



Todos podem programar

# Guia SwiftUI ILUSTRADO

— Por Lidiane Chen e Matheus S. Moreira

1<sup>a</sup> Edição

# Motivação

## Todos podem programar

Este documento tem como objetivo mostrar que todos podem programar e que o **SwiftUI é um grande aliado para você que está iniciando no mundo da programação.**



No processo de desenvolvimento, normalmente consultamos a documentação oficial, que é a fonte de informação mais confiável e completa acerca das informações que precisamos. Porém, há duas questões na **linguagem** dessa documentação que dificultam a aprendizagem: **o conteúdo está na língua inglesa**, sendo que muitas pessoas não possuem domínio desse idioma, e **há a presença de termos técnicos que dificultam o entendimento do conteúdo**, pois muitos deles não são de conhecimento dos estudantes iniciantes.

Além disso, em grande parte da documentação o conteúdo é apenas textual, isto é, muitos exemplos não possuem suporte visual para o que está sendo explicado.



# Sumário

O que você vai encontrar

=

1

## INTRODUÇÃO

O protocolo *View* e seus modificadores

2

## TEXT

Como manipular um dos componentes mais básicos de interface

3

## STACKS

Posicione componentes verticalmente, horizontalmente ou sobrepostos

4

## PADDING

Aplique pequenos ajustes de espaçamento que farão uma grande diferença

5

## IMAGE

Conheça as diferentes formas de inserir uma imagem na tela

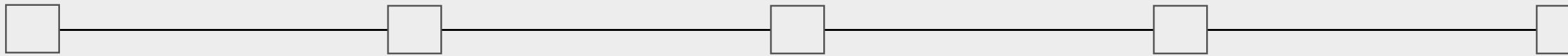
6

## BUTTON

Disponibilize as ações que o usuário precisa executar

# Sumário detalhado

O conteúdo de cada módulo



## TEXT

- Tamanhos de texto
- Pesos de fonte
- Aplicação de cor
- Modo escuro
- Alinhamentos
- Espaçamento entre linhas
- Limite de linhas

## STACK

- Stacks
- Alinhamentos nas Stacks
- Espaçamento
- Stacks aninhadas
- Espaço em branco
- Linha divisória

## PADDING

- Posições
- Valores numéricos

## IMAGES

- Como inserir uma imagem
- Tamanho adaptável
- Aspect Ratio
- Ícones

## BUTTON

- Ações e aparência

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Aprenda a programar!")
            .font(.title)
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```



## Introdução Views e Modifiers

Para construirmos interfaces com o SwiftUI, nós usaremos o que chamamos de *View*, que é todo componente de interface presente no projeto, sendo assim, tudo aquilo que estamos vendo na tela.

Toda *View* no SwiftUI contém *modifiers*, isto é, modificadores que nos permitem personalizar a aparência dos componentes de interface. Desta forma, é possível aplicar o que é proposto pela a identidade visual do seu projeto.

Vamos conhecer alguns recursos desta nova tecnologia da Apple mais a fundo?

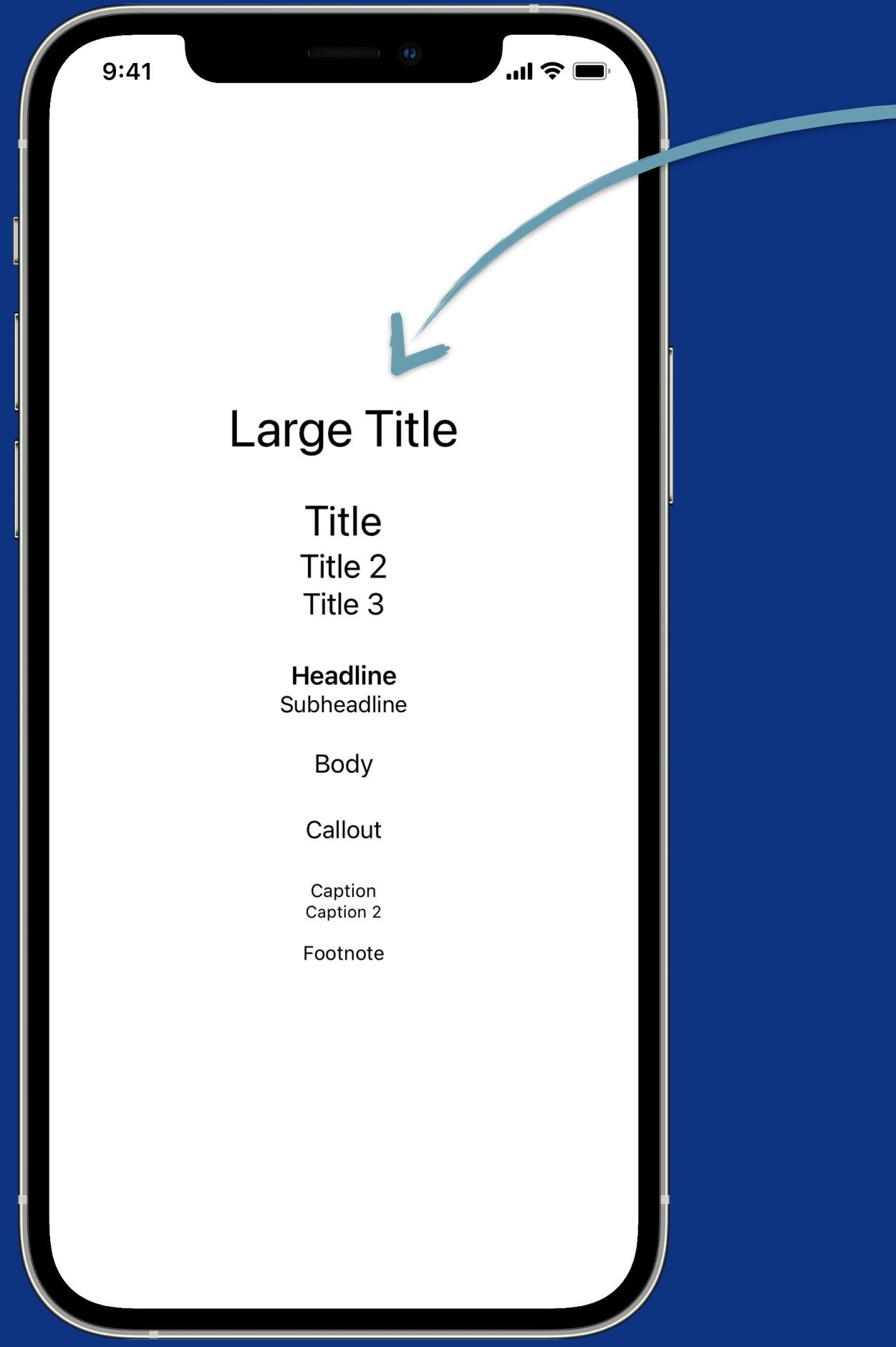
# Text

==

Um *Text* no SwiftUI, como você pode esperar, é uma *view* que nos exibe um texto.

Para as coisas ficarem mais interessantes, vamos explorar diferentes formas de formatar um texto com seus modificadores.

- Tamanhos de texto  
.font
- Pesos de fonte  
.fontWeight
- Aplicação de cor  
.foregroundColor
- Modo escuro  
.foregroundColor(.primary | .secondary)
- Alinhamentos  
.multilineTextAlignment
- Espaçamento entre linhas  
.lineSpacing
- Limite de linhas  
.lineLimit



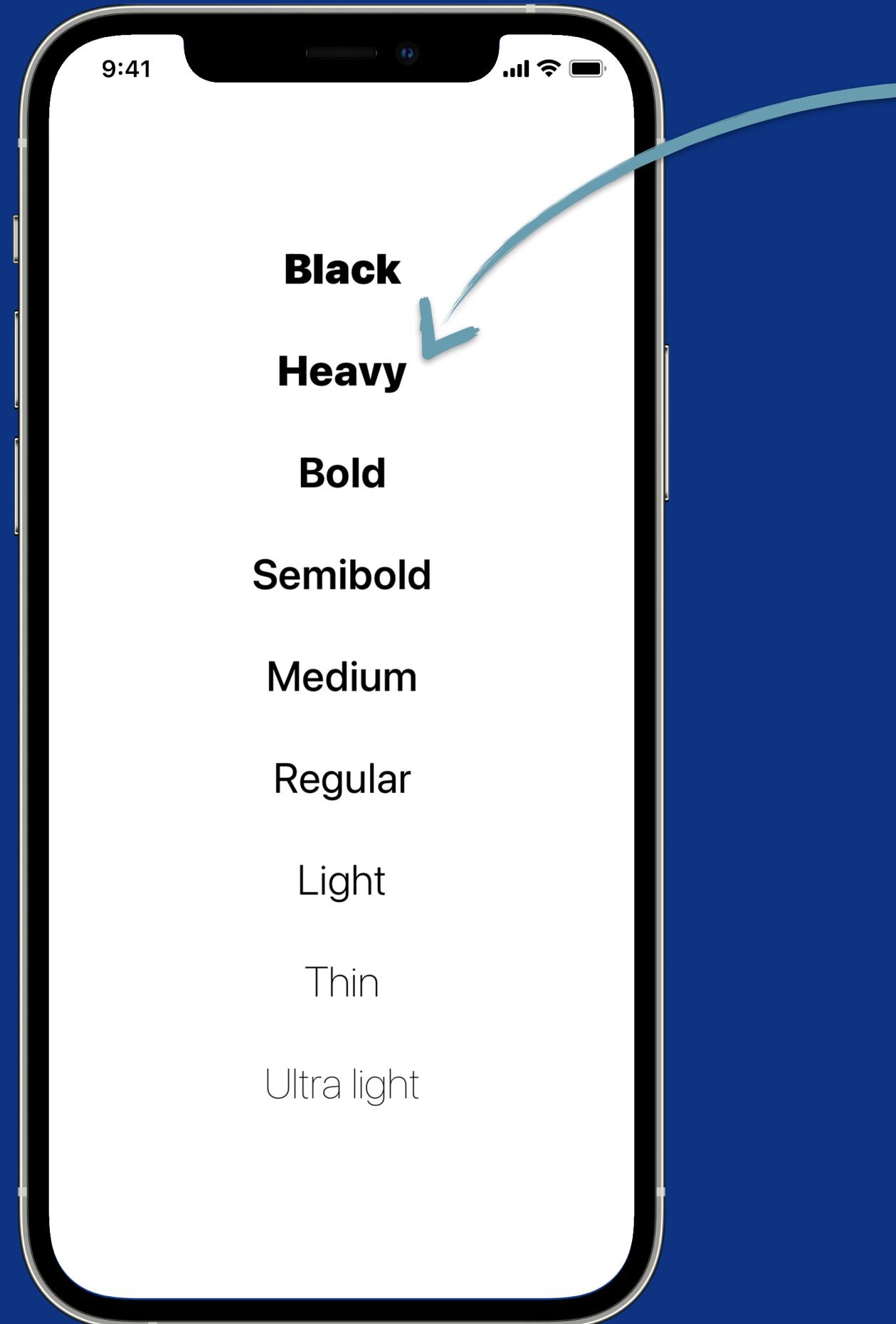
## Tamanhos de texto

### .font

As fontes de um texto estão disponíveis em 11 tamanhos diferentes e podem ser encontradas pelo modificador `.font`.

Tamanhos disponíveis: `.largeTitle | .title | .title2 | .title3 | .headline | .subheadline | .body | .callout | .caption | .caption2 | .footnote`

Para aplicar a fonte no seu projeto basta substituir o valor sublinhado.



```
Text("Heavy\n")  
.fontWeight(.heavy)  
.font(.title)
```

## Pesos de fonte

### .fontWeight

As fontes podem ter diferentes pesos (ou espessuras), tornando-os mais ou menos destacados na leitura, e os declaramos com o modificador `.fontWeight`.

São eles: `.black` | `.heavy` | `.bold` | `.semibold`  
| `.medium` | `.regular` | `.light` | `.thin`  
| `.ultraLight`

Para aplicar o `.fontWeight` no seu projeto basta substituir o valor sublinhado.



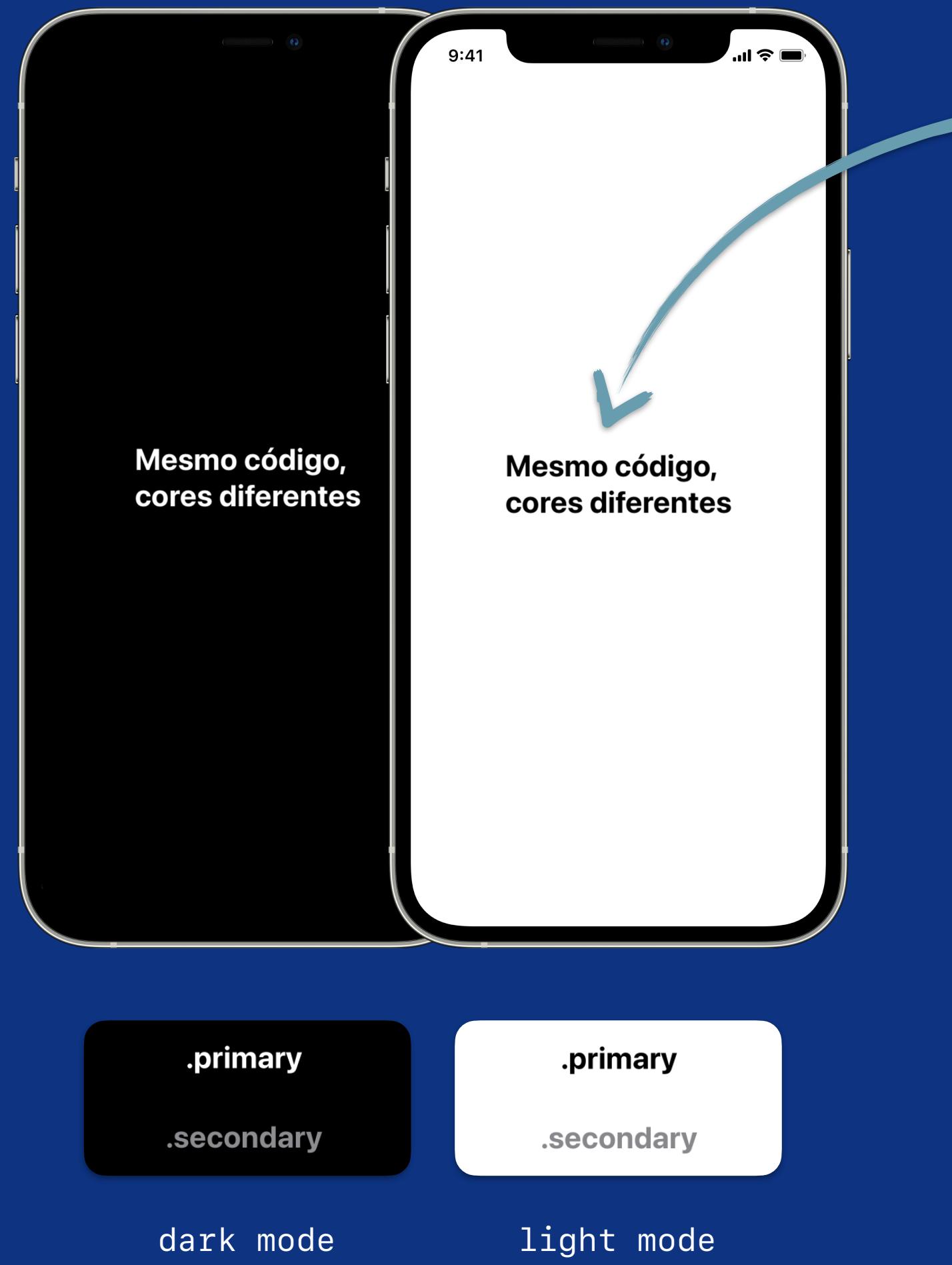
```
Text(" Branco ")  
.foregroundColor(.white)  
.fontWeight(.bold)  
.background(Color.black)  
.font(.title)
```

## Aplicação de cor .foregroundColor

O SwiftUI possui o componente *Color* para aplicarmos cores em diferentes contextos. Para alterar a cor de um texto, usamos os modificadores de *.foregroundColor*.

Cores nativas: `.yellow | .blue | .white | .gray | .orange | .black | .pink | .purple | .green | .red`

Para aplicar a cor da fonte no seu projeto basta substituir o valor sublinhado.



```
Text("Mesmo código,\ncores diferentes")  
    .foregroundColor(.primary)  
    .font(.title)  
    .fontWeight(.bold)  
    .padding()
```

## modo escuro

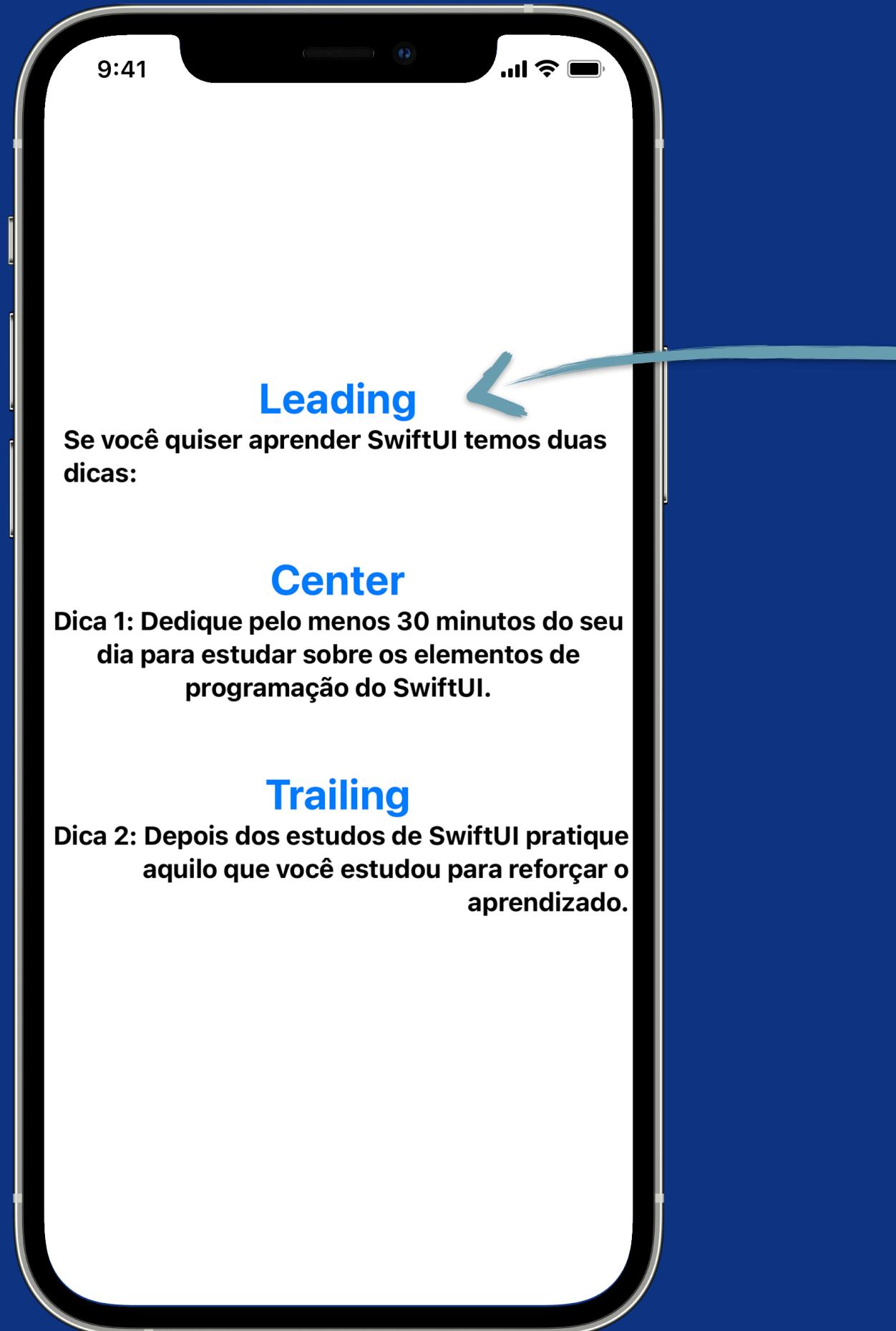
.primary | .secondary

Após o lançamento do iOS13, os aplicativos tornaram-se capazes de comportar o modo escuro (*dark mode*), o que exige uma adaptação da cor do texto para que ele permaneça legível. É o caso das cores *.primary* e *.secondary*.

Essas cores se adaptam automaticamente ao modo escuro e ao modo claro da tela.

Cores disponíveis: .primary | .secondary

Para aplicar a cor no seu projeto basta substituir o valor sublinhado.



```
 VStack {  
     Text("Leading")  
         .foregroundColor(.blue)  
         .font(.title)  
         .fontWeight(.bold)  
  
     Text("Se você quiser aprender SwiftUI temos duas dicas:  
\n\n")  
         .fontWeight(.bold)  
         .multilineTextAlignment(.leading)  
 }
```

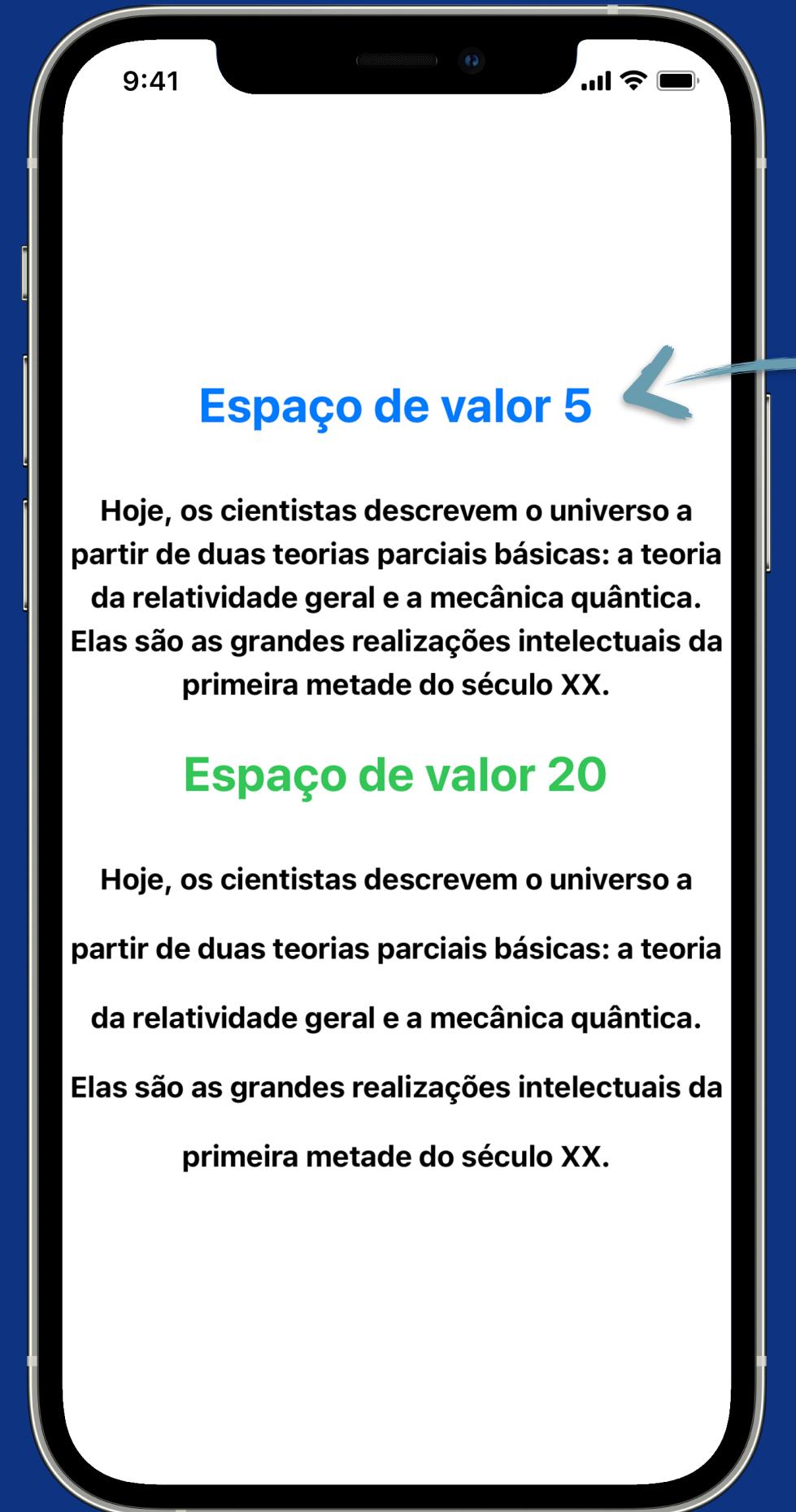
# Alinhamentos

## .multilineTextAlignment

Quando seu texto contém múltiplas linhas, você pode deixá-lo alinhado na esquerda, na direita ou centralizado. Isso é feito com o modificador `.multilineTextAlignment`.

Alinhamentos disponíveis: `.leading` | `.center` | `.trailing`

Para aplicar o alinhamento de texto no seu projeto basta substituir o valor sublinhado. O alinhamento padrão de um texto é o `leading` (esquerda).



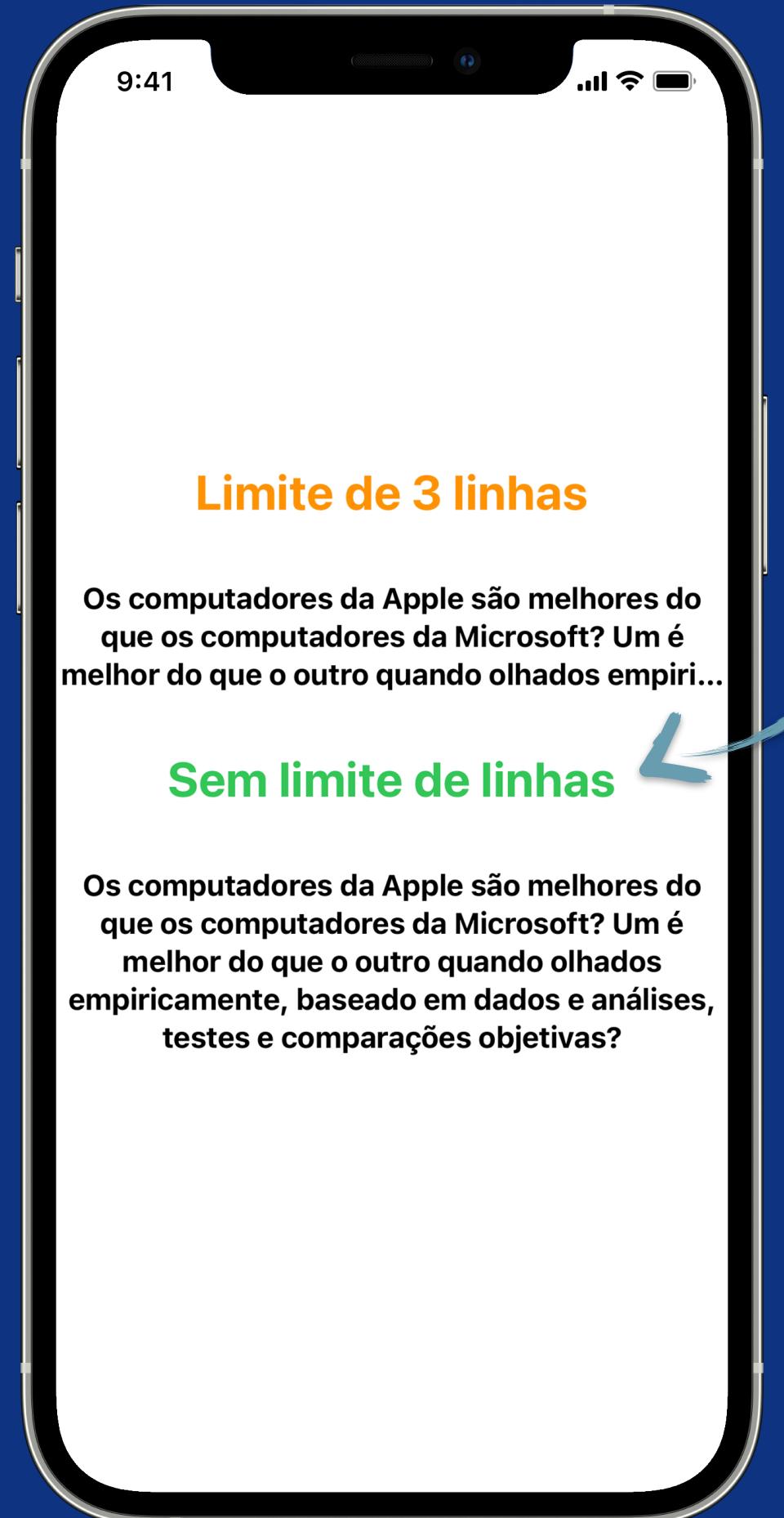
```
 VStack {  
     Text("Espaço de valor 5\n")  
         .foregroundColor(.blue)  
         .font(.title)  
         .fontWeight(.bold)  
  
     Text("Hoje, os cientistas descrevem o universo a partir de duas teorias parciais básicas: a teoria da relatividade geral e a mecânica quântica. Elas são as grandes realizações intelectuais da primeira metade do século XX.\n")  
         .fontWeight(.bold)  
         .multilineTextAlignment(.center)  
         .lineSpacing(5)  
 }
```

## Espaçamento entre linhas .lineSpacing

Quando se tem um texto com múltiplas linhas, é possível alterar o espaçamento entre elas. Fazemos isto passando um valor numérico para o modificador `.lineSpacing`.

Valores possíveis: qualquer valor numérico

Para aplicar a fonte no seu projeto basta substituir o valor sublinhado.



```
 VStack {  
     Text("\nSem limite de linhas\n")  
         .foregroundColor(.green)  
         .font(.title)  
         .fontWeight(.bold)  
  
     Text("Os computadores da Apple são melhores do que os  
computadores da Microsoft? Um é melhor do que o  
outro quando olhados empiricamente, baseado em dados  
e análises, testes e comparações objetivas?\n")  
         .fontWeight(.bold)  
         .multilineTextAlignment(.center)  
         .lineLimit(nil)  
 }
```

## Limite de linhas

### .lineLimit

Quando seu texto contém múltiplas linhas, você pode desejar estabelecer um número máximo de linhas a serem exibidas, independentemente do tamanho do texto. Fazemos isto com o modificador `.lineLimit`.

Valores: `nil` (sem limite de linhas) | qualquer valor numérico (e.g. 3)

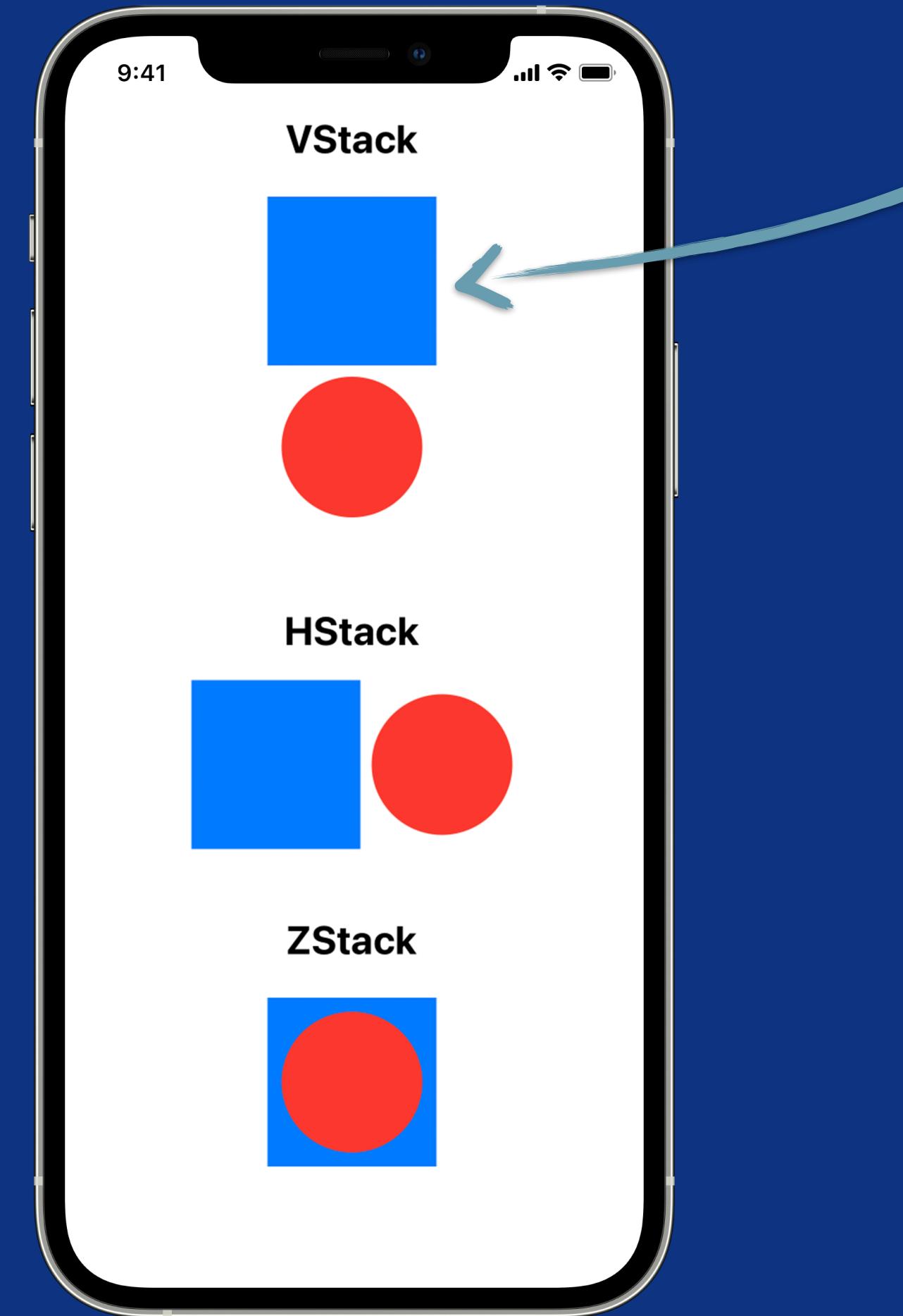
Para aplicar o modificador no seu projeto basta substituir o valor sublinhado.

# Stack

=

As *stacks* são *views* do SwiftUI que usamos para empilhar e alinhar outras *views*. Por padrão, o conteúdo é sempre alinhado no centro de uma stack e essa organização vai se adaptando à medida que vamos inserindo novos elementos de interface em posições específicas.

- Stacks  
VStack | HStack | ZStack
- Alinhamentos nas Stacks  
alignment(:)
- Espaçamento  
spacing(:)
- Stacks Aninhadas  
Uma stack dentro de outra
- Espaço em branco  
Spacer
- Linha divisória  
Divider



```
VStack {  
    Rectangle()  
        .frame(width: 120, height: 120)  
        .foregroundColor(.blue)  
  
    Circle()  
        .frame(width: 100, height: 100)  
        .foregroundColor(.red)  
}.padding()
```

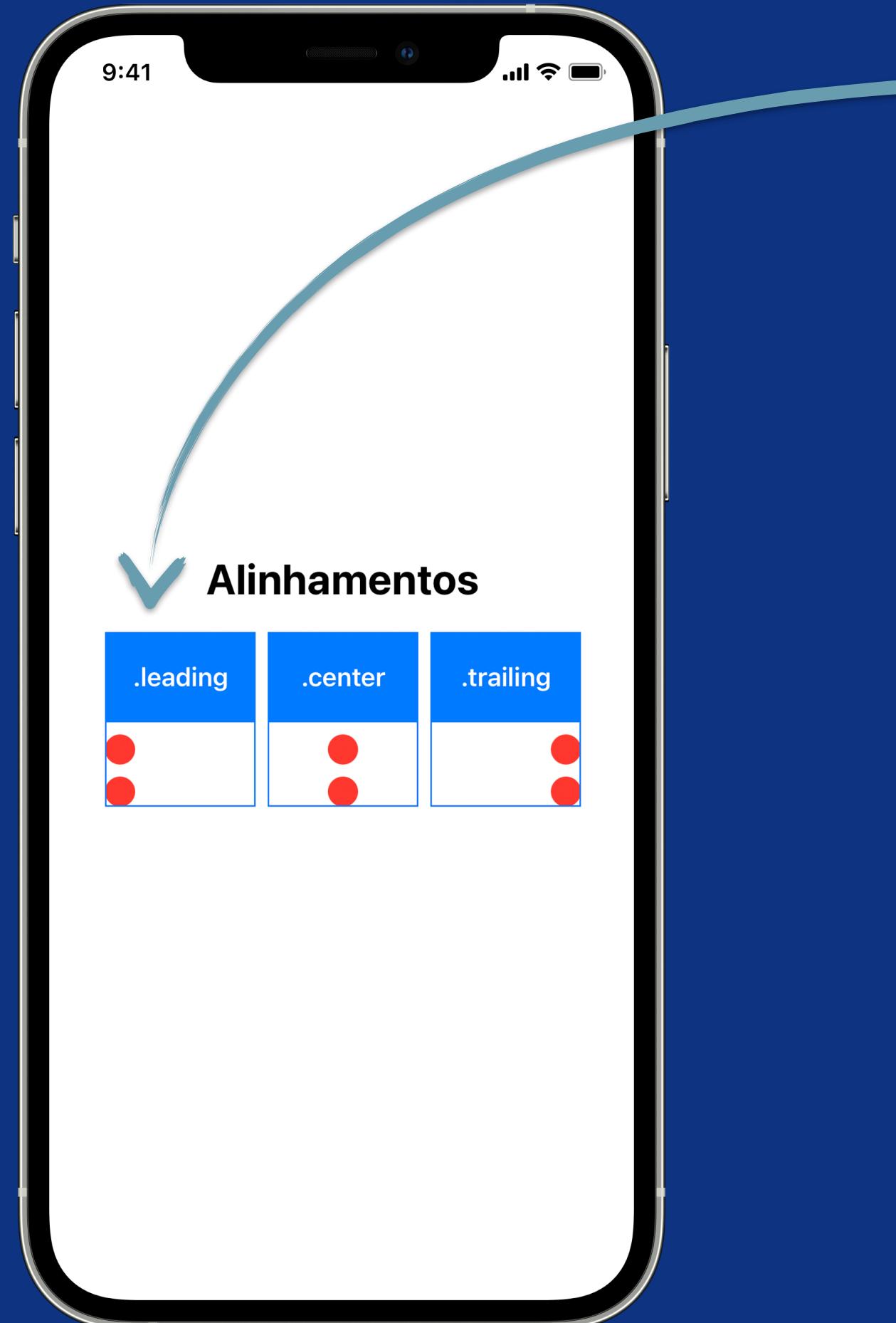
## Stacks

[VStack](#) | [HStack](#) | [ZStack](#)

As stacks são responsáveis pelo empilhamento de views, podendo este ser no sentido vertical, horizontal e frontal.

Tipos de Stacks: [VStack](#) (vertical) | [HStack](#) (horizontal) | [ZStack](#) (frontal)

Para aplicar as stacks no seu projeto basta substituir no local sublinhado.



```
 VStack(alignment: .leading) {  
     ZStack { } // retângulo azul com texto  
     Circle()  
         .frame(width: 20, height: 20)  
         .foregroundColor(.red)  
     Circle()  
         .frame(width: 20, height: 20)  
         .foregroundColor(.red)  
 }
```

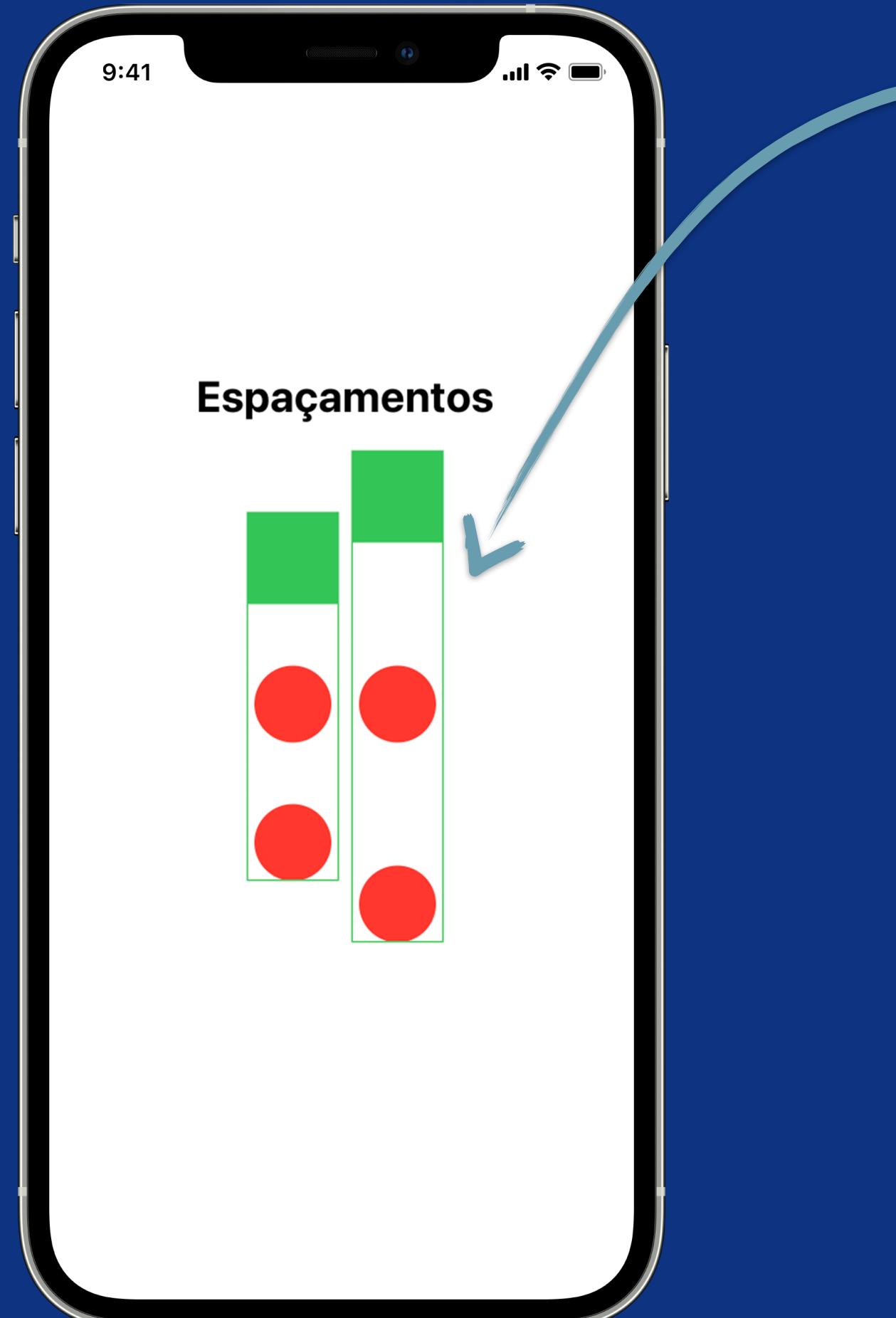
## Alinhamentos nas Stacks

### `alignment( : )`

Com o `.alignment` somos capazes de escolher em que direção da stack os componentes estarão alinhados.

Valores: `.leading` | `.center` | `.trailing`

Para aplicar o alinhamento no seu projeto basta inserir o valor no local sublinhado.



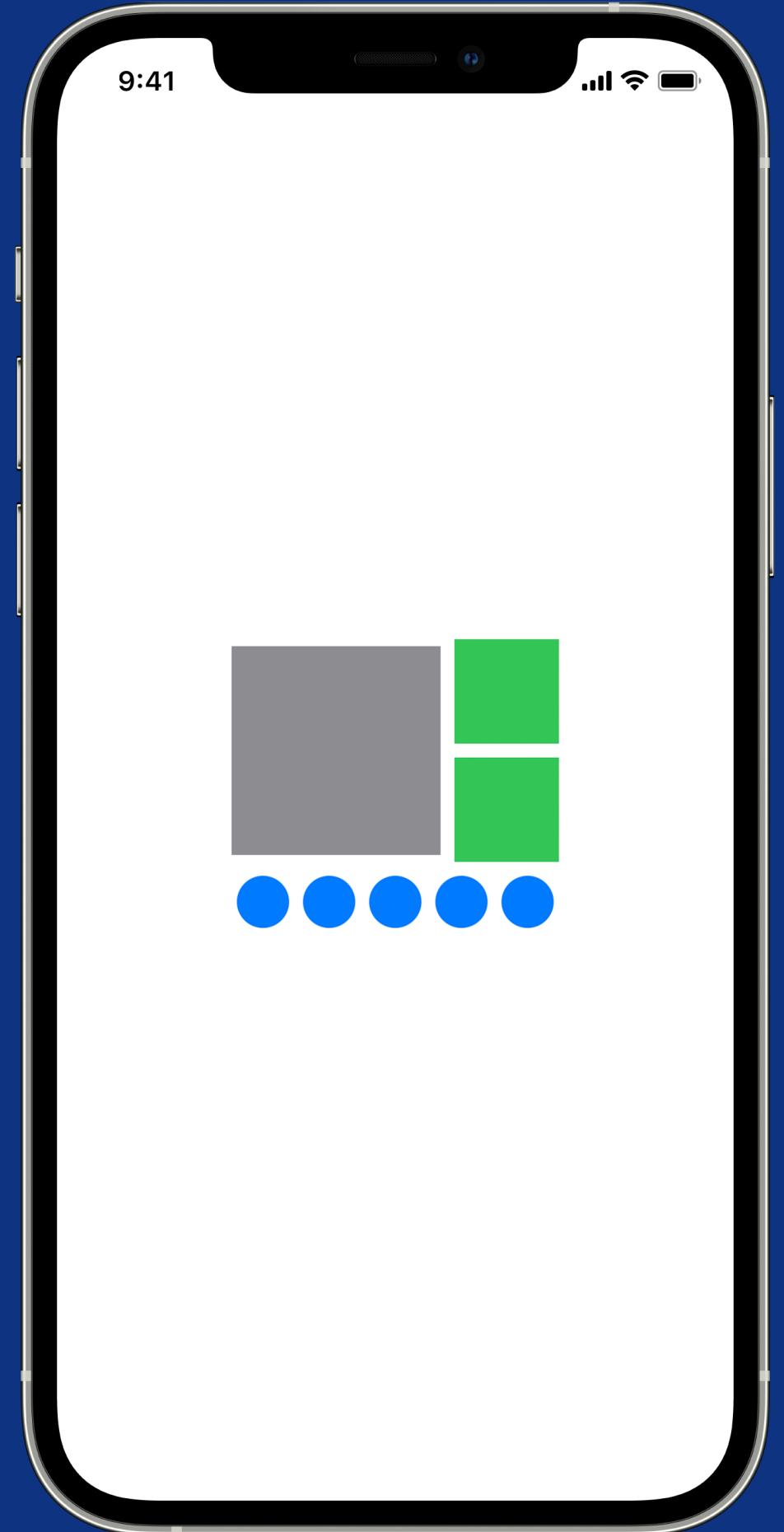
```
VStack(spacing: 80) {  
    Rectangle()  
        .frame(width: 60, height: 60)  
        .foregroundColor(.green)  
  
    Circle()  
        .frame(width: 50, height: 50)  
        .foregroundColor(.red)  
  
    Circle()  
        .frame(width: 50, height: 50)  
        .foregroundColor(.red)  
}.border(Color.green)
```

## Espaçamento spacing(:)

O *spacing*, como o próprio nome diz, modifica o tamanho do espaço entre os componentes dentro de uma stack.

Parâmetro: valor numéricico de sua preferência

Para aplicar o espaçamento no seu projeto basta inserir o valor no local sublinhado.



```
 VStack {  
     HStack {  
         Rectangle()  
             .frame(width: 120, height: 120)  
             .foregroundColor(.gray)  
     }  
     VStack { } // Quadrados verdes  
 }  
 HStack { } // Círculos azuis  
 }
```

## Stacks aninhadas

Uma stack dentro de outra

Para criar uma organização de interface mais complexa, podemos inserir uma stack dentro de outra.

Observe o exemplo ao lado.



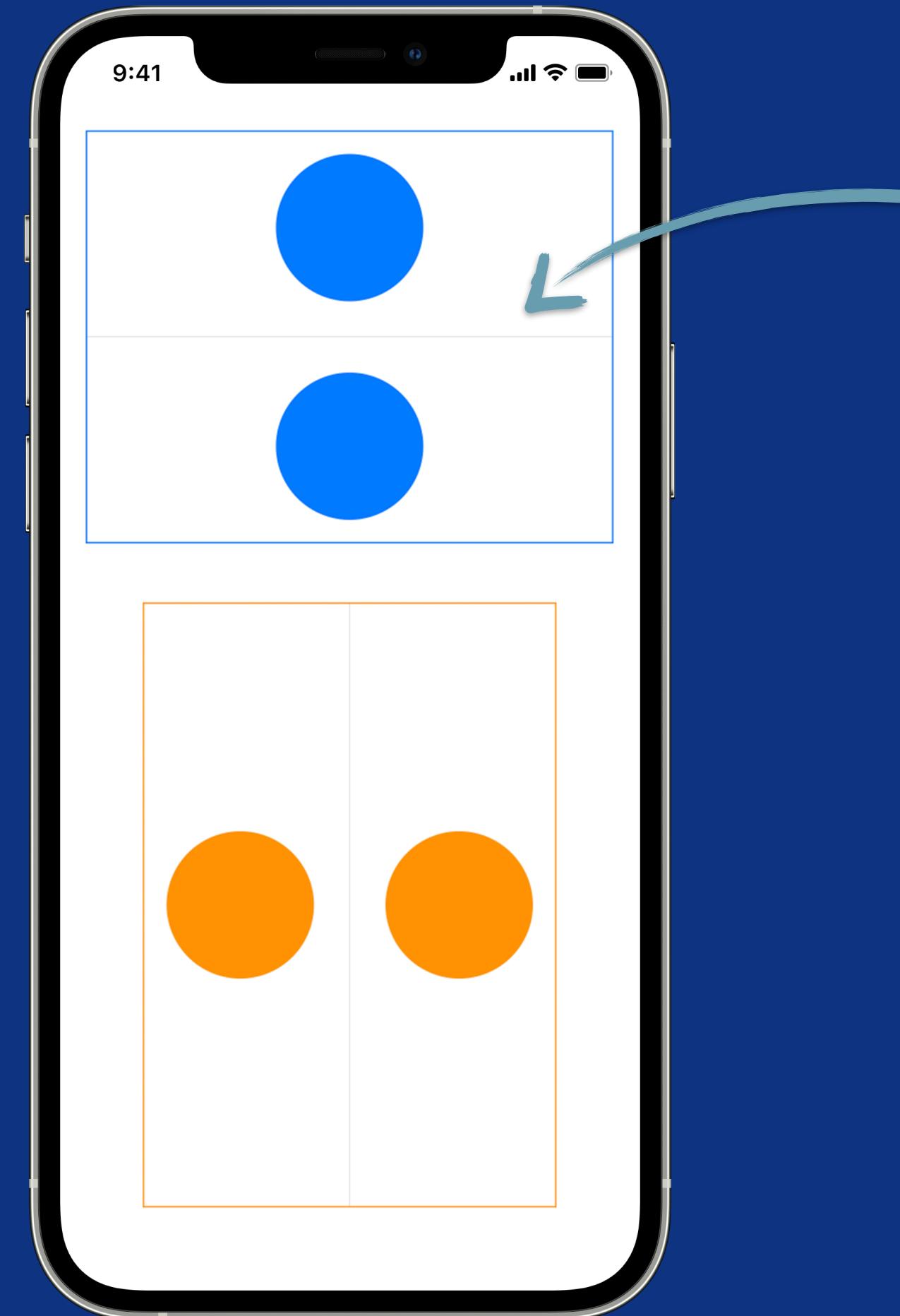
```
VStack {  
    Text("Spacer entre")  
        .font(.title)  
        .fontWeight(.bold)  
    Spacer()  
    Text("dois textos")  
        .font(.title)  
        .fontWeight(.bold)  
}
```

```
VStack {  
    Spacer()  
    Text("Spacer  
inserido\nacima do texto")  
        .font(.title)  
        .fontWeight(.bold)  
}
```

## Espaço em branco Spacer

Quando se trata de posicionar um ou mais componentes de interface na tela, o SwiftUI toma o centro da tela ou de uma stack como localização padrão, no entanto, você pode desejar posicionar um componente em outro lugar.

Para fazer isso, basta incluir um `Spacer` no local desejado. Com isso, ele irá ocupar o maior espaço possível onde foi inserido, deslocando o componente para o lado oposto de onde ele está.



```
 VStack {  
     Circle()  
         .frame(width: 100, height: 100)  
         .foregroundColor(.blue)  
         .padding()  
  
     Divider()  
  
     Circle()  
         .frame(width: 100, height: 100)  
         .foregroundColor(.blue)  
         .padding()  
 } .border(Color.blue).padding()
```

## Linha divisória Divider

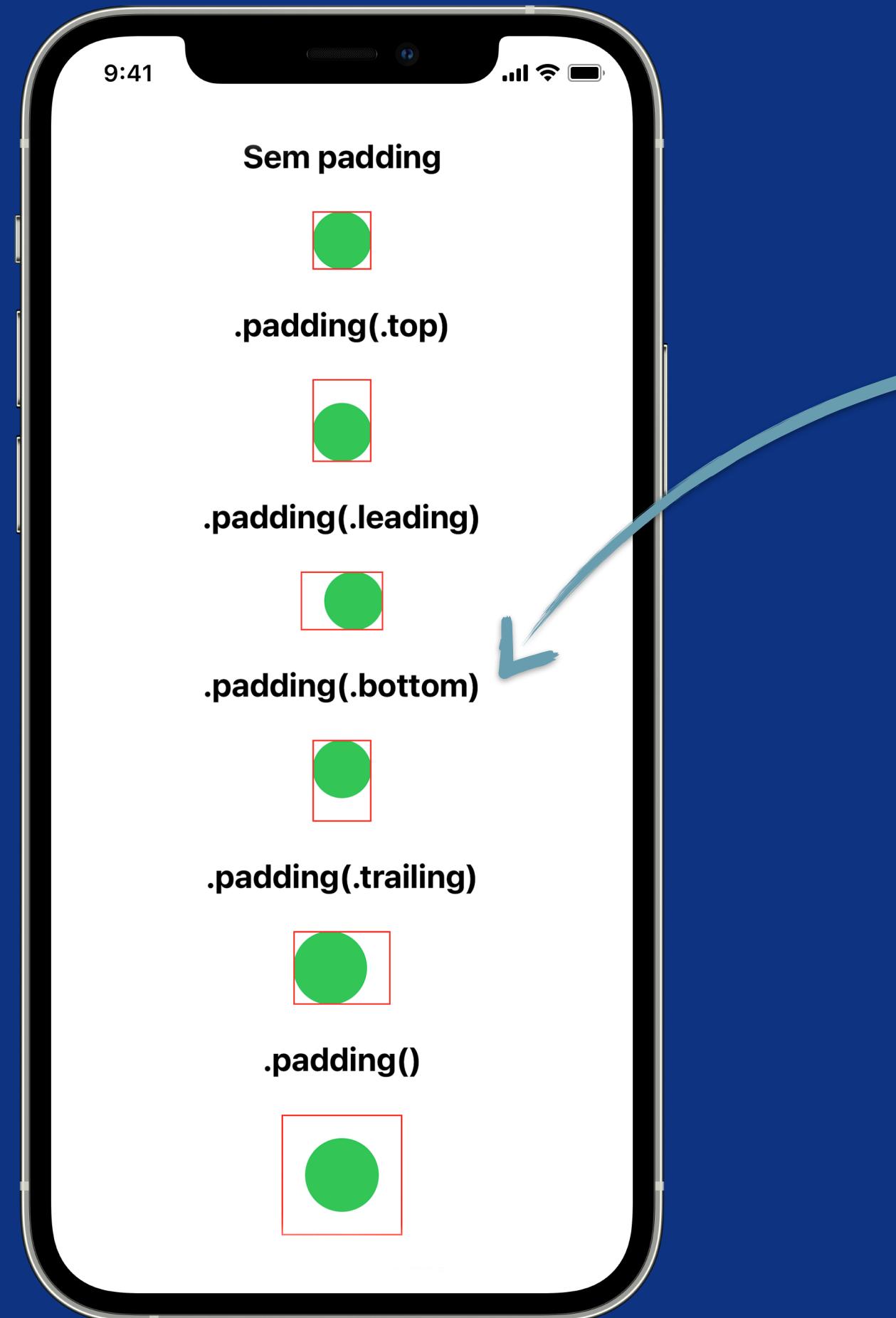
Um *Divider* é uma linha reta acinzentada usada para dividir dois componentes. Ela terá uma orientação vertical ou horizontal dependendo da stack em que for inserida.

# Padding

Um *.padding* é modificador usado quando desejamos preencher um pequeno espaço nos arredores de uma view.

○ Posições  
.padding

○ Valores numéricos  
.padding | Padding com tamanho personalizado



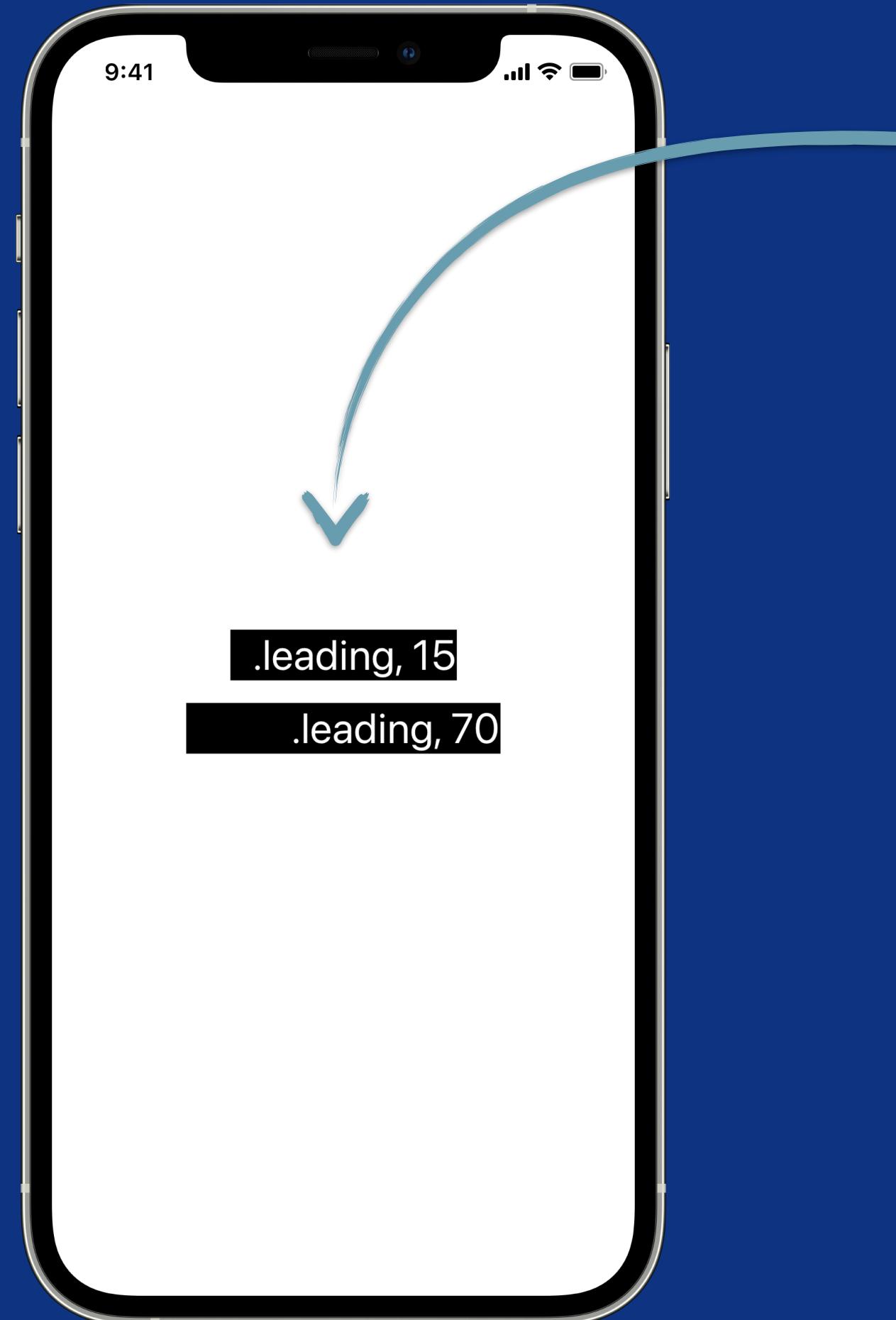
```
 VStack {  
     Text(".padding(.bottom)")  
         .font(.title2)  
         .fontWeight(.bold)  
         .padding()  
  
     Circle()  
         .foregroundColor(.green)  
         .frame(width: 40, height: 40)  
         .padding(.bottom)  
         .border(Color.red)  
 }
```

## Posições .padding

Podemos aplicar um padding no topo (`.top`), na base (`.bottom`), na esquerda (`.leading`) ou na direita (`.trailing`). Também é possível aplicá-lo na esquerda e na direita simultaneamente (`.horizontal`), bem como verticalmente (`.vertical`). Caso a posição não seja informada, o padding é aplicado em todos os lados. Observe os exemplos com o círculo verde destacado.

Valores: `.top` | `.bottom` | `.leading` | `.trailing`  
| `.horizontal` | `.vertical`

Para aplicar o modificador no seu projeto basta substituir o valor sublinhado.



```
 VStack {  
     Text(\".leading, 15\")  
         .font(.title)  
         .foregroundColor(.white)  
         .padding(.leading, 15)  
         .background(Color.black)  
  
     Text(\".leading, 70\")  
         .font(.title)  
         .foregroundColor(.white)  
         .padding(.leading, 70)  
         .background(Color.black)  
         .offset(y: 15)  
 }
```

## Valores numéricicos .padding

Ao inserir um *.padding*, o Xcode automaticamente aplica um espaçamento de valor 20. Para aplicar um padding de tamanho específico, também é possível. Basta inserir um valor numérico, positivo ou negativo, logo após a vírgula.

Valores: qualquer valor numérico

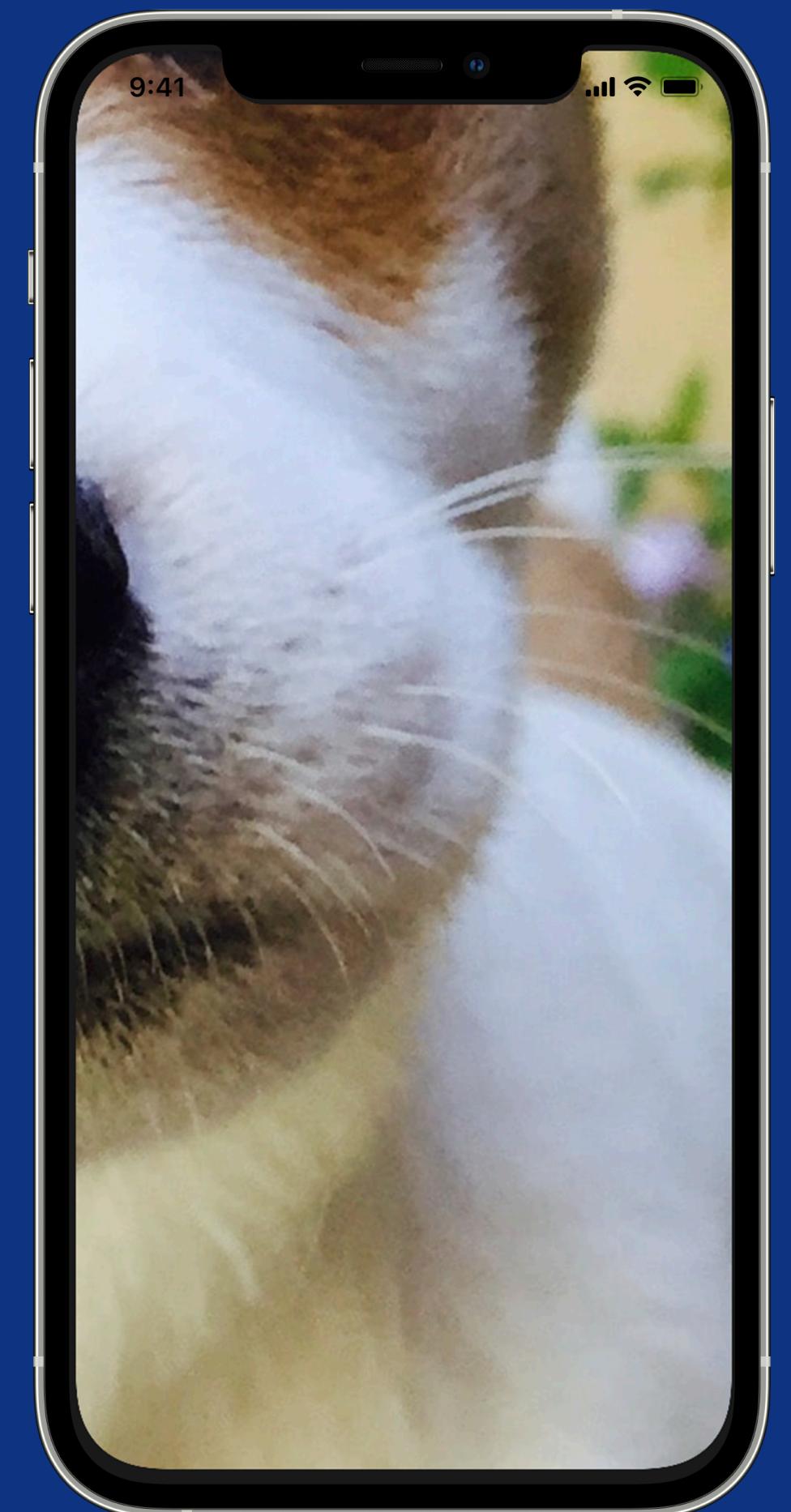
Para aplicar o padding no seu projeto basta substituir o valor sublinhado.



# Image

Existem diferentes formas de inserir uma imagem na tela com o SwiftUI, todas elas através do componente *Image*. Essa inserção deverá levar em conta alguns ajustes para que ela seja exibida conforme o que você tem em mente.

- Como inserir uma imagem  
*Image*
- Tamanho adaptável  
.resizable
- Aspect Ratio  
Fit e Fill
- Ícones  
SF Symbols



`Image("doguinho")`

Assets.xcassets › doguinho

## Como inserir uma imagem Image

Chamar uma imagem é um processo simples, basta chamar o nome da imagem exatamente como foi nomeada na pasta de assets do projeto do Xcode. Porém é possível que seja necessário aplicar alguns modificadores para visualizar a imagem da maneira que se espera.

A seguir será mostrado com aplicar alguns modificadores do *Image*.



```
Image("doguinho")  
    .resizable()
```

## Tamanho adaptável .resizable

O tamanho de uma imagem normal, especialmente aquelas que possuem alta resolução, pode não caber na tela. Para que isso se resolva, é necessário aplicar o modificador `.resizable`.

Este modificador é capaz de fazer uma imagem caber na tela, porém ele não respeita suas proporções originais. Para que elas se mantenham, é preciso inserir outro modificador, como você pode ver na página a seguir.



```
Image("doguinho")
    .resizable()
    .aspectRatio(contentMode: .fill)
```

## Aspect ratio Fit e Fill

O `.aspectRatio` é capaz de receber dois modos de preservação das proporções, um que faz com que a imagem caiba na tela e outro que não, são eles: `.fit` e `.fill`, respectivamente.

Modificadores: `.fit` | `.fill`

Para aplicar o `.aspectRatio` no seu projeto basta substituir o valor sublinhado.



```
HStack {  
    Image(systemName: "thermometer.sun")  
        .font(.largeTitle)  
  
    Image(systemName: "thermometer.sun.fill")  
        .foregroundColor(.blue)  
        .font(.largeTitle)  
  
    Image(systemName: "thermometer.sun.fill")  
        .renderingMode(.original)  
        .font(.largeTitle)  
} .padding()
```

## Ícones SF Symbols

Com o Image também podemos inserir ícones na interface, como é o caso dos [SF Symbols](#), uma coleção fornecida pela Apple para desenvolvedores. Para usá-los, é preciso instalá-los na sua máquina.

Com isso feito, você terá acesso não apenas aos ícones mas também aos seus nomes, que são fornecidos para o Image através do parâmetro `systemName`. Estes ícones são customizáveis, por isso você pode alterar seus tamanhos com o modificador `.font` e suas cores com `.foregroundColor`.

Alguns SF Symbols, lançados com o iOS14, já possuem cores próprias, e basta inserir `.renderingMode(.original)` para visualizá-las.

# Button

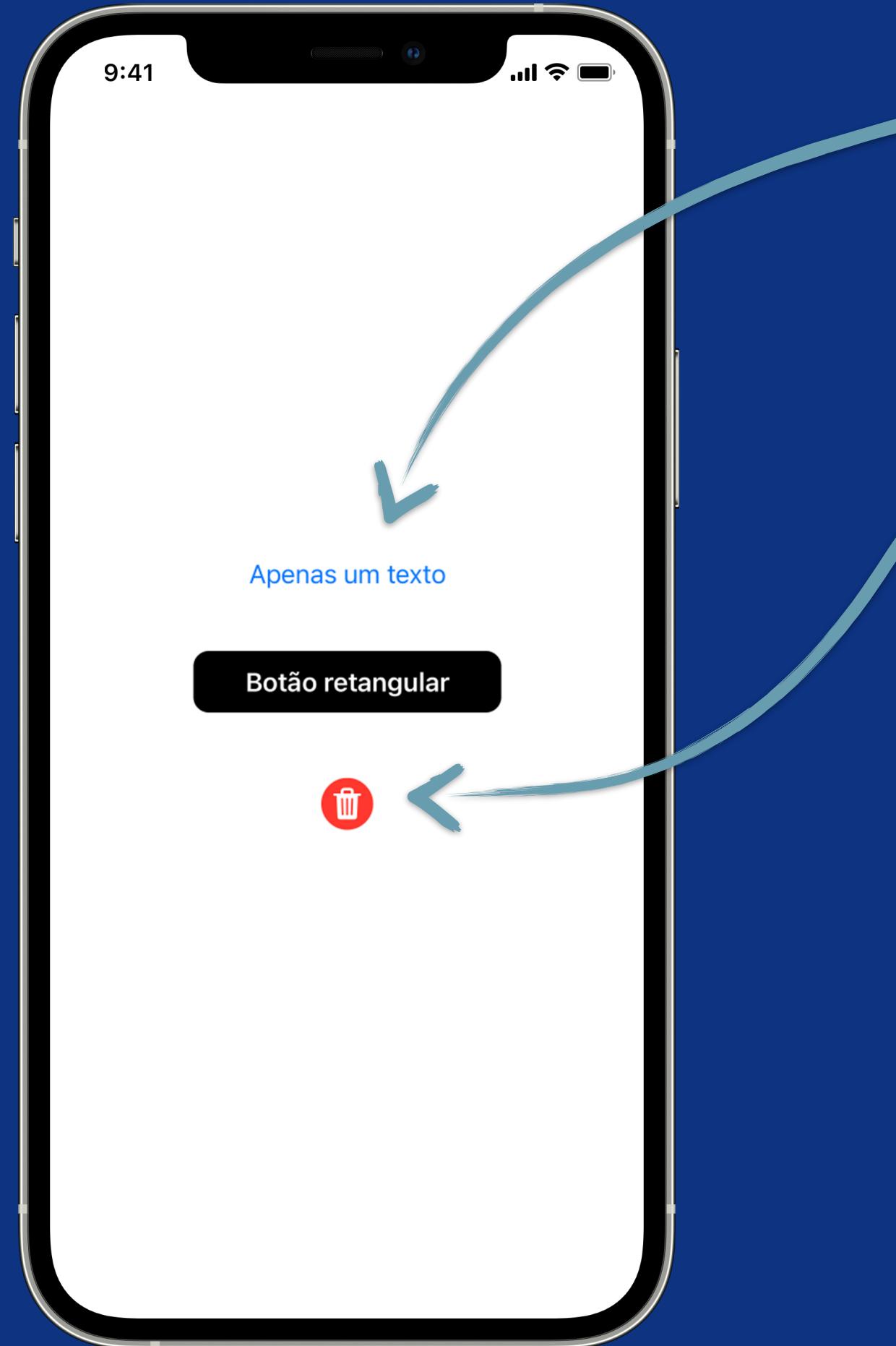
=

Para criarmos um botão, precisamos estabelecer seu corpo e a ação que ele irá realizar.



## Ações e aparência

`action` e `label` | Como configurar a funcionalidade e o visual que um botão terá



```
Button("Apenas um texto",  
      action: { print("Olá") })  
    .padding()
```

```
Button(action: { print("Olá") }) {  
    // Label (aparência)  
    Image(systemName: "trash.circle.fill")  
        .font(.largeTitle)  
        .foregroundColor(.red)  
}.padding()
```

## Ações e aparência action e label

Para criarmos um botão, precisamos estabelecer seu corpo e a ação que ele irá realizar. No SwiftUI, comumente realiza-se isso de duas maneiras diferentes: uma na qual o corpo é apenas um texto e outra na qual ela é uma view qualquer.

# Créditos e referências

Imagen de Capa

Foto por [James Harrison](#) no [Unsplash](#)



Imagen do Doguinho

Foto por [David Clarke](#) no [Unsplash](#)

Fontes e Referências

[Documentação do SwiftUI](#)

# Contatos



Lidiane Chen

[lidianechen@gmail.com](mailto:lidianechen@gmail.com)

Matheus Moreira

[matheussmoreira100@gmail.com](mailto:matheussmoreira100@gmail.com)

# Guia SwiftUI ILUSTRADO

Todos podem programar