5-9入力された個数から料金を計算して表示する

```
//
//  ContentView.swift
//  TextFieldkeyboard
//
//  Created by 市川マサル on 2021/05/24.
//
import SwiftUI
struct ContentView: View {
    @State var kosu:String = ""
    let tanka:Double = 250
    let tax:Double = 1.1
    var body: some View {
        VStack (alignment: .leading) {
            //入力テキストフィールド
            HStack {
                Text("個数:").padding(.horizontal,0)
                TextField("0",text: $kosu)
                    .textFieldStyle(RoundedBorderTextFieldStyle())
                    //キーボードの種類を取得する。
                    .keyboardType(.numberPad)
                    .frame(width: 100)
            }
            .font(.title)
            .frame(width: 200)
            //計算結果の表示
            Group {
                if kosuCheck(min: 1, max: 10) {
                    Text("\(price())円です。")
                        .font(.title)
                }else {
                    Text("個数は1~10個入れてください")
                        .foregroundColor(.red)
                        .font(.headline)
                }
            }.frame(width: 300, height: 30)
```

```
        }
    }
    //個数チェックの関数
    func kosuCheck(min: Int , max:Int) -> Bool {
        guard let num = Int(kosu) else {
            return false
        }
        //範囲に入っていればtrue
        return (num>=min && num<=max)
    }
    //料金の計算
    func price() -> Int {
        //kosuを数値に変換できる時アンラップする。
        if let num = Double(kosu) {
            let result = Int(tanka * num * tax)
            return result
        }else {
            return -1
        }
    }
}
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()

    }
}
```

数字だけが入力できるキーボードを表示する

キーボードの種類は、keyboardType(.numberPad)のように指定する。numberPadを指定すると0~9の数字だけのキーボードが表示される。

```
 TextField("0",text: $kosu)
      .textFieldStyle(RoundedBorderTextFieldStyle())
```

```
    //キーボードの種類を取得する。
    .keyboardType(.numberPad)
```

テキストフィールドの計算結果を表示する

入力できる個数を1~10にs制限している。まず、kosuCheck(min:1 , max:10)で個数チェックを行い、個数チェックに合格したならprice()で料金を計算して結果を表示する。もし、テキストフィールドが空だったり、1~10の個数(整数)でなかったりしたならば、「個数は1~10個を入れてください」と赤文字が表示される。
kosuCheck()で分岐した結果がtrueでもfalseでもTextで表示するので、全体をGroup{}で囲い結果のTextに共通したframeサイズを指定している。Groupは複数のビューをひとまとめにして同じ設定をしたいときなどで便利に使う事ができる。

```
Group {
            if kosuCheck(min: 1, max: 10) {
                Text("\(price())円です。")
                    .font(.title)
            }else {
                Text("個数は1~10個入れてください")
                    .foregroundColor(.red)
                    .font(.headline)
            }
        }.frame(width: 300, height: 30)
    }
```

個数をチェックする

個数を1~10に制限している。個数はユーザーが定義したkosuCheck(min: max:)でチェックしている。個数のチェックでは,
kosuに入っている値がString型なので先に数値に変換(キャスト)する必要がある。
Int(kosu)でStringをIntつまり整数に変換できるが、kosuを必ず整数に変換できるとは限らない。
そこでguard let-elseと言う構文を使い、Int(kosu)を整数に変換できたならば変換後の値をnumに代入し,変換できなかった場合はfalseに戻して処理を終了する。

```
//個数チェックの関数
   func kosuCheck(min: Int , max:Int) -> Bool {
      //Int(kosu)でkosuを整数に変換できたならばnumに代入し、できなければfalseを返し
```

```
        guard let num = Int(kosu) else {
            return false
        }
        //範囲に入っていればtrue
        return (num>=min && num<=max)
    }
```

numに入った個数は、(min...max).contains(num)の式でnumがmin...maxの範囲に入ってるかどうかをチェックする。

料金を計算する。

kosuが1~10の整数だった時、price()で料金を計算する。price()ではDouble(kosu)をDouble型に変換してnumに代入してその値を使って単価と税金を掛け合わせて料金を計算している。kosuをDouble型にしている理由は、Double型のtaxと掛け合わせるため。
結果は再びInt()で整数に戻す。
尚、kosuCheck()でkosuが1~10の整数に変換できるチェックしているのでDouble(kosu)がnilになることはないが、安全のためif let-elseを使ってオプションバインディングと呼ばれるコードを入れてDouble(kosu)がnilであった場合のエラーを回避している。

```
//料金の計算
    func price() -> Int {
        //kosuを数値に変換できる時アンラップする。
        if let num = Double(kosu) {
          //Double(kosu)を浮動小数点に変換できたならばnumに代入して料金を計算して値を
            let result = Int(tanka * num * tax)
            return result
        }else {
            return -1
        }
    }
```

タップでキーボードを下げるコードを追加する。

テキストフィールドが入力状態になった時には表示されるキーボードはreturnキー(改行キー)を入力することができる。しかし、kesyboardTypeがnumberPadで表示されるキーボードにはretrunキーがない事から、実機では表示されたキーボードを下げる事ができない。そこで、表示をタップするとキーボードが下がるようにコードを追加する。

```swift
//
//  ContentView.swift
//  TextFieldkeyboard
//
//  Created by 市川マサル on 2021/05/24.
//
import SwiftUI
extension UIApplication {
    //キーボードを下げる
    func endEditing(){
        sendAction(
            #selector(UIResponder.resignFirstResponder),
            to: nil, from: nil, for: nil
        )
    }
}
struct ContentView: View {
    @State var kosu:String = ""
    let tanka:Double = 250
    let tax:Double = 1.1
    var body: some View {
        ZStack {
            //背景のタップでキーボードを下げる
            Color.white//背景を作る
                .onTapGesture {
                   UIApplication.shared.endEditing()
                }
        }
        VStack (alignment: .leading) {
            //入力テキストフィールド
            HStack {
                Text("個数:").padding(.horizontal,0)
                TextField("0",text: $kosu)
                    .textFieldStyle(RoundedBorderTextFieldStyle())
                    //キーボードの種類を取得する。
                    .keyboardType(.numberPad)
                    .frame(width: 100)
```

```
            }
            .font(.title)
            .frame(width: 200)
            //計算結果の表示
            Group {
                if kosuCheck(min: 1, max: 10) {
                    Text("\(price())円です。")
                        .font(.title)
                }else {
                    Text("個数は1~10個入れてください")
                        .foregroundColor(.red)
                        .font(.headline)
                }
            }.frame(width: 300, height: 30)
        }
    }
    //個数チェックの関数
    func kosuCheck(min: Int , max:Int) -> Bool {
        guard let num = Int(kosu) else {
            return false
        }
        //範囲に入っていればtrue
        return (num>=min && num<=max)
    }
    //料金の計算
    func price() -> Int {
        //kosuを数値に変換できる時アンラップする。
        if let num = Double(kosu) {
            let result = Int(tanka * num * tax)
            return result
        }else {
            return -1
        }
    }
}
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
```

```
        ContentView()
    }
}
```

タップ受けて実行する

タップで何かの処理を実行したい時、onTapGestureに実行したコードを書く方法がある。ここではZStackeで背景に白いColor.Whiteを敷き、これにonTapGestureにキーボードを下げるコードを書いている。

```
Color.white//背景を作る
                .onTapGesture {
                    UIApplication.shared.endEditing()
                }
```

onTapGestureで実行するendEdting()はextensionを使ってUIApplicationクラスを拡張したメソッド。

```
func endEditing(){
        sendAction(
            //編集の終了を告げている。
            #selector(UIResponder.resignFirstResponder),
            to: nil, from: nil, for: nil
        )
    }
```