. Selecione uma API de nível 19 no mínimo (KitKat).
5. Depois que o Android Studio concluir a criação do app, abra o arquivo activity\_main.xml (**app > res > layout > activity\_main.xml**).
6. Se necessário, alterne para a visualização **Code**.
7. Substitua todo o texto por este XML:

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:layout\_margin="16dp"  
    android:orientation="vertical"  
    tools:context=".MainActivity">  
  
    <TextView  
        android:layout\_width="wrap\_content"  
        android:layout\_height="48dp"  
        android:layout\_gravity="center\_horizontal"  
        android:gravity="center\_vertical"  
        android:text="@string/primary\_color"  
        android:textAllCaps="true"  
        android:textSize="12sp" />  
  
    <Button  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="center\_horizontal"  
        android:text="@string/button" />  
  
    <TextView  
        android:layout\_width="wrap\_content"  
        android:layout\_height="48dp"  
        android:layout\_gravity="center\_horizontal"  
        android:layout\_marginTop="8dp"  
        android:gravity="center\_vertical"  
        android:text="@string/secondary\_color"  
        android:textAllCaps="true"  
        android:textSize="12sp" />  
  
    <com.google.android.material.floatingactionbutton.FloatingActionButton  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="center\_horizontal"  
        android:contentDescription="@string/email\_icon"  
        app:srcCompat="@android:drawable/ic\_dialog\_email" />  
  
</LinearLayout>

1. Abra o arquivo strings.xml (**app > res > values > strings.xml**).
2. Substitua todo o texto por este XML:

<resources>  
    <string name="app\_name">TipTime</string>  
    <string name="primary\_color">Primary color</string>  
    <string name="button">Button</string>  
    <string name="secondary\_color">Secondary color</string>  
    <string name="email\_icon">email icon</string>  
</resources>

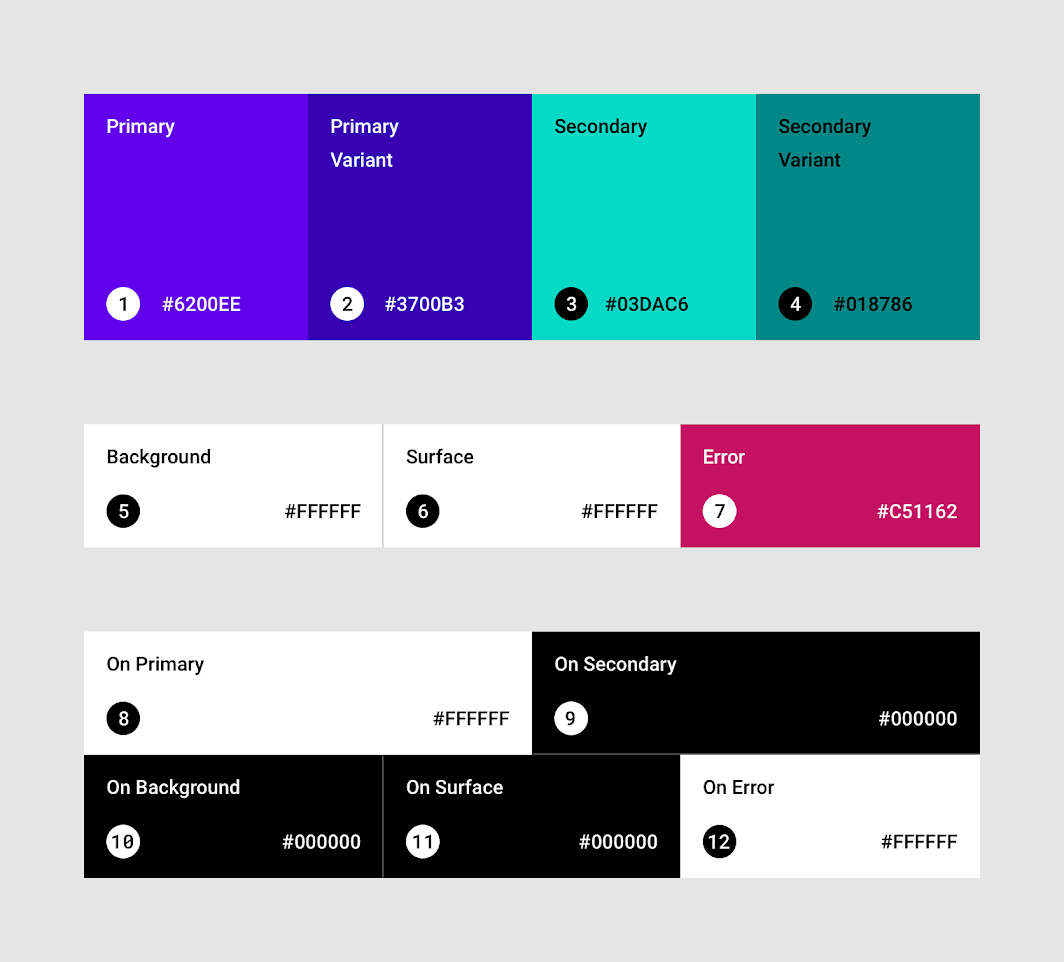
1. Execute o app. Ele ficará parecido com a captura de tela abaixo.



O app inclui uma TextView e um Button para permitir que você veja as opções de cor em um app Android real. A cor do botão mudará para a cor principal do tema nas próximas etapas.

Saiba mais sobre cores de tema

Cada parte da IU dos apps Android usa cores diferentes. Para ajudar você a usar as cores de forma significativa no app e aplicá-las consistentemente, o sistema de tema agrupa as cores em [12 atributos nomeados](https://material.io/design/material-theming/implementing-your-theme.html#color) relacionados à cor que será usada pelo texto, pelos ícones e muito mais. Seu tema não precisa especificar todos eles. Você escolherá as cores primárias e secundárias, além das cores do texto e dos ícones desenhados com essas cores.



As cores "ativadas" são usadas para textos e ícones desenhados nas diferentes superfícies.

|  |  |  |
| --- | --- | --- |
| **MULHERES** | **Nome** | **Atributo do tema** |
| 1 | Primário | colorPrimary |
| 2 | Variante Primária | colorPrimaryVariant |
| 3 | Secundário | colorSecondary |
| 4 | Variante Secundária | colorSecondaryVariant |
| 5 | Fundo | colorBackground |
| 6 | Superfície | colorSurface |
| 7 | Erro | colorError |
| 8 | No Primário | colorOnPrimary |
| 9 | No Secundário | colorOnSecondary |
| 10 | No fundo | colorOnBackground |
| 11 | Na Superfície | colorOnSurface |
| 12 | Com Erro | colorOnError |

Veja as cores definidas no tema padrão.

1. No Android Studio, abra o arquivo themes.xml (**app > res > values > themes.xml**).
2. O nome do tema, Theme.TipTime, é baseado no nome do app.

<style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">

1. Observe que a linha do XML também especifica um tema pai, Theme.MaterialComponents.DayNight.DarkActionBar. DayNight é um tema predefinido na biblioteca de componentes do Material Design. DarkActionBar significa que a barra de ações usa uma cor escura. Assim como uma classe herda os atributos da classe mãe, um tema herda os atributos do tema pai.

**Observação:**os atributos de cor de tema que não estiverem definidos em um tema usarão a cor do tema pai.

1. Analise os itens no arquivo e observe que os nomes são semelhantes aos do diagrama acima: colorPrimary, colorSecondary e assim por diante. themes.xml

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/purple\_500</item>  
        <item name="colorPrimaryDark">@color/purple\_700</item>  
        <item name="colorOnPrimary">@color/white</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/teal\_200</item>  
        <item name="colorSecondaryVariant">@color/teal\_700</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

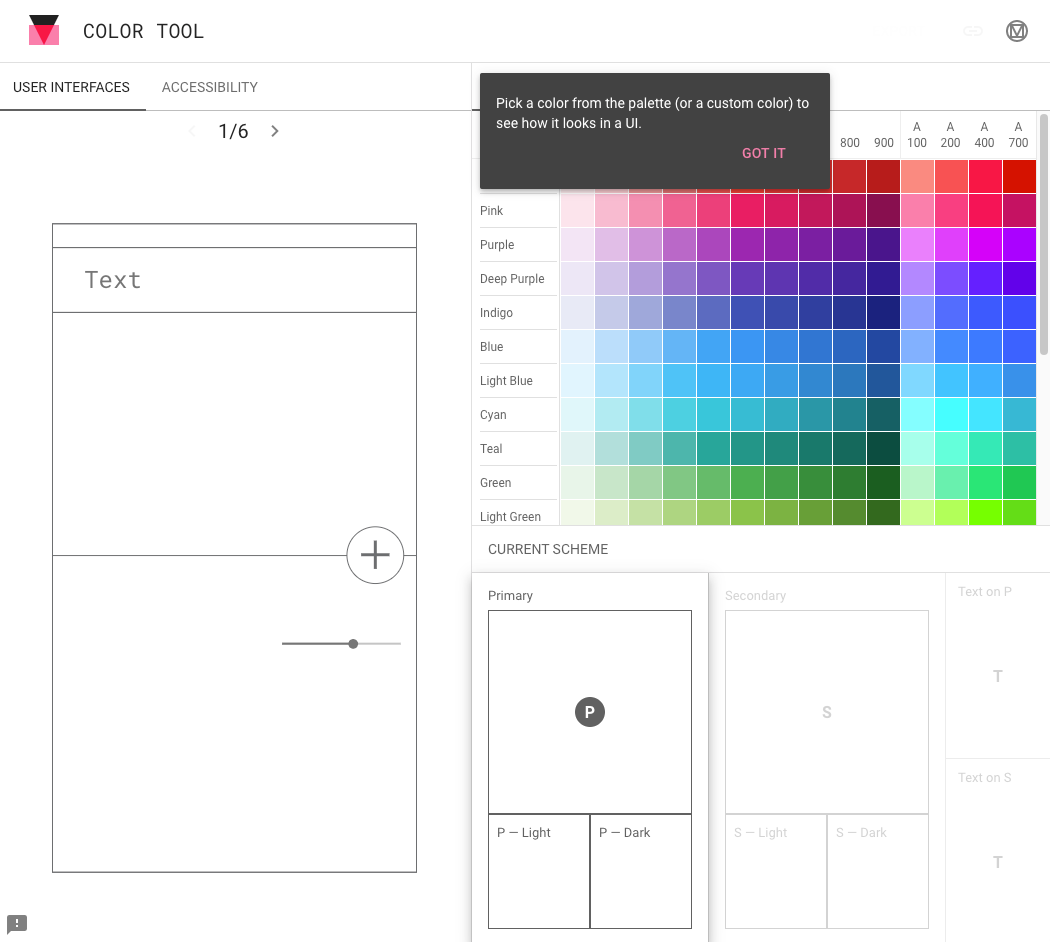
Nem todos os atributos de tema de cor são definidos. As cores que não forem definidas herdarão a cor do tema pai.

1. O Android Studio desenha uma cor pequena de exemplo na margem esquerda. 
2. Por fim, as cores são especificadas como recursos de cor, por exemplo, @color/purple\_500, em vez de usar um valor RGB diretamente.
3. Abra o arquivo colors.xml (**app > res > values > colors.xml**). Você verá os valores hexadecimais de cada recurso de cor. Lembre-se de que o #FF à esquerda é o valor alfa, o que significa que a cor é 100% opaca. colors.xml

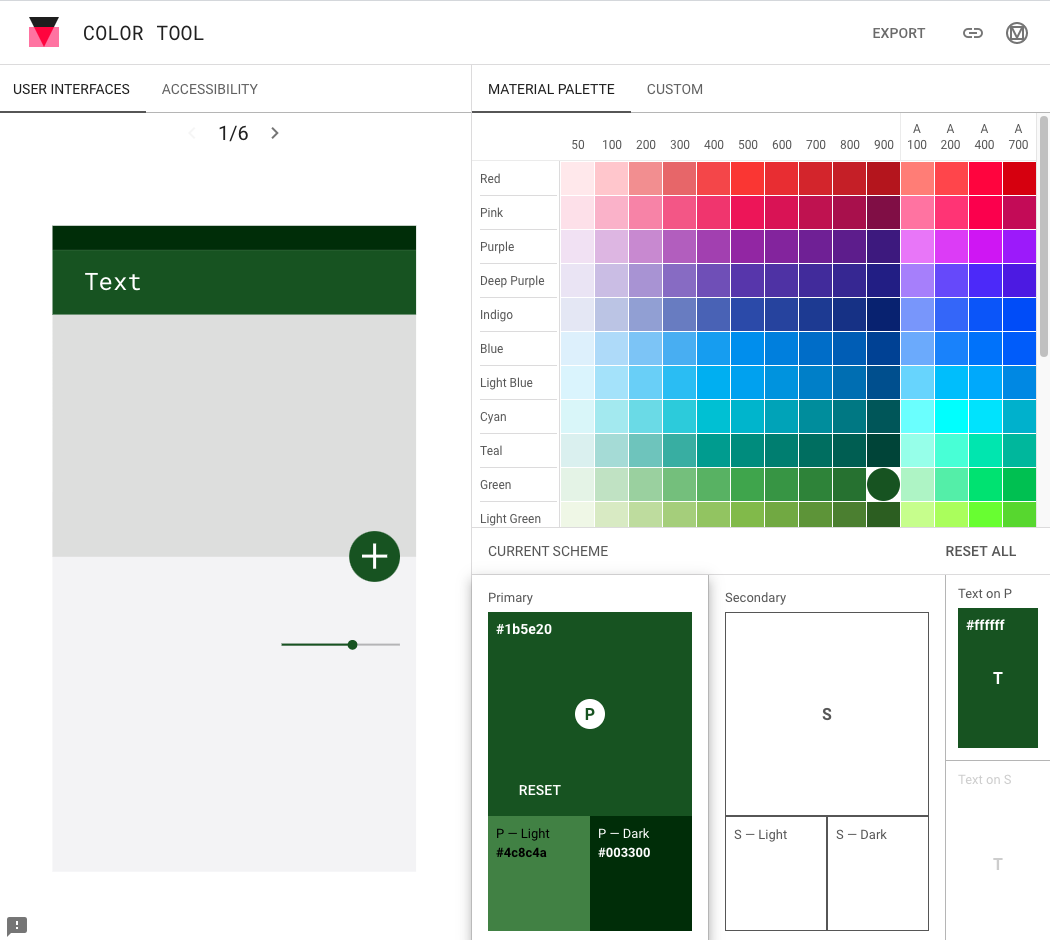
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="purple\_200">#FFBB86FC</color>  
    <color name="purple\_500">#FF6200EE</color>  
    <color name="purple\_700">#FF3700B3</color>  
    <color name="teal\_200">#FF03DAC5</color>  
    <color name="teal\_700">#FF018786</color>  
    <color name="black">#FF000000</color>  
    <color name="white">#FFFFFFFF</color>  
</resources>

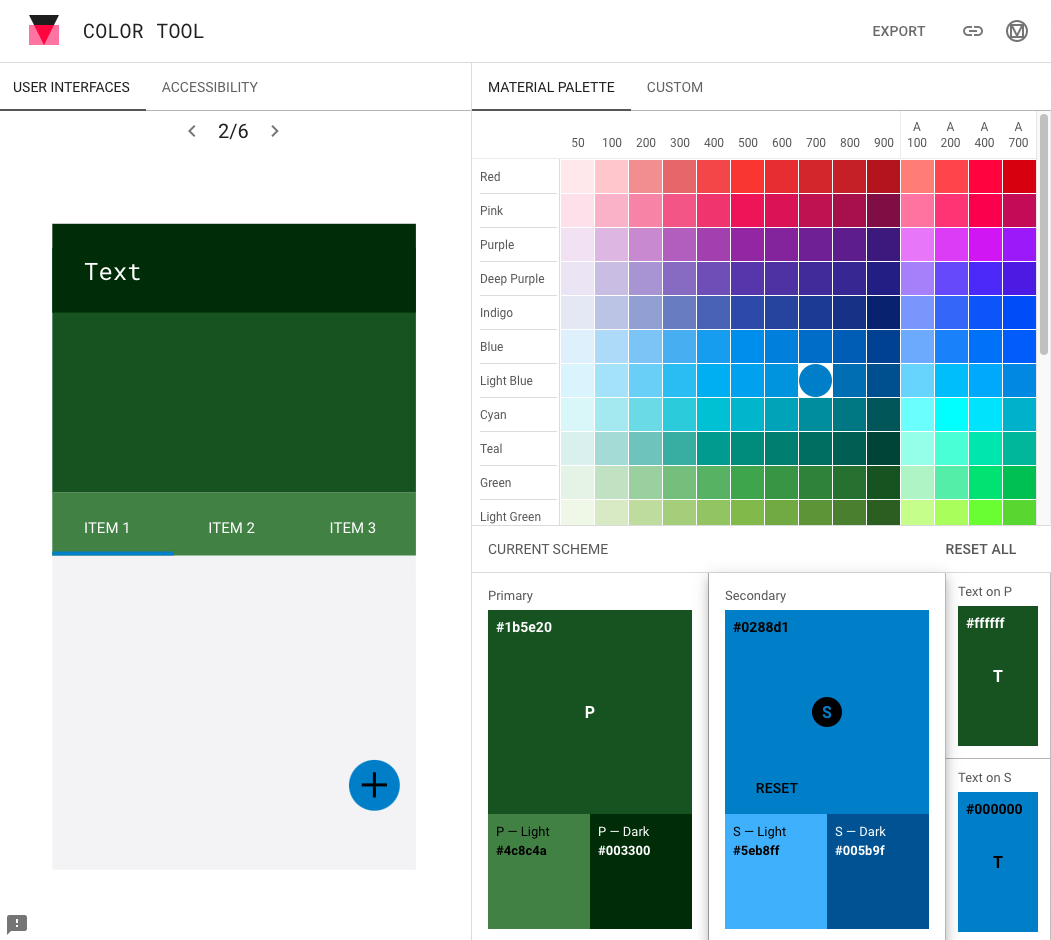
[4. Escolher cores para o tema do app](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#3)

Agora que você já conhece um pouco sobre os atributos do tema, é hora de escolher algumas cores. A maneira mais fácil de fazer isso é com o [Color Tool](https://material.io/resources/color/#!/?view.left=0&view.right=0), um app baseado na Web fornecido pela equipe do Material Design. A ferramenta oferece uma paleta de cores predefinidas e permite ver facilmente como elas serão exibidas quando usadas em diferentes elementos da IU.

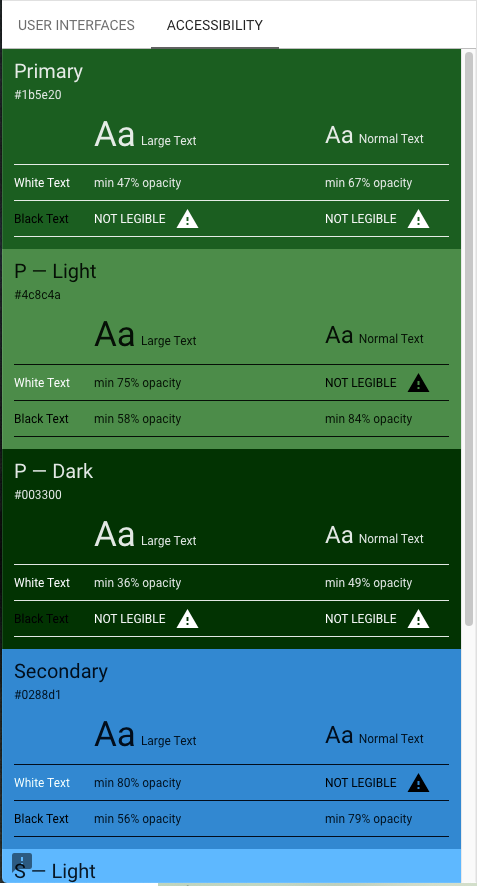


Escolher algumas cores

1. Comece selecionando uma cor principal na seção **Primary** (colorPrimary), por exemplo, **Green 900**. A ferramenta de cor mostrará a aparência em um modelo do app, além de selecionar as variantes **Light** e **Dark**. 
2. Toque na seção **Secondary** e escolha uma cor secundária (colorSecondary) de sua preferência, por exemplo, **Light Blue 700**. A cor mostra o que será exibido no modelo do app e seleciona novamente as variantes **Light** e **Dark**.
3. O modelo do app inclui seis simulações de "telas". Veja como suas opções de cor serão exibidas nas diferentes telas tocando nas setas acima do modelo.



1. A ferramenta de cores também fornece a guia **Accessibility** para informar se as cores têm contraste suficiente para serem lidas quando usadas com texto preto ou branco. Para tornar seu app mais acessível, é preciso garantir que o contraste de cores seja alto o suficiente: 4,5:1 ou maior para textos pequenos e 3,0:1 ou maior para textos grandes. Saiba mais sobre [contraste de cores](https://support.google.com/accessibility/android/answer/7158390?hl=pt-br).



1. Para primaryColorVariant e secondaryColorVariant, você pode escolher a variante clara ou escura sugerida.

**Observação:** também é possível usar o [gerador de paletas do Material Design](https://material.io/design/color/the-color-system.html#tools-for-picking-colors) para escolher uma cor secundária. Quando você escolher uma cor principal, a ferramenta sugerirá cores que sejam [complementares](https://en.wikipedia.org/wiki/Complementary_colors), [análogas](https://en.wikipedia.org/wiki/Analogous_colors) ou [triádicas](https://en.wikipedia.org/wiki/Color_scheme#Triadic).

Adicionar as cores ao app

A definição de recursos de cores facilita a reutilização consistente das mesmas cores em partes diferentes do app.

1. No Android Studio, abra o arquivo colors.xml (**app > res > values > colors.xml**).
2. Após as cores existentes, defina um recurso de cor chamado green usando o valor selecionado acima, #1B5E20.

<color name="green">#1B5E20</color>

1. Continue definindo recursos para as outras cores. A maioria dos valores é gerada pela ferramenta de cores. Observe que os valores para green\_light e blue\_light são diferentes daqueles mostrados pela ferramenta. Você vai usá-los em uma etapa futura.

|  |  |
| --- | --- |
| green | #1B5E20 |
| green\_dark | #003300 |
| green\_light | #A5D6A7 |
| blue | #0288D1 |
| blue\_dark | #005B9F |
| blue\_light | #81D4FA |

Como já existem recursos de cor definidos para preto e branco, você não precisa defini-los.

O arquivo colors.xml do app agora terá esta aparência:

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="purple\_200">#FFBB86FC</color>  
    <color name="purple\_500">#FF6200EE</color>  
    <color name="purple\_700">#FF3700B3</color>  
    <color name="teal\_200">#FF03DAC5</color>  
    <color name="teal\_700">#FF018786</color>  
    <color name="black">#FF000000</color>  
    <color name="white">#FFFFFFFF</color>  
  
    <color name="green">#1B5E20</color>  
    <color name="green\_dark">#003300</color>  
    <color name="green\_light">#A5D6A7</color>  
    <color name="blue">#0288D1</color>  
    <color name="blue\_dark">#005B9F</color>  
    <color name="blue\_light">#81D4FA</color>  
</resources>

Usar as cores no tema

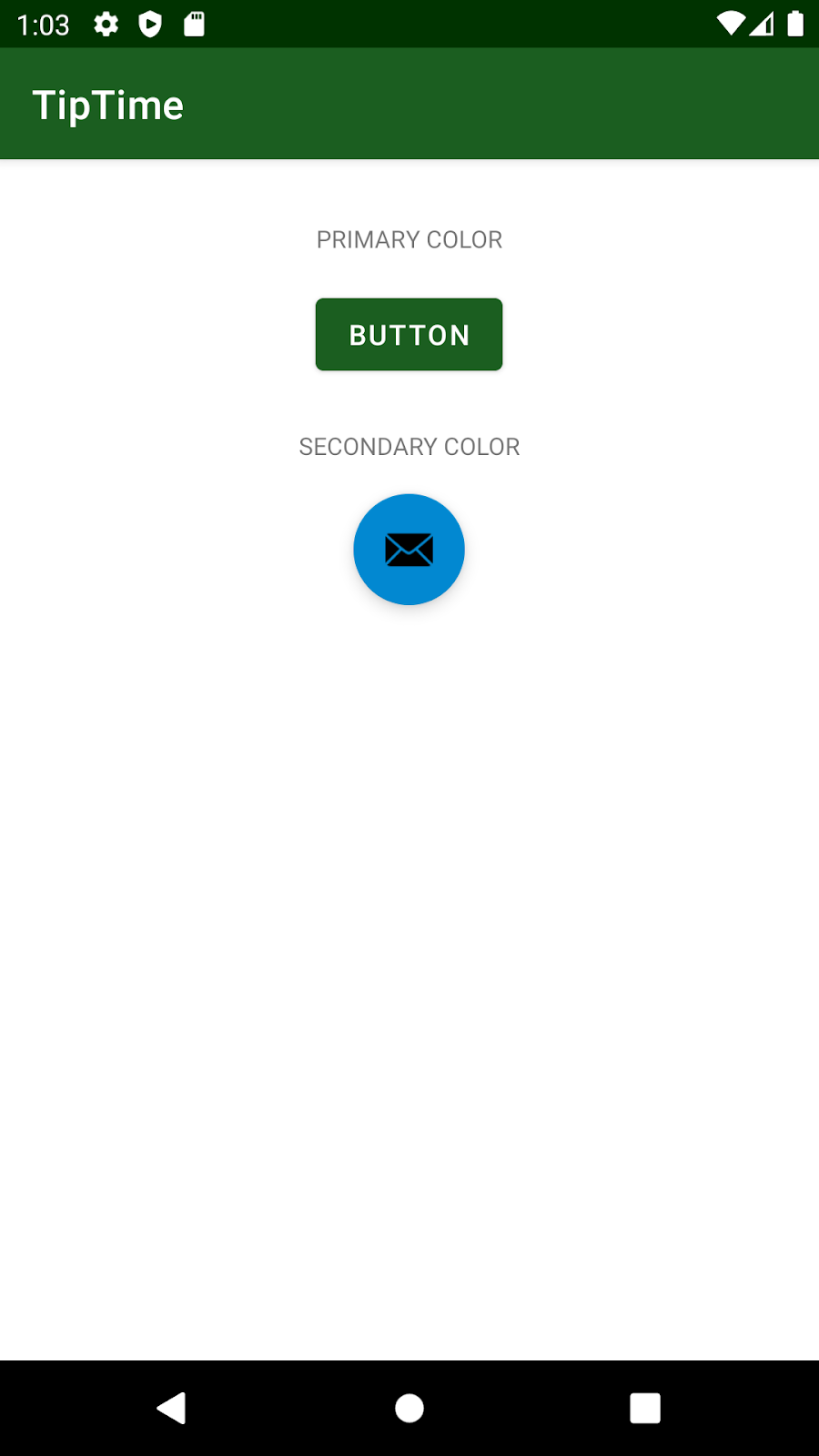
Agora que você definiu nomes para as cores selecionadas, é hora de usá-las no seu tema.

1. Abra o arquivo themes.xml (**app > res > values > themes > themes.xml**).
2. Mude a colorPrimary para a cor principal que você selecionou, @color/green.
3. Mude colorPrimaryVariant para @color/green\_dark.
4. Mude colorSecondary para @color/blue.
5. Mude colorSecondaryVariant para @color/blue\_dark.
6. Verifique se as opções **Text on P** e **Text on S** ainda estão definidas como branco (#FFFFFF) e preto (#000000). Se você estiver usando a ferramenta de cores por conta própria e selecionar outras cores, talvez seja necessário definir mais recursos de cor.

Quando terminar, o tema ficará assim:

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green</item>  
        <item name="colorPrimaryVariant">@color/green\_dark</item>  
        <item name="colorOnPrimary">@color/white</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue</item>  
        <item name="colorSecondaryVariant">@color/blue\_dark</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

1. Execute seu app no emulador ou em um dispositivo e veja o visual do novo tema no app.



## [5. Tema escuro](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#4)

O modelo de app incluiu um tema claro padrão e também uma variante de [tema escuro](https://developer.android.com/guide/topics/ui/look-and-feel/darktheme?hl=pt-br). Um tema escuro usa cores mais escuras e discretas e:

* Pode reduzir significativamente o consumo de energia (dependendo da tecnologia da tela do dispositivo).
* Melhora a visibilidade para usuários com problemas de visão e que tenham sensibilidade ao brilho da luz.
* Facilita o uso do dispositivo em um ambiente com pouca luz.

## Como escolher cores para o tema escuro

As cores de um tema escuro precisam ser legíveis. Os temas escuros usam uma cor de superfície escura com tons de cores limitados. Para garantir a legibilidade, as cores primárias geralmente são versões menos saturadas das cores primárias do tema claro.

Para oferecer mais flexibilidade e usabilidade em um tema escuro, recomendamos usar tons mais claros (200 a 50) em um tema escuro, em vez do tema de cores padrão (tons de saturação de 900 a 500). Anteriormente, você escolheu verde 200 e azul claro 200 como as cores claras. Para o app, você usará as cores claras como as principais e as cores primárias como as variantes.

## Atualizar a versão escura do tema

1. Abra o arquivo themes.xml (night)( **app> res> values> themes> themes.xml (night)** ).

**Observação:** este arquivo **themes.xml** é diferente do arquivo **themes.xml** anterior. Ele contém a versão escura do tema. Os recursos neste arquivo serão usados quando o **tema escuro** estiver ativado no dispositivo.

1. Mude a colorPrimary para a variante clara da cor primária que você selecionou, @color/green\_light.
2. Mude colorPrimaryVariant para @color/green.
3. Mude colorSecondary para @color/blue\_light.
4. Mude colorSecondaryVariant para @color/blue\_light.

Quando você terminar, seu arquivo themes.xml (night) terá esta aparência:

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Application theme for dark theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green\_light</item>  
        <item name="colorPrimaryVariant">@color/green</item>  
        <item name="colorOnPrimary">@color/black</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue\_light</item>  
        <item name="colorSecondaryVariant">@color/blue\_light</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

1. Agora, as cores originais definidas no arquivo colors.xml, por exemplo, purple\_200, não são mais usadas; então você pode excluí-las.

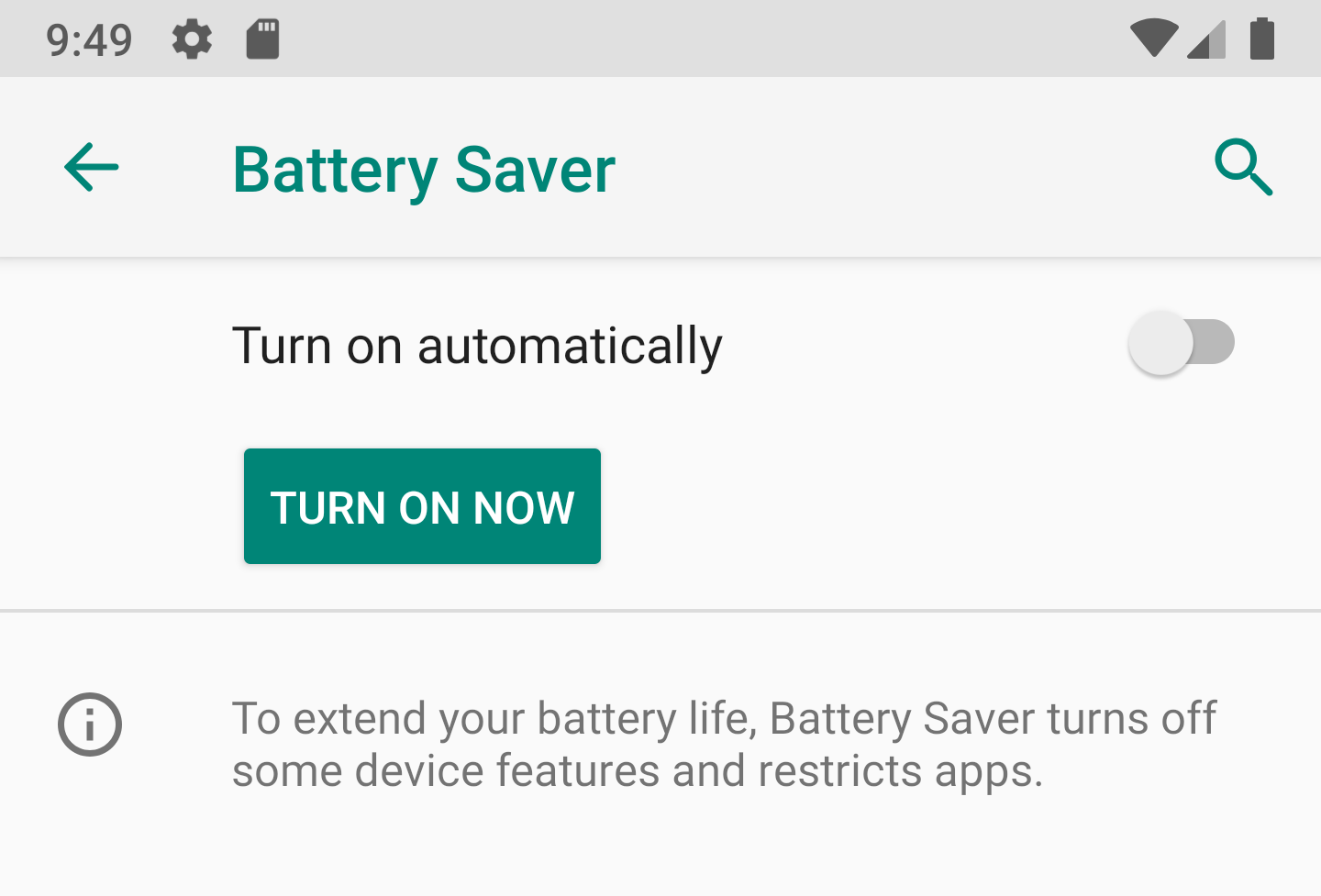
## Testar o tema escuro

Você pode ver como seu tema será exibido no modo escuro ativando-o no dispositivo.

**Observação**: o tema escuro requer um dispositivo ou emulador com a API 28 (Android 9) ou a versão 29 (Android 10) ou mais recente.

### **Para a API 28 (Android 9)**

1. Execute o app novamente.
2. Acesse o app **Config.**.
3. Na seção **Bateria**, localize **Economia de bateria**.

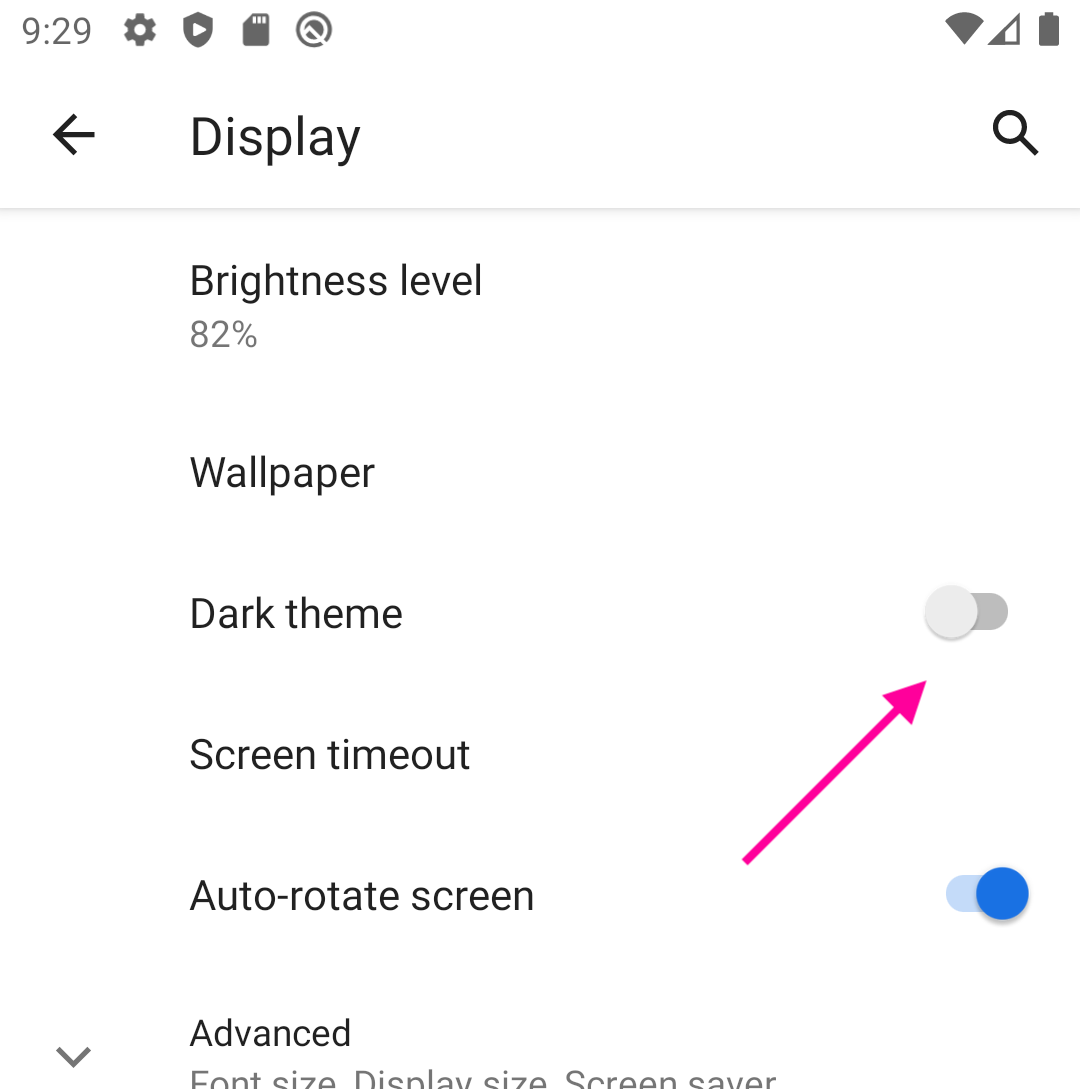


1. Pressione **Ativar agora**.

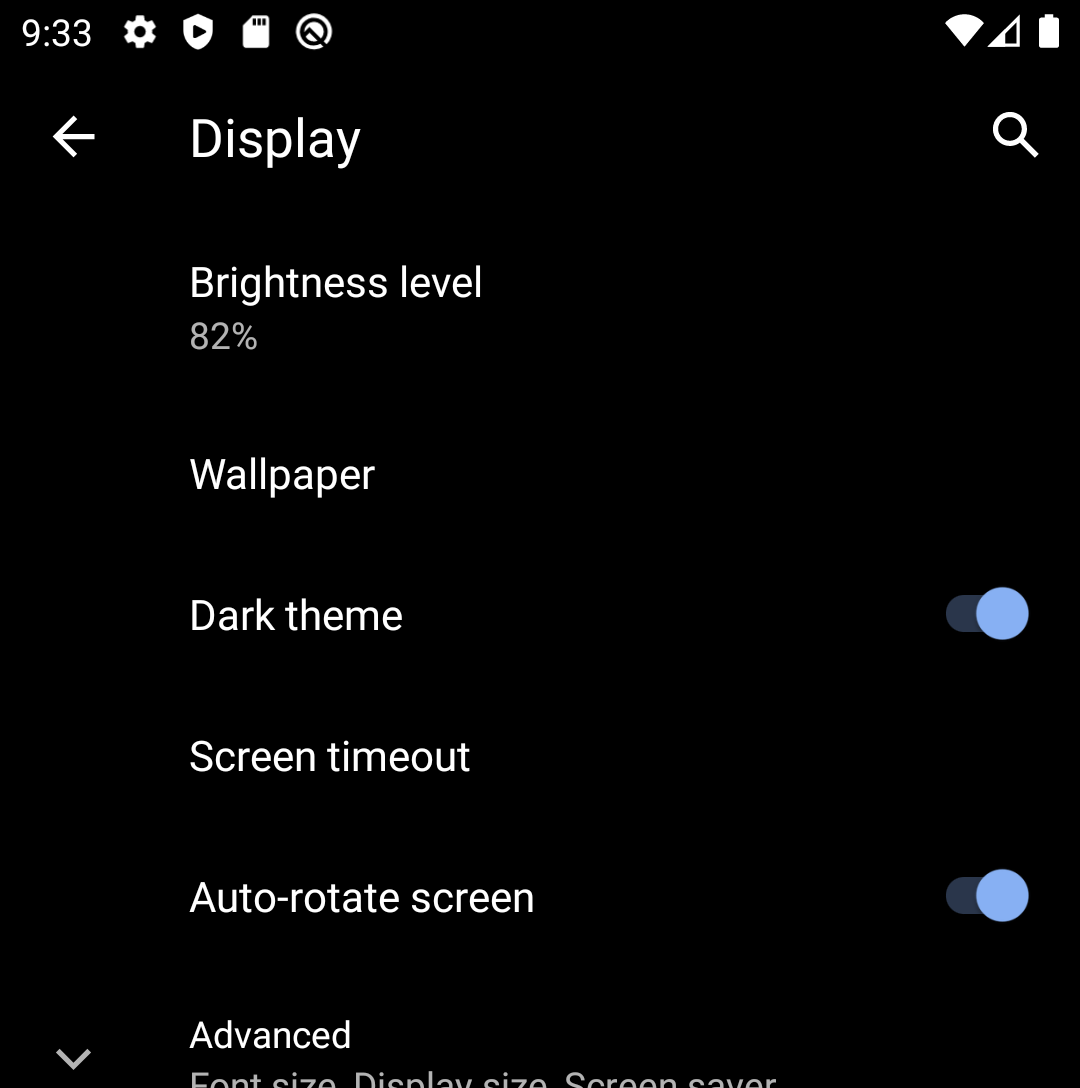
Conclua as etapas abaixo.

### **Para a API 29 (Android 10) ou mais recente**

1. Execute o app novamente.
2. Acesse o app **Config.**.
3. Na seção **Tela**, localize a chave **Tema escuro**.

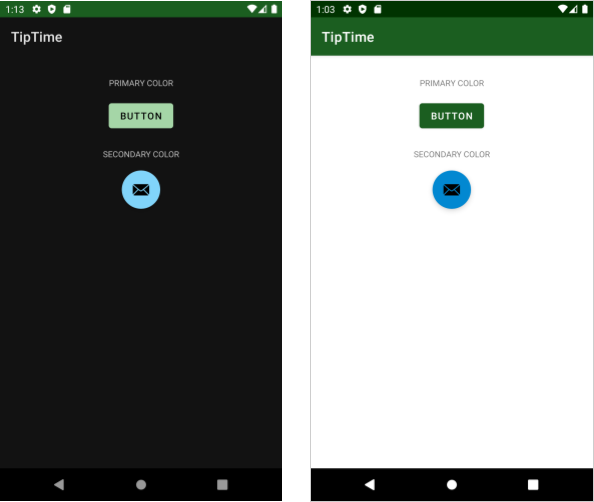


1. Mude a posição da chave **Tema escuro** para "Ativado". O dispositivo passará a usar o modo noturno.



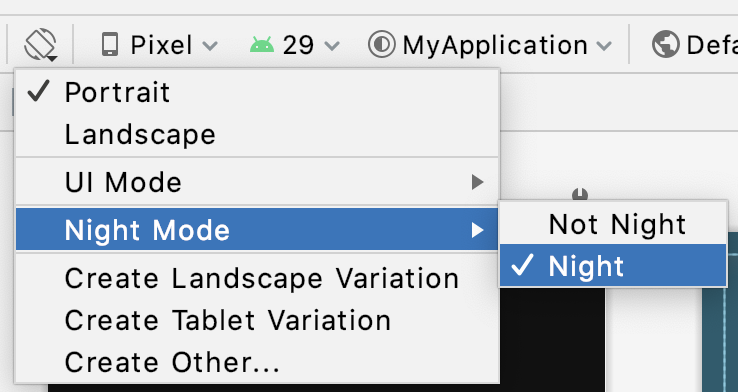
### **Para qualquer API**

1. Volte para seu app e veja as diferenças.



A mudança mais óbvia é que o segundo plano está mais escuro e com texto claro, em vez de um segundo plano claro com texto escuro. Além disso, as cores usadas para os botões são menos vibrantes no tema escuro do que no tema claro.

Parabéns! Você criou um novo tema para o app com um tema claro e um escuro.

**Observação:** você também pode ver como seu tema será exibido usando o **Design Editor** do Android Studio. O menu **Orientation for Preview** contém configurações para **Night** ou **Not Night**. 

## [6. Código da solução](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#5)

Este codelab se concentrou na personalização das cores do tema, mas há muitos atributos que podem ser personalizados, incluindo texto, forma, estilo de botões e muito mais. Confira este artigo para ver outras formas de personalizar um tema de app. [Estilo do Android: atributos de tema comuns](https://medium.com/androiddevelopers/android-styling-common-theme-attributes-8f7c50c9eaba) (em inglês).

Veja abaixo o código da solução para este codelab.

colors.xml( **app> res> values> colors.xml** )

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="black">#FF000000</color>  
    <color name="white">#FFFFFFFF</color>  
    <color name="green">#1B5E20</color>  
    <color name="green\_dark">#003300</color>  
    <color name="green\_light">#A5D6A7</color>  
    <color name="blue">#0288D1</color>  
    <color name="blue\_dark">#005B9F</color>  
    <color name="blue\_light">#81D4FA</color>  
</resources>

values/themes.xml( **app> res> valores> temas> themes.xml** )

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green</item>  
        <item name="colorPrimaryVariant">@color/green\_dark</item>  
        <item name="colorOnPrimary">@color/white</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue</item>  
        <item name="colorSecondaryVariant">@color/blue\_dark</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

values-night/themes.xml( **app> res> valores> temas> themes.xml (noite)** )

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Application theme for dark theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green\_light</item>  
        <item name="colorPrimaryVariant">@color/green</item>  
        <item name="colorOnPrimary">@color/black</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue\_light</item>  
        <item name="colorSecondaryVariant">@color/blue\_light</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>