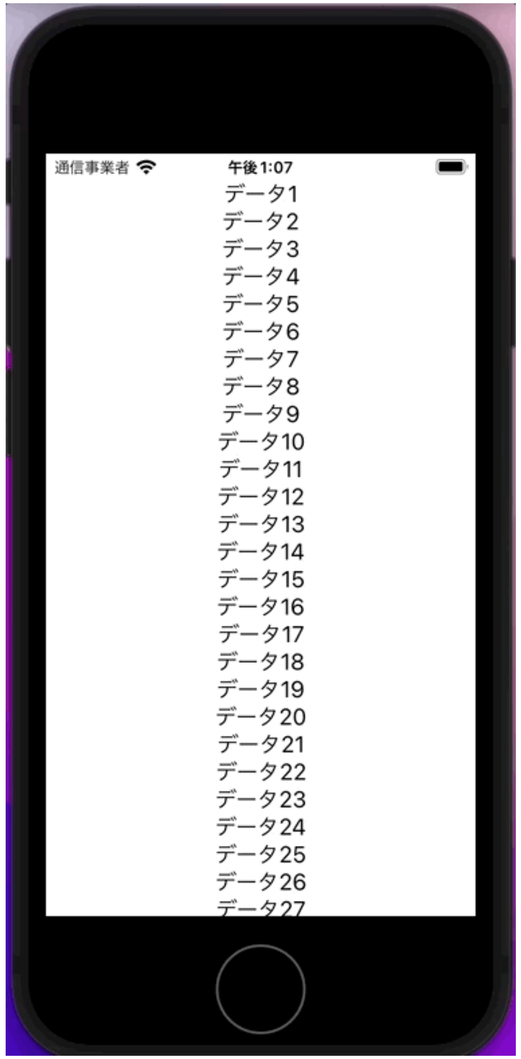
スクロール ビュー

ScrollView

CodeCandy

ScrollView（スクロール　ビュー）

スクロールができるViewです。
SwiftUI はこの機能を持つ2つのビュー、
ScrollViewとList を提供しています。



CodeCandy

リスト

List

1列に並べられたデータの行を表示する

```
10  struct ContentView: View {
11      var body: some View {
12          List {
13              Text("🍎りんご")
14              Text("🍊オレンジ")
15              Text("🍋レモン")
16              Text("🍓いちご")
17              Text("🥝キューイ")
18              Text("🍇ぶどう")
19          } // List ここまで
20      }
21  }
```

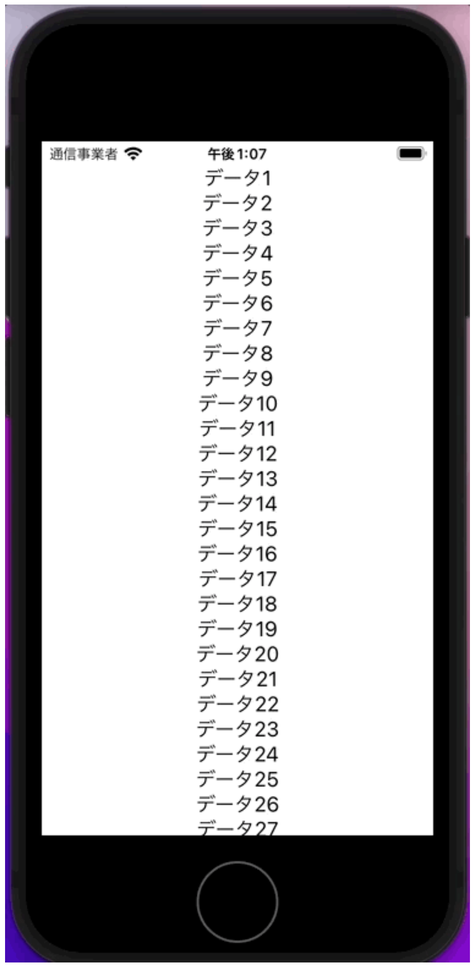
ここから

ここまで

CodeCandy

ScrollViewで100個のデータをスクロール

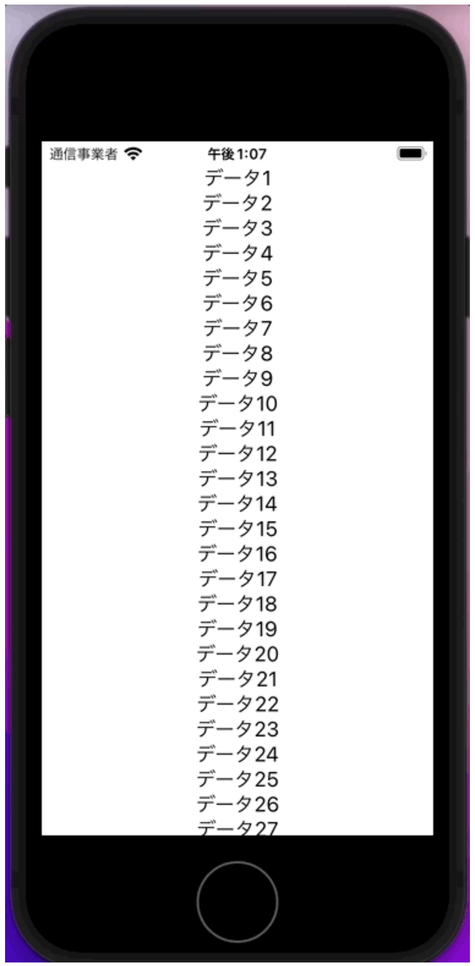
```
24  struct ContentView: View {
25      var body: some View {
26          ScrollView {
27              ForEach(1..<100) { index in
28                  Text("データ\(index)")
29              }
30          }
31      }
32  }
```



CodeCandy

{ }（コードブロック）の役割を復習

```
24  struct ContentView: View {
25      var body: some View {
26          ScrollView {
27              ForEach(1..<100) { index in
28                  Text("データ\(index)")
29              }
30          }
31      }
32  }
```



「}」をコードブロックと言います。
複数のコードを束ねる役割です。
「{」「}」で囲まれた範囲で
プログラムコードが実行されます。

CodeCandy

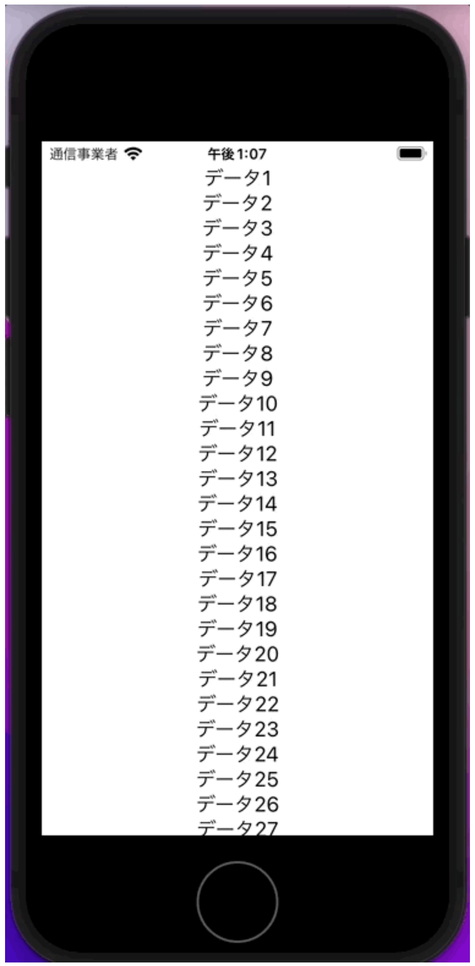
ScrollViewで表示できる範囲

```
24  struct ContentView: View {
25      var body: some View {
        ScrollView {
27          ForEach(1..<100) { index in
28              Text("データ\(index)")
29          }
        }
31      }
32  }
```

ここから

ここまで

ScrollViewのコードブロック外に
記述したViewは、スクロールの
対象にはなりません。



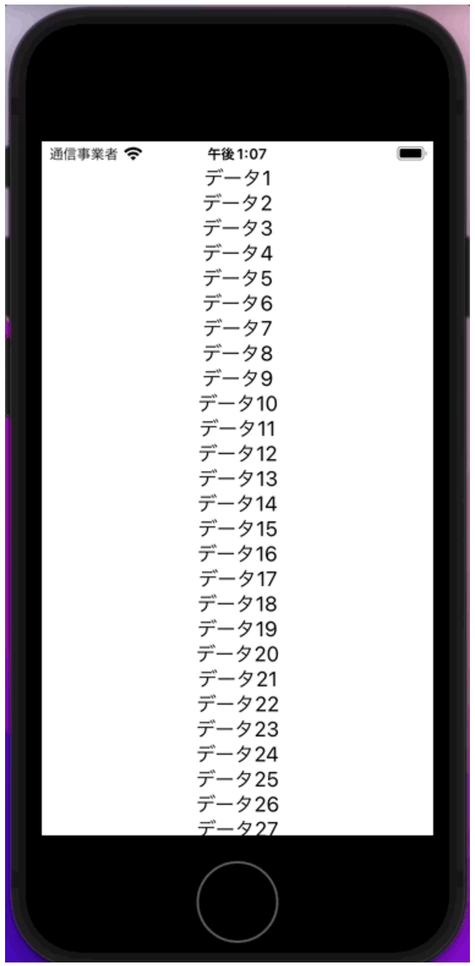
CodeCandy

ループ文 ForEachの復習

```
24  struct ContentView: View {
25      var body: some View {
26          ScrollView {
27              ForEach(1..<100) { index in
28                  Text("データ\(index)")
29              }
30          }
31      }
32  }
```

繰り返す条件

条件の間コードブロックの間の
コードを繰り返し実行ができる



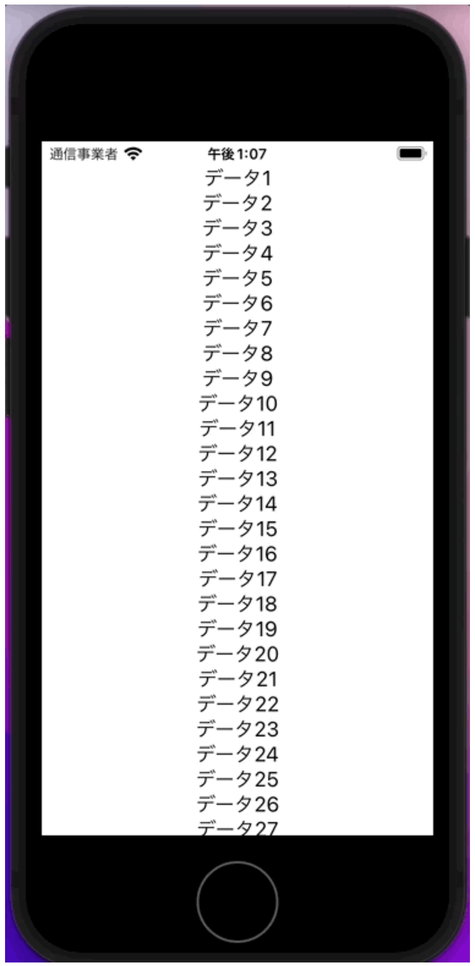
CodeCandy

ループ文 ForEachの復習

```
24  struct ContentView: View {
25      var body: some View {
26          ScrollView {
27              ForEach(1..<100) { index in
28                  Text("データ\(index)")
29              }
30          }
31      }
32  }
```

要素を順番に定数に格納して
コードブロックの範囲で
利用ができる

定数の名前はSwiftの予約語で
なければ任意で指定できる。
ForEachのコードブロックの中で
定数が利用できる。



CodeCandy

「\」バックスラッシュの入力方法

「option」と「¥」を同時実行

CodeCandy

ScrollViewとListの違い

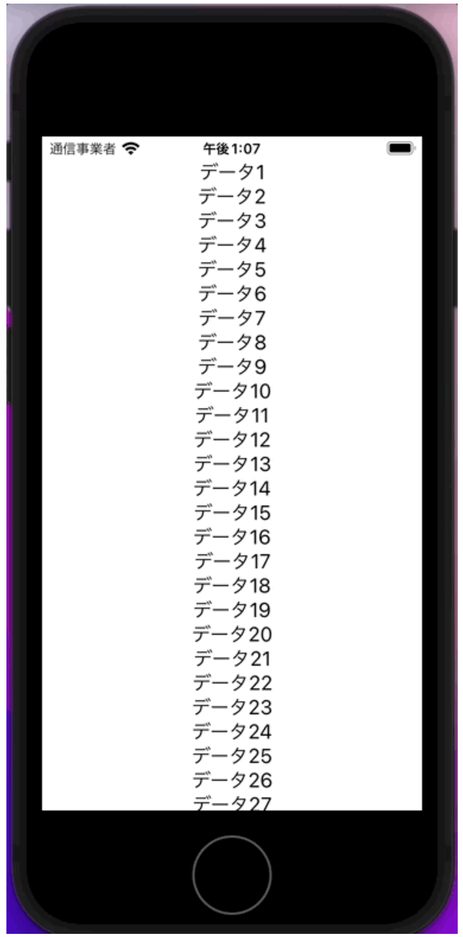
ScrollView

- 垂直・水平のスクロールが可能。
- 垂直・水平の全方向スクロールが可能。
- Listのような区切り線がない。
- 画面表示時に全データをメモリ上に生成。

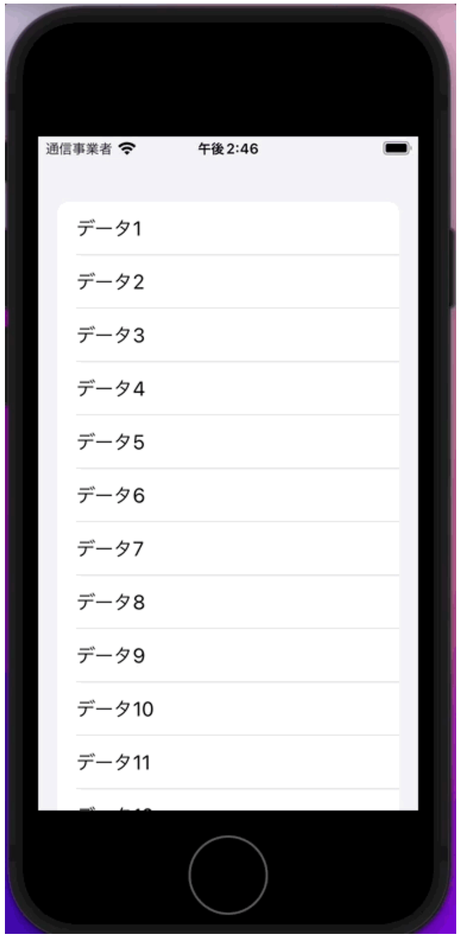
List

- 垂直のみスクロールが可能。
- デフォルト区切り線が表示される（変更可能）
- デフォルトで適度な余白が設けられている。
- スクロールで表示領域に現れるタイミングでメモリ上に生成。

ScrollView



List

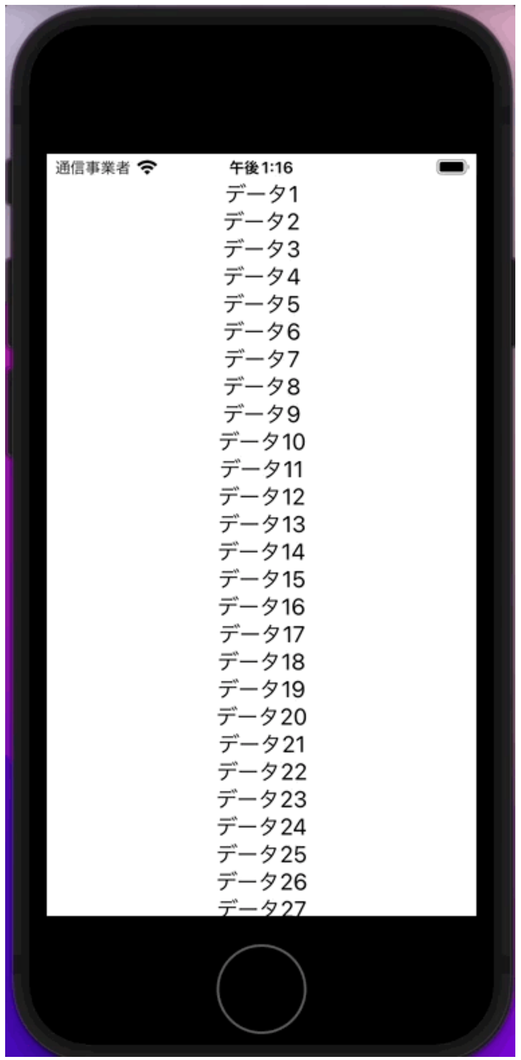


CodeCandy

Indicator（インジケーター）の非表示設定

インジケーターは対象物の動作状態を確認できるもの。
true（表示） or false（非表示） で切り替えが可能。
指定しなければ、ture（表示）が適用される。

```
10  struct ContentView: View {
11      var body: some View {
12          ScrollView(showsIndicators: false) {
13              ForEach(1..<100) { index in
14                  Text("データ\(index)")
15              }
16          }
17      }
18  }
```

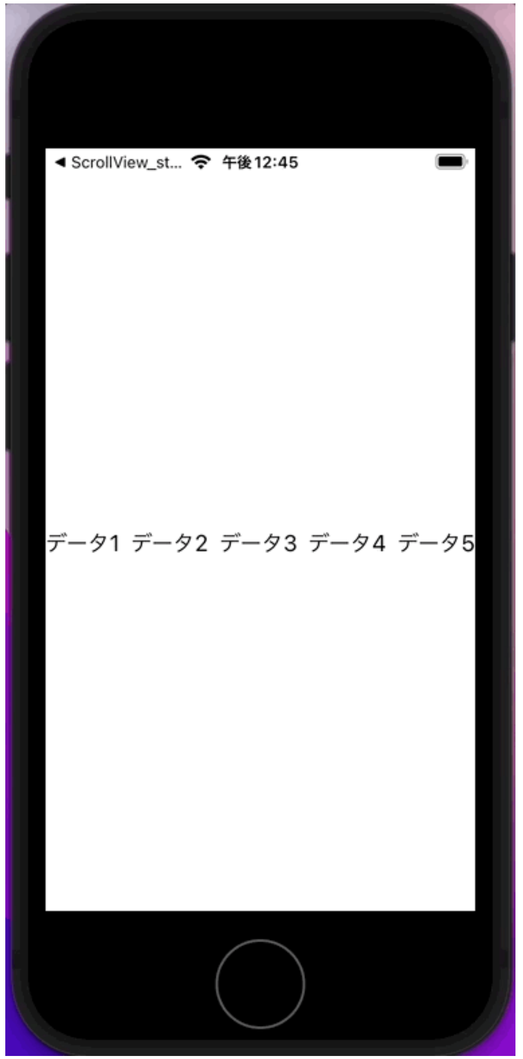


CodeCandy

垂直・水平スクロールの指定方法

.vertical（垂直）.horizontal（水平）で切り替え
指定しなければ .vertical（垂直）が適用される
HStackを用いて水平スクロールを行う

```
10  struct ContentView: View {
11      var body: some View {
12          // 水平スクロール
13          ScrollView(.horizontal) {
14              HStack {
15                  ForEach(1..<100) { index in
16                      Text("データ\(index)")
17                  }
18              }
19          }
20      }
21  }
```



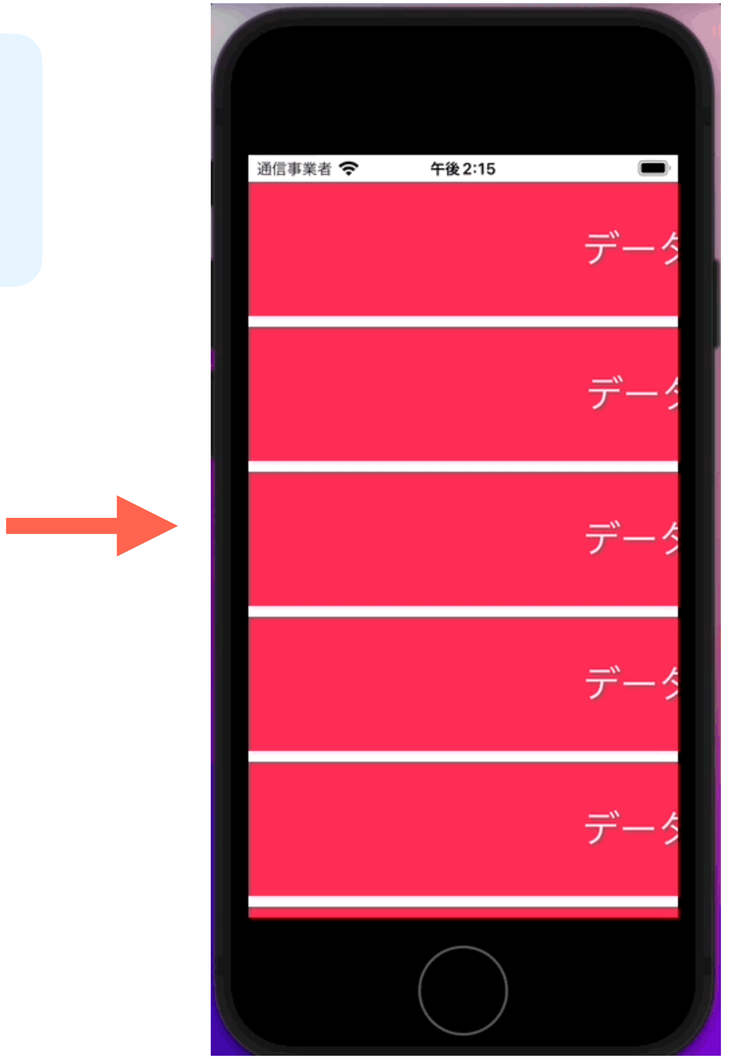
HStackのコードブロックの間の
データを水平スクロールできる

CodeCandy

全方向スクロール

.vertical（垂直）.horizontal（水平）両方を指定して
全方向にスクロールさせることも可能。
ポイントは、[ ]（配列）で指定をすること。

```
10  struct ContentView: View {
11      var body: some View {
12          // 水平・垂直全方向スクロールを指定できる
13           ScrollView([.horizontal, .vertical]) {
14              ForEach(0..<100) { index in
15                  Text("データ\(index)")
16                      .frame(width: 600, height: 100) // 表示サイズ
17                      .foregroundColor(.white)        // 文字色
18                      .background(.pink)              // 背景色
19                      .font(.largeTitle)              // 文字サイズ
20              }
21          }
22      }
23  }
```

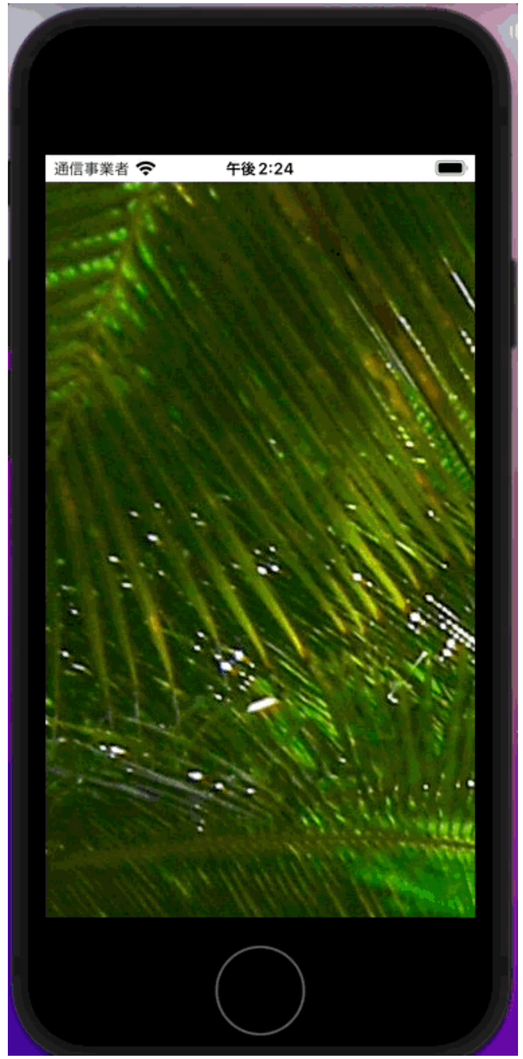


CodeCandy

全方向スクロール

ScrollViewの表示エリア内でスクロールできるを制御できる
なので、Imageでも可能

```
10  struct ContentView: View {
11      var body: some View {
12          // 水平・垂直全方向スクロールを指定できる
13           ScrollView([.horizontal, .vertical]) {
14              Image("beach")
15          }
16      }
17  }
```



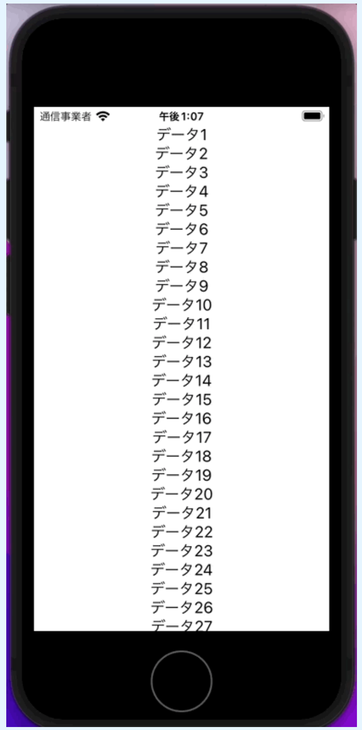
CodeCandy

メモリ生成タイミングの違いを実験しよう

メモリとはコンピューター（スマホ）のデータを記憶する領域

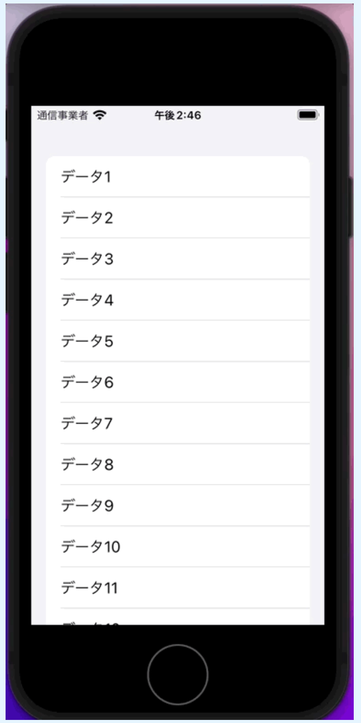
ScrollView

画面表示時に全てのViewをメモリ上に生成。
この例だと一気に100個分のViewが
メモリ上に生成される。



List

画面表示時には、表示されているViewのみが
メモリ上に生成。
スクロールで表示領域に現れるタイミングで
順次Viewがメモリ上に生成される。

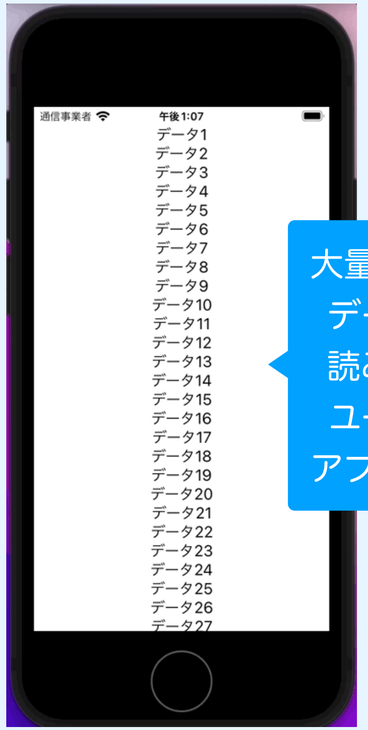


CodeCandy

メモリ生成タイミングの違いを実験しよう

ScrollView

画面表示時に全てのViewをメモリ上に生成。
この例だと一気に100個分のViewが
メモリ上に生成される。



大量のデータや画像等の
データを表示させると
読み込みが遅くなる。
ユーザーを待たせたり
アプリが落ちる可能性。

List

画面表示時には、表示されているViewのみが
メモリ上に生成。
スクロールで表示領域に現れるタイミングで
順次Viewがメモリ上に生成される。

表示されている箇所だけ
データをメモリ上に
展開して表示させるので
大量のデータでも
読み込みが早くなる。
遅延読み込みが行われる。

CodeCandy

メモリ生成タイミングの違いを実験しよう

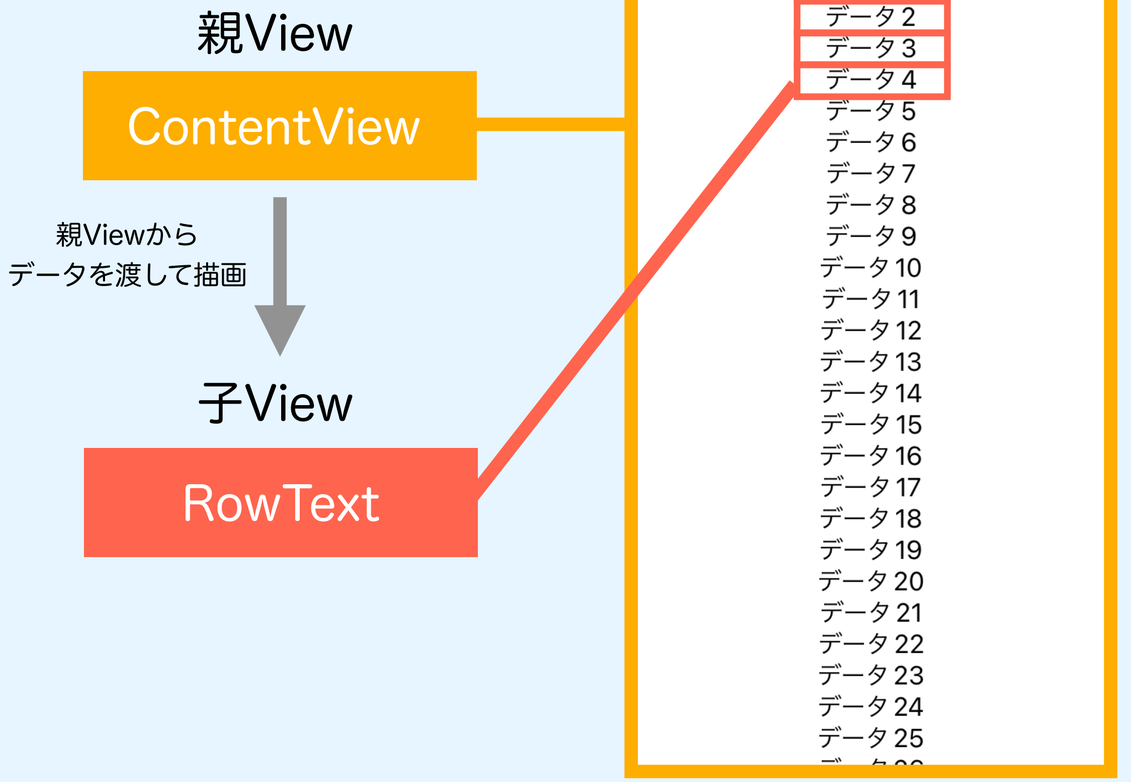
```swift
// 独自の子Viewを定義
struct RowText: View {
    // 受け取った引数を保持する定数
    let outputText: String

    // イニシャライザー（最初に動くメソッド）
    init(_ inputText: String) {
        print(inputText) // ログ出力
        self.outputText = inputText
    }

    var body: some View {
        Text(outputText)
    }
}

struct ContentView: View {
    var body: some View {
        ScrollView() {
            ForEach(0..<100) { index in
                // 子Viewを用いて描画
                RowText("データ\(index)")
            }
        }
    }
}
```

親Viewから子Viewにデータを渡して描画。
イニシャライザーでデータを出力して
どのタイミングでインスタンス化されているか確認している実験コード。

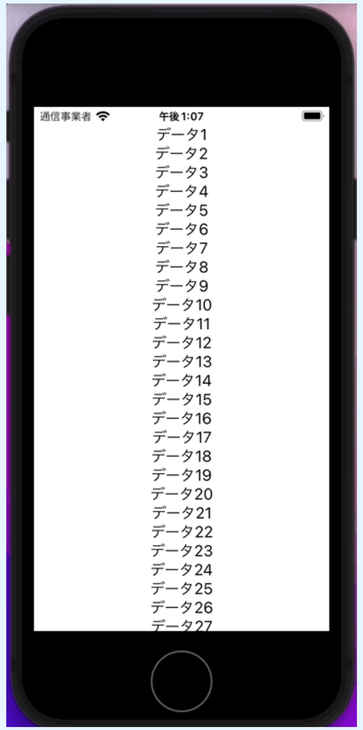


CodeCandy

ScrollViewとListの使い分け

ScrollView

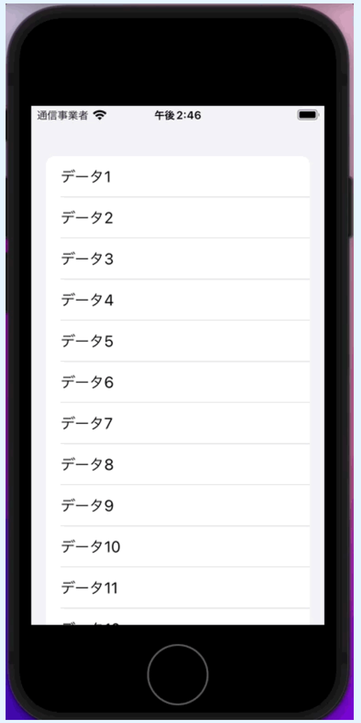
画面表示時に全てのViewをメモリ上に生成。
この例だと一気に100個分のViewが
インスタンス化（メモリ上に生成）される。



少量の固定データを
表示する場合に利用

List

画面表示時には、表示されているViewのみが
メモリ上に生成。
スクロールで表示領域に現れるタイミングで
順次Viewがインスタンス化（メモリ上に生成）される。



データ量が可変で
大量のデータを表示
する場合に利用

CodeCandy

ScrollViewでも遅延読み込みが可能

iOS14.0以上で、LazyVStackとLazyHStackを
用いて遅延読み込み可能になりました。

LazyVStack（垂直）

```
26  struct ContentView: View {
27      var body: some View {
28          ScrollView() {
29              LazyVStack() {
30                  ForEach(0..<100) { index in
31                      Text("データ\(index)")
32                  }
33              }
34          }
35      }
36  }
```

LazyHStack（水平）

```
26  struct ContentView: View {
27      var body: some View {
28          ScrollView() {
29              LazyHStack() {
30                  ForEach(0..<100) { index in
31                      Text("データ\(index)")
32                  }
33              }
34          }
35      }
36  }
```

CodeCandy

ScrollViewとListの使い分け

サポートするiOSのバージョンと、
データの見せ方、作りたいデザインによって選択

		垂直スクロール	水平スクロール	全方向スクロール	遅延読み込み
iOS13以降	List	○	×	○	○
iOS13以降	ScrollView	○	○	○	×
iOS14以降	ScrollView LazyVStack LazyHStack	○	○	○	○

CodeCandy