Taschenrechner App - Version 1

User
Generiere bitte eine einfache Taschenrechner App für iOS und watchOS, sowie macOS in SwiftUI

ChatGPT
All the files uploaded by the user have been fully loaded. Searching won't provide additional information.

ChatGPT

~~~
```swift
import SwiftUI

@main
struct CalculatorApp: App {
    var body: some Scene {
        WindowGroup {
            CalculatorView()
        }
    }
}

struct CalculatorView: View {
    @State private var display = "0"
    @State private var firstNumber: Double? = nil
    @State private var operation: String? = nil
    @State private var isNewEntry = true

    let buttons: [[String]] = [
        ["7", "8", "9", "/"],
        ["4", "5", "6", "*"],
        ["1", "2", "3", "-"],
        ["0", "C", "=", "+"]
    ]

    var body: some View {
        VStack(spacing: 12) {
            Text(display)
                .font(.largeTitle)
                .frame(maxWidth: .infinity, alignment: .trailing)
                .padding()

            ForEach(buttons, id: \.self) { row in
                HStack(spacing: 12) {
                    ForEach(row, id: \.self) { item in
                        Button(item) {
                            handleInput(item)
                        }
                        .frame(maxWidth: .infinity, maxHeight: .infinity)
                        .font(.title2)
                        .buttonStyle(.borderedProminent)
                    }
                }
            }
        }
        .padding()
    }

    func handleInput(_ input: String) {
        if let number = Double(input) {
            if isNewEntry {
                display = input
                isNewEntry = false
            } else {
                display += input
            }
        } else if input == "C" {
            display = "0"
            firstNumber = nil
            operation = nil
            isNewEntry = true
        } else if input == "=" {
            if let first = firstNumber,
               let op = operation,
               let second = Double(display) {
                let result: Double
                switch op {
                case "+": result = first + second
                case "-": result = first - second
                case "*": result = first * second
                case "/": result = second != 0 ? first / second : 0
                default: result = 0
                }
                display = String(result)
                firstNumber = nil
                operation = nil
                isNewEntry = true
            }
        } else {
            firstNumber = Double(display)
            operation = input
            isNewEntry = true
        }
    }
}
```
~~~

User
Bitte ändere das Design, sodass es moderner aussieht und verändere die Art und Weise, wie die Buttons funktionieren, damit die Funktionalität der App jederzeit einfach erweitert werden kann.

ChatGPT

```swift
import SwiftUI

enum CalcButton: String, CaseIterable {
    case zero = "0", one = "1", two = "2", three = "3", four = "4"
    case five = "5", six = "6", seven = "7", eight = "8", nine = "9"
    case add = "+", subtract = "−", multiply = "×", divide = "÷"
    case equal = "=", clear = "C"

    var type: ButtonType {
        switch self {
        case .add, .subtract, .multiply, .divide:
            return .operation
        case .equal:
            return .equal
        case .clear:
            return .clear
        default:
            return .number
        }
    }
}

enum ButtonType {
    case number, operation, equal, clear
}

struct CalculatorView: View {
    @State private var display = "0"
    @State private var currentValue: Double = 0
    @State private var pendingOperation: CalcButton? = nil
    @State private var isNewInput = true

    let layout: [[CalcButton]] = [
        [.seven, .eight, .nine, .divide],
        [.four, .five, .six, .multiply],
        [.one, .two, .three, .subtract],
        [.clear, .zero, .equal, .add]
    ]

    var body: some View {
        VStack(spacing: 16) {
            Text(display)
                .font(.system(size: 48, weight: .medium))
                .frame(maxWidth: .infinity, alignment: .trailing)
                .padding()
                .background(.ultraThinMaterial)
                .cornerRadius(16)

            ForEach(layout, id: \.self) { row in
                HStack(spacing: 16) {
                    ForEach(row, id: \.self) { button in
                        Button {
                            handle(button)
                        } label: {
                            Text(button.rawValue)
                                .font(.title2)
                                .frame(maxWidth: .infinity, maxHeight: .infinity)
                        }
                        .buttonStyle(CalcButtonStyle(type: button.type))
                    }
                }
            }
        }
        .padding()
    }

    func handle(_ button: CalcButton) {
        switch button.type {
        case .number:
            if isNewInput {
                display = button.rawValue
                isNewInput = false
            } else {
                display += button.rawValue
            }

        case .operation:
            currentValue = Double(display) ?? 0
            pendingOperation = button
            isNewInput = true

        case .equal:
            guard let op = pendingOperation else { return }
            let secondValue = Double(display) ?? 0
            let result = calculate(currentValue, secondValue, op)
            display = String(result)
            pendingOperation = nil
            isNewInput = true

        case .clear:
            display = "0"
            currentValue = 0
            pendingOperation = nil
            isNewInput = true
        }
    }

    func calculate(_ a: Double, _ b: Double, _ op: CalcButton) -> Double {
        switch op {
        case .add: return a + b
        case .subtract: return a - b
        case .multiply: return a * b
        case .divide: return b != 0 ? a / b : 0
        default: return 0
        }
    }
}

struct CalcButtonStyle: ButtonStyle {
    let type: ButtonType

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.white)
            .background(background)
            .cornerRadius(14)
            .scaleEffect(configuration.isPressed ? 0.95 : 1)
    }

    var background: Color {
        switch type {
        case .number: return .gray.opacity(0.7)
        case .operation: return .orange
        case .equal: return .blue
        case .clear: return .red
        }
    }
}

@main
struct CalculatorApp: App {
    var body: some Scene {
        WindowGroup {
            CalculatorView()
        }
    }
}
```

User

Bitte mach, dass man die Zeichen wie "+", "−" usw. in der Leiste oben sieht, und füge einen Rechnungsverlauf hinzu.

Desweiteren, füge bitte mathematische Funktionen wie "sin", "cos" und "tan" hinzu und mathematische Zeichen wie π

ChatGPT

```swift
import SwiftUI

enum CalcButton: Hashable {
    case number(String)
    case operation(Operation)
    case function(Function)
    case constant(Constant)
    case equal
    case clear
}

enum Operation: String {
    case add = "+", subtract = "−", multiply = "×", divide = "÷"
}

enum Function: String {
    case sin, cos, tan
}

enum Constant: String {
    case pi = "π"
}

@main
struct CalculatorApp: App {
    var body: some Scene {
        WindowGroup {
            CalculatorView()
        }
    }
}

struct CalculatorView: View {
    @State private var display = "0"
    @State private var expression = ""
    @State private var storedValue: Double?
    @State private var pendingOperation: Operation?
    @State private var isNewInput = true
    @State private var history: [String] = []

    let layout: [[CalcButton]] = [
        [.function(.sin), .function(.cos), .function(.tan), .constant(.pi)],
        [.number("7"), .number("8"), .number("9"), .operation(.divide)],
        [.number("4"), .number("5"), .number("6"), .operation(.multiply)],
        [.number("1"), .number("2"), .number("3"), .operation(.subtract)],
        [.clear, .number("0"), .equal, .operation(.add)]
    ]

    var body: some View {
        VStack(spacing: 12) {
            VStack(alignment: .trailing, spacing: 4) {
                Text(expression)
                    .font(.caption)
                    .foregroundColor(.secondary)
                Text(display)
                    .font(.system(size: 44, weight: .medium))
            }
            .frame(maxWidth: .infinity, alignment: .trailing)
            .padding()
            .background(.ultraThinMaterial)
            .cornerRadius(16)

            ScrollView {
                VStack(alignment: .leading, spacing: 4) {
                    ForEach(history.reversed(), id: \.self) { item in
                        Text(item)
                            .font(.caption)
                            .foregroundColor(.secondary)
                    }
                }
            }
            .frame(height: 80)

            ForEach(layout, id: \.self) { row in
                HStack(spacing: 12) {
                    ForEach(row, id: \.self) { button in
                        Button {
                            handle(button)
                        } label: {
                            Text(title(for: button))
                                .frame(maxWidth: .infinity, maxHeight: .infinity)
                                .font(.title3)
                        }
                        .buttonStyle(CalcButtonStyle(button: button))
                    }
                }
            }
        }
        .padding()
    }

    func handle(_ button: CalcButton) {
        switch button {
        case .number(let value):
            if isNewInput {
                display = value
                isNewInput = false
            } else {
                display += value
            }
            expression += value

        case .operation(let op):
            storedValue = Double(display)
            pendingOperation = op
            expression += " \(op.rawValue) "
            isNewInput = true

        case .function(let fn):
            let value = Double(display) ?? 0
            let result: Double
            switch fn {
            case .sin: result = sin(value)
            case .cos: result = cos(value)
            case .tan: result = tan(value)
            }
            history.append("\(fn.rawValue)(\(display)) = \(result)")
            display = format(result)
            expression = "\(fn.rawValue)(\(expression))"
            isNewInput = true

        case .constant(let constant):
            let value: Double = constant == .pi ? .pi : 0
            display = format(value)
            expression += constant.rawValue
            isNewInput = true

        case .equal:
            guard let a = storedValue,
                  let op = pendingOperation,
                  let b = Double(display) else { return }
            let result = calculate(a, b, op)
            history.append("\(expression) = \(result)")
            display = format(result)
            expression = format(result)
            storedValue = nil
            pendingOperation = nil
            isNewInput = true

        case .clear:
            display = "0"
            expression = ""
            storedValue = nil
            pendingOperation = nil
            isNewInput = true
        }
    }

    func calculate(_ a: Double, _ b: Double, _ op: Operation) -> Double {
        switch op {
        case .add: return a + b
        case .subtract: return a - b
        case .multiply: return a * b
        case .divide: return b == 0 ? 0 : a / b
        }
    }

    func format(_ value: Double) -> String {
        value.truncatingRemainder(dividingBy: 1) == 0
        ? String(Int(value))
        : String(value)
    }
    func title(for button: CalcButton) -> String {
        switch button {
        case .number(let v): return v
        case .operation(let o): return o.rawValue
        case .function(let f): return f.rawValue
        case .constant(let c): return c.rawValue
        case .equal: return "="
        case .clear: return "C"
        }
    }
}

struct CalcButtonStyle: ButtonStyle {
    let button: CalcButton

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.white)
            .background(color)
            .cornerRadius(14)
            .scaleEffect(configuration.isPressed ? 0.95 : 1)
    }

    var color: Color {
        switch button {
        case .number: return .gray.opacity(0.7)
        case .operation: return .orange
        case .function, .constant: return .purple
        case .equal: return .blue
        case .clear: return .red
        }
    }
}
```

User

Der Verlauf funktioniert gut, aber: Stelle sicher dass Rechnungen erst in den Verlauf gelegt werden, wenn das = Zeichen gedrückt wurde, des Weiteren bitte verändere die Art und Weise wie man "sin", "cos", "tan" etc. eingeben kann, sodass man z.B. sin drücken kann und danach die Zahl, dafür füge bitte "(" und ")" hinzu.

ChatGPT

```swift
import SwiftUI

enum CalcButton: Hashable {
    case number(String)
    case operation(Operation)
    case function(Function)
    case constant(Constant)
    case leftParen
    case rightParen
    case equal
    case clear
}

enum Operation: String {
    case add = "+", subtract = "−", multiply = "×", divide = "÷"
}

enum Function: String {
    case sin, cos, tan
}

enum Constant: String {
    case pi = "π"
}

@main
struct CalculatorApp: App {
    var body: some Scene {
        WindowGroup {
            CalculatorView()
        }
    }
}

struct CalculatorView: View {
    @State private var display = "0"
    @State private var expression = ""
    @State private var history: [String] = []
    @State private var isNewInput = true

    let layout: [[CalcButton]] = [
        [.function(.sin), .function(.cos), .function(.tan), .constant(.pi)],
        [.leftParen, .rightParen, .clear, .operation(.divide)],
        [.number("7"), .number("8"), .number("9"), .operation(.multiply)],
        [.number("4"), .number("5"), .number("6"), .operation(.subtract)],
        [.number("1"), .number("2"), .number("3"), .operation(.add)],
        [.number("0"), .equal]
    ]

    var body: some View {
        VStack(spacing: 12) {
            VStack(alignment: .trailing, spacing: 4) {
                Text(expression)
                    .font(.caption)
                    .foregroundColor(.secondary)
                Text(display)
                    .font(.system(size: 44, weight: .medium))
            }
            .frame(maxWidth: .infinity, alignment: .trailing)
            .padding()
            .background(.ultraThinMaterial)
            .cornerRadius(16)

            ScrollView {
                VStack(alignment: .leading, spacing: 4) {
                    ForEach(history.reversed(), id: \.self) { item in
                        Text(item)
                            .font(.caption)
                            .foregroundColor(.secondary)
                    }
                }
            }
            .frame(height: 80)

            ForEach(layout, id: \.self) { row in
                HStack(spacing: 12) {
                    ForEach(row, id: \.self) { button in
                        Button {
                            handle(button)
                        } label: {
                            Text(title(for: button))
                                .frame(maxWidth: .infinity, maxHeight: .infinity)
                                .font(.title3)
                        }
                        .buttonStyle(CalcButtonStyle(button: button))
                    }
                }
            }
        }
        .padding()
    }

    func handle(_ button: CalcButton) {
        switch button {
        case .number(let value):
            if isNewInput {
                display = value
                isNewInput = false
            } else {
                display += value
            }
            expression += value

        case .operation(let op):
            expression += " \(op.rawValue) "
            isNewInput = true

        case .function(let fn):
            expression += "\(fn.rawValue)("

        case .leftParen:
            expression += "("

        case .rightParen:
            expression += ")"

        case .constant(let constant):
            if constant == .pi {
                expression += "π"
                display = format(.pi)
            }

        case .equal:
            let sanitized = expression
                .replacingOccurrences(of: "×", with: "*")
                .replacingOccurrences(of: "÷", with: "/")
                .replacingOccurrences(of: "π", with: "\(Double.pi)")
                .replacingOccurrences(of: "sin", with: "sin")
                .replacingOccurrences(of: "cos", with: "cos")
                .replacingOccurrences(of: "tan", with: "tan")

            if let result = evaluate(sanitized) {
                history.append("\(expression) = \(format(result))")
                display = format(result)
                expression = format(result)
                isNewInput = true
            }

        case .clear:
            display = "0"
            expression = ""
            isNewInput = true
        }
    }

    func evaluate(_ expr: String) -> Double? {
        let expression = NSExpression(format: expr)
        return expression.expressionValue(with: nil, context: nil) as? Double
    }

    func format(_ value: Double) -> String {
        value.truncatingRemainder(dividingBy: 1) == 0
        ? String(Int(value))
        : String(value)
    }

    func title(for button: CalcButton) -> String {
        switch button {
        case .number(let v): return v
        case .operation(let o): return o.rawValue
        case .function(let f): return f.rawValue
        case .constant(let c): return c.rawValue
        case .leftParen: return "("
        case .rightParen: return ")"
        case .equal: return "="
        case .clear: return "C"
        }
    }
}

struct CalcButtonStyle: ButtonStyle {
    let button: CalcButton

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.white)
            .background(color)
            .cornerRadius(14)
            .scaleEffect(configuration.isPressed ? 0.95 : 1)
    }

    var color: Color {
        switch button {
        case .number: return .gray.opacity(0.7)
        case .operation: return .orange
        case .function, .constant, .leftParen, .rightParen: return .purple
        case .equal: return .blue
        case .clear: return .red
        }
    }
}
```

User
Wenn ich sin(03) eingebe bekomme ich den folgenden Fehler:
Thread 1: "Unable to parse function name 'sin:' into supported selector (sin:) ".

Bitte füge "," für Dezimalzahlen hinzu und stelle sicher, dass mathematische Fehler korrekt angezeigt werden und die App nicht zum abstürzen bringen.

ChatGPT

```swift
import SwiftUI

enum CalcButton: Hashable {
    case number(String)
    case operation(Operation)
    case function(Function)
    case constant(Constant)
    case leftParen
    case rightParen
    case equal
    case clear
    case decimal
}

enum Operation: String {
    case add = "+", subtract = "−", multiply = "×", divide = "÷"
}

enum Function: String {
    case sin, cos, tan
}

enum Constant: String {
    case pi = "π"
}

@main
struct CalculatorApp: App {
    var body: some Scene {
        WindowGroup {
            CalculatorView()
        }
    }
}

struct CalculatorView: View {
    @State private var display = "0"
    @State private var expression = ""
    @State private var history: [String] = []
    @State private var errorMessage: String?
    @State private var isNewInput = true

    let layout: [[CalcButton]] = [
        [.function(.sin), .function(.cos), .function(.tan), .constant(.pi)],
        [.leftParen, .rightParen, .decimal, .operation(.divide)],
        [.number("7"), .number("8"), .number("9"), .operation(.multiply)],
        [.number("4"), .number("5"), .number("6"), .operation(.subtract)],
        [.number("1"), .number("2"), .number("3"), .operation(.add)],
        [.clear, .number("0"), .equal]
    ]

    var body: some View {
        VStack(spacing: 12) {
            VStack(alignment: .trailing, spacing: 4) {
                Text(expression)
                    .font(.caption)
                    .foregroundColor(.secondary)
                Text(errorMessage ?? display)
                    .font(.system(size: 44, weight: .medium))
                    .foregroundColor(errorMessage == nil ? .primary : .red)
            }
            .frame(maxWidth: .infinity, alignment: .trailing)
            .padding()
            .background(.ultraThinMaterial)
            .cornerRadius(16)
            ScrollView {
                VStack(alignment: .leading, spacing: 4) {
                    ForEach(history.reversed(), id: \.self) { item in
                        Text(item)
                            .font(.caption)
                            .foregroundColor(.secondary)
                    }
                }
            }
            .frame(height: 80)

            ForEach(layout, id: \.self) { row in
                HStack(spacing: 12) {
                    ForEach(row, id: \.self) { button in
                        Button {
                            handle(button)
                        } label: {
                            Text(title(for: button))
                                .frame(maxWidth: .infinity, maxHeight: .infinity)
                                .font(.title3)
                        }
                        .buttonStyle(CalcButtonStyle(button: button))
                    }
                }
            }
        }
        .padding()
    }

    func handle(_ button: CalcButton) {
        errorMessage = nil

        switch button {
        case .number(let value):
            if isNewInput {
                display = value
                isNewInput = false
            } else {
                display += value
            }
            expression += value

        case .decimal:
            if !display.contains(",") {
                display += ","
                expression += ","
                isNewInput = false
            }

        case .operation(let op):
            expression += " \(op.rawValue) "
            isNewInput = true

        case .function(let fn):
            expression += "\(fn.rawValue)("
            isNewInput = true

        case .leftParen:
            expression += "("

        case .rightParen:
            expression += ")"

        case .constant(let constant):
            if constant == .pi {
                expression += "π"
                display = format(.pi)
                isNewInput = true
            }

        case .equal:
            let prepared = prepareExpression(expression)
            guard let prepared else {
                errorMessage = "Ungültiger Ausdruck"
                return
            }

            if let result = safeEvaluate(prepared) {
                history.append("\(expression) = \(format(result))")
                display = format(result)
                expression = format(result)
                isNewInput = true
            } else {
                errorMessage = "Mathematischer Fehler"
            }

        case .clear:
            display = "0"
            expression = ""
            history.removeAll()
            errorMessage = nil
            isNewInput = true
        }
    }

    func prepareExpression(_ expr: String) -> String? {
        expr
            .replacingOccurrences(of: "×", with: "*")
            .replacingOccurrences(of: "÷", with: "/")
            .replacingOccurrences(of: ",", with: ".")
            .replacingOccurrences(of: "π", with: "\(Double.pi)")
            .replacingOccurrences(of: "sin(", with: "FUNCTION(sin:, ")
            .replacingOccurrences(of: "cos(", with: "FUNCTION(cos:, ")
            .replacingOccurrences(of: "tan(", with: "FUNCTION(tan:, ")
```

```swift
    }

    func safeEvaluate(_ expr: String) -> Double? {
        let nsExpr = NSExpression(format: expr)
        let result = nsExpr.expressionValue(with: nil, context: nil)
        return result as? Double
    }

    func format(_ value: Double) -> String {
        let formatted = value.truncatingRemainder(dividingBy: 1) == 0
            ? String(Int(value))
            : String(value)
        return formatted.replacingOccurrences(of: ".", with: ",")
    }

    func title(for button: CalcButton) -> String {
        switch button {
        case .number(let v): return v
        case .operation(let o): return o.rawValue
        case .function(let f): return f.rawValue
        case .constant(let c): return c.rawValue
        case .leftParen: return "("
        case .rightParen: return ")"
        case .decimal: return ","
        case .equal: return "="
        case .clear: return "C"
        }
    }
}

struct CalcButtonStyle: ButtonStyle {
    let button: CalcButton

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.white)
            .background(color)
            .cornerRadius(14)
            .scaleEffect(configuration.isPressed ? 0.95 : 1)
    }

    var color: Color {
        switch button {
        case .number, .decimal: return .gray.opacity(0.7)
        case .operation: return .orange
        case .function, .constant, .leftParen, .rightParen: return .purple
        case .equal: return .blue
        case .clear: return .red
        }
    }
}
```

User

Wenn ich "sin(03)" eingebe bekomme ich immer noch diesen Fehler:

```
*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: 'Unable to parse the format string
"FUNCTION(sin:, 03) == 1"'
*** First throw call stack:
(
        0   CoreFoundation                      0x00000001804f71d0 __exceptionPreprocess + 172
        1   libobjc.A.dylib                     0x000000018009c094 objc_exception_throw + 72
        2   Foundation                          0x00000001810eb6b0 +[NSPredicate predicateWithFormat:arguments:] + 216
        3   Foundation                          0x000000018104ee24 +[NSExpression expressionWithFormat:arguments:] + 64
        4   Foundation                          0x0000000180f21d5c
$sSo12NSExpressionC10FoundationE6format_ABSSh_s7CVarArg_pdtcfC + 104
        5   Taschenrechner.debug.dylib          0x0000000100c1ab5c $s14Taschenrechner14CalculatorViewV12safeEvaluateySdSgSSF +
128
        6   Taschenrechner.debug.dylib          0x0000000100c1862c $s14Taschenrechner14CalculatorViewV6handleyyAA10CalcButtonOF
+ 5096
        7   Taschenrechner.debug.dylib          0x0000000100c17234
$s14Taschenrechner14CalculatorViewV4bodyQrvg7SwiftUI05TupleC0VyAE0C0PAEE12cornerRadius_11antialiasedQr12CoreGraphics7CGFloatV_SbtFQO
8   SwiftUI                             0x00000001da4cf920
$s7SwiftUI17ContextMenuBridgeC07contextD11Interaction_027willPerformPreviewActionForD4With8animatorySo09UIContextdG0C_So0oD13Configu
+ 24
        9   SwiftUI                             0x00000001dabd0db0
$sScM14assumeIsolated_4file4linexxyKScMYcXE_s12StaticStringVSutKs8SendableRzlFZyt_Tg5 + 132
        10  SwiftUI                             0x00000001dab9b7ac $s7SwiftUI12ButtonActionO14callAsFunctionyyF + 388
        11  SwiftUI                             0x00000001d9e8a8f0
$s7SwiftUI27PlatformItemListButtonStyleV8makeBody13configurationQrAA09PrimitivefG13ConfigurationV_tFyycAGYbcfu_yycfu0_TATm + 52
        12  SwiftUI                             0x00000001da46770c
$s7SwiftUI14ButtonBehaviorV5ended33_AEEDD090E917AC57C12008D974DC6805LLyyF + 224
        13  SwiftUI                             0x00000001da46d258
$s7SwiftUI14ButtonBehaviorV4bodyQrvgyyYbcACyxGYbcfu_yyYbcfu0_TA + 32
        14  SwiftUI                             0x00000001dab47818
$s7SwiftUI14_ButtonGestureV12internalBodyQrvgAA04_MapD0VyAA09PrimitivecD0VytGyXEfU_ySo7CGPointVSgcfU0_yyScMYcXEfU_TA + 28
        15  SwiftUI                             0x00000001dabd0db0
$sScM14assumeIsolated_4file4linexxyKScMYcXE_s12StaticStringVSutKs8SendableRzlFZyt_Tg5 + 132
        16  SwiftUI                             0x00000001dab429f4
$s7SwiftUI14_ButtonGestureV12internalBodyQrvgAA04_MapD0VyAA09PrimitivecD0VytGyXEfU_ySo7CGPointVSgcfU0_ + 80
        17  SwiftUI                             0x00000001dab489a8
$s7SwiftUI31PrimitiveButtonGestureCallbacks33_2218E1141B3D7C3A65B6697591AFB638LLV8dispatch5phase5stateyycSgAA0E5PhaseOyAA0cdE4CoreAC
+ 84
        18  SwiftUICore                         0x00000001db026db4
$sypSgSo8_NSRangeVSpy10ObjectiveC8ObjCBoolVGIgnyy_AacGIegnyy_TRTA + 20
        19  SwiftUICore                         0x00000001db19d080 $sIeg_ytIegr_TR + 20
        20  SwiftUI                             0x00000001da0622c4 $sytIegr_Ieg_TR + 20
        21  SwiftUI                             0x00000001da31ede0
$s7SwiftUI17DragAndDropBridgeC15dragInteraction_16sessionWillBeginySo06UIDragH0C_So0L7Session_ptFyycfU0_ + 44
        22  SwiftUICore                         0x00000001db46a708 $s7SwiftUI6UpdateO15dispatchActionsyyFZ + 1092
        23  SwiftUICore                         0x00000001db469760 $s7SwiftUI6UpdateO3endyyFZ + 124
        24  SwiftUICore                         0x00000001db469498
$s7SwiftUI6UpdateO13enqueueAction6reason_s6UInt32VAA16CustomEventTraceV0eI4TypeO6ReasonOSg_yyctFZ + 188
        25  SwiftUI                             0x00000001da7288e0
$s7SwiftUI32UIKitResponderEventBindingBridgeC12flushActionssyyF + 400
        26  SwiftUI                             0x00000001da72893c
$s7SwiftUI32UIKitResponderEventBindingBridgeC12flushActionssyyFTo + 24
        27  UIKitCore                           0x0000000185e04bb0 -[UIGestureRecognizerTarget
_sendActionWithGestureRecognizer:] + 76
        28  UIKitCore                           0x0000000185e0dd84 _UIGestureRecognizerSendTargetActions + 88
        29  UIKitCore                           0x0000000185e0aa68 _UIGestureRecognizerSendActions + 296
        30  UIKitCore                           0x0000000185e0a62c -[UIGestureRecognizer _updateGestureForActiveEvents] + 320
        31  UIKitCore                           0x0000000185e0f2c8 -[UIGestureRecognizer gestureNode:didUpdatePhase:] + 296
        32  Gestures                            0x0000000230fec6a0 GFGestureNodeDefaultValue + 3760
        33  Gestures                            0x000000023100869c _swift_stdlib_malloc_size + 22468
        34  Gestures                            0x00000002310343dc __swift_memcpy24_8 + 19448
        35  Gestures                            0x000000023105b464 GFGestureNodeCoordinatorCreate + 2080
        36  UIKitCore                           0x0000000185dfc090 -[UIGestureEnvironment _updateForEvent:window:] + 468
        37  UIKitCore                           0x0000000186384450 -[UIWindow sendEvent:] + 2796
        38  UIKitCore                           0x00000001863622e4 -[UIApplication sendEvent:] + 376
        39  UIKitCore                           0x0000000186211270 __dispatchPreprocessedEventFromEventQueue + 1184
        40  UIKitCore                           0x0000000186213f24 __processEventQueue + 4800
        41  UIKitCore                           0x000000018620c4d0 updateCycleEntry + 168
        42  UIKitCore                           0x000000018582f378 _UIUpdateSequenceRunNext + 120
        43  UIKitCore                           0x00000001862640a4 schedulerStepScheduledMainSectionContinue + 56
        44  UpdateCycle                         0x00000002501912b4 _ZN2UC10DriverCore18continueProcessingEv + 80
        45  CoreFoundation                      0x00000001804563a4 __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ +
24
        46  CoreFoundation                      0x00000001804562ec __CFRunLoopDoSource0 + 168
        47  CoreFoundation                      0x0000000180455a78 __CFRunLoopDoSources0 + 220
        48  CoreFoundation                      0x0000000180454c4c __CFRunLoopRun + 760
        49  CoreFoundation                      0x000000018044fcec _CFRunLoopRunSpecificWithOptions + 496
        50  GraphicsServices                    0x0000000192a669bc GSEventRunModal + 116
        51  UIKitCore                           0x0000000186348574 -[UIApplication _run] + 772
        52  UIKitCore                           0x000000018634c79c UIApplicationMain + 124
        53  SwiftUI                             0x00000001da58d620
$s7SwiftUI17KitRendererCommon33_ACC2C5639A7D76F611E170E831FCA491LLys5NeverOyXlXpFAESpySpys4Int8VGSgGXEfU_ + 164
        54  SwiftUI                             0x00000001da58d368 $s7SwiftUI6runAppys5NeverOxAA0D0RzlF + 180
        55  SwiftUI                             0x00000001da31b42c $s7SwiftUI3AppPAAE4mainyyFZ + 148
        56  Taschenrechner.debug.dylib          0x0000000100c1359c $s14Taschenrechner13CalculatorAppV5$mainyyFZ + 40
        57  Taschenrechner.debug.dylib          0x0000000100c1c1a0 __debug_main_executable_dylib_entry_point + 12
        58  dyld                                0x0000000100d993d0 start_sim + 20
        59  ???                                 0x0000000100c6cd54 0x0 + 4307995988
)
libc++abi: terminating due to uncaught exception of type NSException
*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: 'Unable to parse the format string
"FUNCTION(sin:, 03) == 1"'
terminating due to uncaught exception of type NSException
CoreSimulator 1051.17.7 - Device: iPhone 17 (3C9DF8DD-313D-47E3-83DF-4B1F0F943DC7) - Runtime: iOS 26.2 (23C54) - DeviceType:
iPhone 17
```

ChatGPT

~~~~
```swift
import SwiftUI
import Foundation

enum CalcButton: Hashable {
    case number(String)
    case operation(Operation)
    case function(Function)
    case constant(Constant)
    case leftParen
    case rightParen
    case decimal
    case equal
    case clear
}

enum Operation: String {
    case add = "+", subtract = "−", multiply = "×", divide = "÷"
}

enum Function: String {
    case sin, cos, tan
}

enum Constant: String {
    case pi = "π"
}

@main
struct CalculatorApp: App {
    var body: some Scene {
        WindowGroup {
            CalculatorView()
        }
    }
}

struct CalculatorView: View {
    @State private var expression = ""
    @State private var display = "0"
    @State private var history: [String] = []
    @State private var errorMessage: String?
    @State private var isNewInput = true

    let layout: [[CalcButton]] = [
        [.function(.sin), .function(.cos), .function(.tan), .constant(.pi)],
        [.leftParen, .rightParen, .decimal, .operation(.divide)],
        [.number("7"), .number("8"), .number("9"), .operation(.multiply)],
        [.number("4"), .number("5"), .number("6"), .operation(.subtract)],
        [.number("1"), .number("2"), .number("3"), .operation(.add)],
        [.clear, .number("0"), .equal]
    ]

    var body: some View {
        VStack(spacing: 12) {
            VStack(alignment: .trailing, spacing: 4) {
~~~~

```
                Text(expression)
                    .font(.caption)
                    .foregroundColor(.secondary)

                Text(errorMessage ?? display)
                    .font(.system(size: 44, weight: .medium))
                    .foregroundColor(errorMessage == nil ? .primary : .red)
            }
            .frame(maxWidth: .infinity, alignment: .trailing)
            .padding()
            .background(.ultraThinMaterial)
            .cornerRadius(16)

            ScrollView {
                VStack(alignment: .leading, spacing: 4) {
                    ForEach(history.reversed(), id: \.self) {
                        Text($0)
                            .font(.caption)
                            .foregroundColor(.secondary)
                    }
                }
            }
            .frame(height: 80)

            ForEach(layout, id: \.self) { row in
                HStack(spacing: 12) {
                    ForEach(row, id: \.self) { button in
                        Button {
                            handle(button)
                        } label: {
                            Text(title(for: button))
                                .frame(maxWidth: .infinity, maxHeight: .infinity)
                                .font(.title3)
                        }
                        .buttonStyle(CalcButtonStyle(button: button))
                    }
                }
            }
        }
        .padding()
    }

    func handle(_ button: CalcButton) {
        errorMessage = nil

        switch button {
        case .number(let value):
            if isNewInput {
                display = value
                isNewInput = false
            } else {
                display += value
            }
            expression += value

        case .decimal:
            if !display.contains(",") {
                display += ","
                expression += ","
            }

        case .operation(let op):
            expression += " \(op.rawValue) "
            isNewInput = true

        case .function(let fn):
            expression += "\(fn.rawValue)("
            isNewInput = true

        case .leftParen:
            expression += "("

        case .rightParen:
            expression += ")"

        case .constant(.pi):
            expression += "π"
            display = format(.pi)
            isNewInput = true

        case .equal:
            let original = expression
            guard let result = evaluateExpression(expression) else {
                errorMessage = "Mathematischer Fehler"
                return
            }
            history.append("\(original) = \(format(result))")
            display = format(result)
            expression = format(result)
            isNewInput = true

        case .clear:
            expression = ""
            display = "0"
            history.removeAll()
            errorMessage = nil
            isNewInput = true
        }
    }

    func evaluateExpression(_ expr: String) -> Double? {
        var sanitized = expr
            .replacingOccurrences(of: "π", with: "\(Double.pi)")
            .replacingOccurrences(of: ",", with: ".")
            .replacingOccurrences(of: "×", with: "*")
            .replacingOccurrences(of: "÷", with: "/")

        sanitized = resolveFunctions(in: sanitized)

        let formatter = NumberFormatter()
        formatter.decimalSeparator = "."

        let exp = NSExpression(format: sanitized)
        return exp.expressionValue(with: nil, context: nil) as? Double
    }

    func resolveFunctions(in expr: String) -> String {
        var result = expr

        for fn in [Function.sin, .cos, .tan] {
            while let range = result.range(of: "\(fn.rawValue)(") {
                guard let end = matchingParen(in: result, from: range.upperBound) else { break }
                let inner = String(result[range.upperBound..<end])
                if let value = Double(inner) {
                    let computed: Double
                    switch fn {
                    case .sin: computed = sin(value)
                    case .cos: computed = cos(value)
                    case .tan: computed = tan(value)
                    }
                    result.replaceSubrange(range.lowerBound...end, with: "\(computed)")
                } else {
                    return ""
                }
            }
        }
        return result
    }

    func matchingParen(in text: String, from index: String.Index) -> String.Index? {
        var depth = 1
        var i = index
        while i < text.endIndex {
            if text[i] == "(" { depth += 1 }
            if text[i] == ")" { depth -= 1 }
            if depth == 0 { return i }
            i = text.index(after: i)
        }
        return nil
    }

    func format(_ value: Double) -> String {
        let str = value.truncatingRemainder(dividingBy: 1) == 0
            ? String(Int(value))
            : String(value)
        return str.replacingOccurrences(of: ".", with: ",")
    }

    func title(for button: CalcButton) -> String {
        switch button {
        case .number(let v): return v
        case .operation(let o): return o.rawValue
        case .function(let f): return f.rawValue
        case .constant(let c): return c.rawValue
        case .leftParen: return "("
        case .rightParen: return ")"
        case .decimal: return ","
        case .equal: return "="
        case .clear: return "C"
        }
    }
}

struct CalcButtonStyle: ButtonStyle {
    let button: CalcButton

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.white)
            .background(color)
            .cornerRadius(14)
            .scaleEffect(configuration.isPressed ? 0.95 : 1)
    }

    var color: Color {
        switch button {
        case .number, .decimal: return .gray.opacity(0.7)
        case .operation: return .orange
        case .function, .constant, .leftParen, .rightParen: return .purple
        case .equal: return .blue
        case .clear: return .red
        }
    }
}
```

User

Das hat es nun behoben, füge als nächstes bitte den Verlauf als Popup-Fenster hinzu, anstatt ihn in der Mitte des Bildschirms zu zeigen.

ChatGPT

```swift
import SwiftUI
import Foundation

enum CalcButton: Hashable {
    case number(String)
    case operation(Operation)
    case function(Function)
    case constant(Constant)
    case leftParen
    case rightParen
    case decimal
    case equal
    case clear
    case history
}

enum Operation: String {
    case add = "+", subtract = "−", multiply = "×", divide = "÷"
}

enum Function: String {
    case sin, cos, tan
}

enum Constant: String {
    case pi = "π"
}

@main
struct CalculatorApp: App {
    var body: some Scene {
        WindowGroup {
            CalculatorView()
        }
    }
}

struct CalculatorView: View {
    @State private var expression = ""
    @State private var display = "0"
    @State private var history: [String] = []
    @State private var errorMessage: String?
    @State private var isNewInput = true
    @State private var showHistory = false

    let layout: [[CalcButton]] = [
        [.history, .function(.sin), .function(.cos), .function(.tan)],
        [.constant(.pi), .leftParen, .rightParen, .decimal],
        [.number("7"), .number("8"), .number("9"), .operation(.divide)],
        [.number("4"), .number("5"), .number("6"), .operation(.multiply)],
        [.number("1"), .number("2"), .number("3"), .operation(.subtract)],
        [.clear, .number("0"), .equal, .operation(.add)]
    ]

    var body: some View {
        VStack(spacing: 12) {
            VStack(alignment: .trailing, spacing: 4) {
                Text(expression)
                    .font(.caption)
                    .foregroundColor(.secondary)

                Text(errorMessage ?? display)
                    .font(.system(size: 44, weight: .medium))
                    .foregroundColor(errorMessage == nil ? .primary : .red)
            }
            .frame(maxWidth: .infinity, alignment: .trailing)
            .padding()
            .background(.ultraThinMaterial)
            .cornerRadius(16)

            ForEach(layout, id: \.self) { row in
                HStack(spacing: 12) {
                    ForEach(row, id: \.self) { button in
                        Button {
                            handle(button)
                        } label: {
                            Text(title(for: button))
                                .frame(maxWidth: .infinity, maxHeight: .infinity)
                                .font(.title3)
                        }
                        .buttonStyle(CalcButtonStyle(button: button))
                    }
                }
            }
        }
        .padding()
        .sheet(isPresented: $showHistory) {
            HistoryView(history: history)
        }
    }

    func handle(_ button: CalcButton) {
        errorMessage = nil

        switch button {
        case .history:
            showHistory = true

        case .number(let value):
            if isNewInput {
                display = value
                isNewInput = false
            } else {
                display += value
            }
            expression += value

        case .decimal:
            if !display.contains(",") {
                display += ","
                expression += ","
            }

        case .operation(let op):
            expression += " \(op.rawValue) "
            isNewInput = true

        case .function(let fn):
            expression += "\(fn.rawValue)("
            isNewInput = true

        case .leftParen:
            expression += "("

        case .rightParen:
            expression += ")"

        case .constant(.pi):
            expression += "π"
            display = format(.pi)
            isNewInput = true

        case .equal:
            let original = expression
            guard let result = evaluateExpression(expression) else {
                errorMessage = "Mathematischer Fehler"
                return
            }
            history.append("\(original) = \(format(result))")
            display = format(result)
            expression = format(result)
            isNewInput = true

        case .clear:
            expression = ""
            display = "0"
            errorMessage = nil
            isNewInput = true
        }
    }

    func evaluateExpression(_ expr: String) -> Double? {
        var sanitized = expr
            .replacingOccurrences(of: "π", with: "\(Double.pi)")
            .replacingOccurrences(of: ",", with: ".")
            .replacingOccurrences(of: "×", with: "*")
            .replacingOccurrences(of: "÷", with: "/")

        sanitized = resolveFunctions(in: sanitized)

        let exp = NSExpression(format: sanitized)
        return exp.expressionValue(with: nil, context: nil) as? Double
    }

    func resolveFunctions(in expr: String) -> String {
        var result = expr

        for fn in [Function.sin, .cos, .tan] {
            while let range = result.range(of: "\(fn.rawValue)(") {
                guard let end = matchingParen(in: result, from: range.upperBound) else { break }
                let inner = String(result[range.upperBound..<end])
                guard let value = Double(inner) else { return "" }

                let computed: Double
                switch fn {
                case .sin: computed = sin(value)
                case .cos: computed = cos(value)
                case .tan: computed = tan(value)
                }

                result.replaceSubrange(range.lowerBound...end, with: "\(computed)")
            }
        }
        return result
    }

    func matchingParen(in text: String, from index: String.Index) -> String.Index? {
        var depth = 1
        var i = index
        while i < text.endIndex {
            if text[i] == "(" { depth += 1 }
            if text[i] == ")" { depth -= 1 }
            if depth == 0 { return i }
            i = text.index(after: i)
        }
        return nil
    }

    func format(_ value: Double) -> String {
        let str = value.truncatingRemainder(dividingBy: 1) == 0
            ? String(Int(value))
            : String(value)
        return str.replacingOccurrences(of: ".", with: ",")
    }

    func title(for button: CalcButton) -> String {
        switch button {
        case .number(let v): return v
        case .operation(let o): return o.rawValue
        case .function(let f): return f.rawValue
        case .constant(let c): return c.rawValue
        case .leftParen: return "("
        case .rightParen: return ")"
        case .decimal: return ","
        case .equal: return "="
        case .clear: return "C"
        case .history: return "≡"
        }
    }
}

struct HistoryView: View {
    let history: [String]

    var body: some View {
        NavigationStack {
            List(history.reversed(), id: \.self) {
                Text($0)
            }
            .navigationTitle("Verlauf")
        }
    }
}

struct CalcButtonStyle: ButtonStyle {
    let button: CalcButton

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.white)
            .background(color)
            .cornerRadius(14)
            .scaleEffect(configuration.isPressed ? 0.95 : 1)
    }

    var color: Color {
        switch button {
        case .number, .decimal: return .gray.opacity(0.7)
        case .operation: return .orange
        case .function, .constant, .leftParen, .rightParen: return .purple
        case .equal: return .blue
        case .clear: return .red
        case .history: return .black.opacity(0.7)
        }
    }
}
```

User

Das einzige Problem was ich momentan mit dieser Version noch habe ist: Wenn man eine Rechnung eingibt und z.B. "1+1+" eingibt stürzt die App wieder ab.

ChatGPT

```swift
import SwiftUI
import Foundation

enum CalcButton: Hashable {
    case number(String)
    case operation(Operation)
    case function(Function)
    case constant(Constant)
    case leftParen
    case rightParen
    case decimal
    case equal
    case clear
    case history
}

enum Operation: String {
    case add = "+", subtract = "−", multiply = "×", divide = "÷"
}

enum Function: String {
    case sin, cos, tan
}

enum Constant: String {
    case pi = "π"
}

@main
struct CalculatorApp: App {
    var body: some Scene {
        WindowGroup {
            CalculatorView()
        }
    }
}

struct CalculatorView: View {
    @State private var expression = ""
    @State private var display = "0"
    @State private var history: [String] = []
    @State private var errorMessage: String?
    @State private var isNewInput = true
    @State private var showHistory = false

    let layout: [[CalcButton]] = [
        [.history, .function(.sin), .function(.cos), .function(.tan)],
        [.constant(.pi), .leftParen, .rightParen, .decimal],
        [.number("7"), .number("8"), .number("9"), .operation(.divide)],
        [.number("4"), .number("5"), .number("6"), .operation(.multiply)],
        [.number("1"), .number("2"), .number("3"), .operation(.subtract)],
        [.clear, .number("0"), .equal, .operation(.add)]
    ]

    var body: some View {
        VStack(spacing: 12) {
            VStack(alignment: .trailing, spacing: 4) {
                Text(expression)
                    .font(.caption)
                    .foregroundColor(.secondary)

                Text(errorMessage ?? display)
                    .font(.system(size: 44, weight: .medium))
                    .foregroundColor(errorMessage == nil ? .primary : .red)
            }
            .frame(maxWidth: .infinity, alignment: .trailing)
            .padding()
            .background(.ultraThinMaterial)
            .cornerRadius(16)

            ForEach(layout, id: \.self) { row in
                HStack(spacing: 12) {
                    ForEach(row, id: \.self) { button in
                        Button {
                            handle(button)
                        } label: {
                            Text(title(for: button))
                                .frame(maxWidth: .infinity, maxHeight: .infinity)
                                .font(.title3)
                        }
                        .buttonStyle(CalcButtonStyle(button: button))
                    }
                }
            }
        }
        .padding()
        .sheet(isPresented: $showHistory) {
            HistoryView(history: history)
        }
    }

    func handle(_ button: CalcButton) {
        errorMessage = nil

        switch button {
        case .history:
            showHistory = true

        case .number(let value):
            if isNewInput {
                display = value
                isNewInput = false
            } else {
                display += value
            }
            expression += value

        case .decimal:
            if !display.contains(",") {
                display += ","
                expression += ","
            }

        case .operation(let op):
            guard !expression.isEmpty,
                  !expression.hasSuffix("+"),
                  !expression.hasSuffix("−"),
                  !expression.hasSuffix("×"),
                  !expression.hasSuffix("÷"),
                  !expression.hasSuffix("(")
            else { return }
            expression += " \(op.rawValue) "
            isNewInput = true

        case .function(let fn):
            expression += "\(fn.rawValue)("
            isNewInput = true

        case .leftParen:
            expression += "("

        case .rightParen:
            guard balancedParentheses(afterAppending: ")") else { return }
            expression += ")"

        case .constant(.pi):
            expression += "π"
            display = format(.pi)
            isNewInput = true

        case .equal:
            let trimmed = trimInvalidEnd(expression)
            guard !trimmed.isEmpty else {
                errorMessage = "Ungültiger Ausdruck"
                return
            }

            guard let result = evaluateExpression(trimmed) else {
                errorMessage = "Mathematischer Fehler"
                return
            }

            history.append("\(trimmed) = \(format(result))")
            expression = format(result)
            display = format(result)
            isNewInput = true
        case .clear:
            expression = ""
            display = "0"
            errorMessage = nil
            isNewInput = true
        }
    }

    func trimInvalidEnd(_ expr: String) -> String {
        var result = expr.trimmingCharacters(in: .whitespaces)
        while result.hasSuffix("+")
            || result.hasSuffix("-")
            || result.hasSuffix("×")
            || result.hasSuffix("÷")
            || result.hasSuffix("(") {
            result.removeLast()
            result = result.trimmingCharacters(in: .whitespaces)
        }
        return result
    }

    func balancedParentheses(afterAppending char: String) -> Bool {
        let test = expression + char
        let open = test.filter { $0 == "(" }.count
        let close = test.filter { $0 == ")" }.count
        return close <= open
    }

    func evaluateExpression(_ expr: String) -> Double? {
        var sanitized = expr
            .replacingOccurrences(of: "π", with: "\(Double.pi)")
            .replacingOccurrences(of: ",", with: ".")
            .replacingOccurrences(of: "×", with: "*")
            .replacingOccurrences(of: "÷", with: "/")

        sanitized = resolveFunctions(in: sanitized)
        guard isValidMathExpression(sanitized) else { return nil }

        let nsExpr = NSExpression(format: sanitized)
        return nsExpr.expressionValue(with: nil, context: nil) as? Double
    }

    func isValidMathExpression(_ expr: String) -> Bool {
        let invalidPatterns = ["++", "--", "**", "//", "+*", "*/", "/+", "-*", "*+"]
        for pattern in invalidPatterns {
            if expr.contains(pattern) { return false }
        }
        return true
    }

    func resolveFunctions(in expr: String) -> String {
        var result = expr

        for fn in [Function.sin, .cos, .tan] {
            while let range = result.range(of: "\(fn.rawValue)(") {
                guard let end = matchingParen(in: result, from: range.upperBound) else { return "" }
                let inner = String(result[range.upperBound..<end])
                guard let value = Double(inner) else { return "" }

                let computed: Double
                switch fn {
                case .sin: computed = sin(value)
                case .cos: computed = cos(value)
                case .tan: computed = tan(value)
                }

                result.replaceSubrange(range.lowerBound...end, with: "\(computed)")
            }
        }
        return result
    }

    func matchingParen(in text: String, from index: String.Index) -> String.Index? {
        var depth = 1
        var i = index
        while i < text.endIndex {
            if text[i] == "(" { depth += 1 }
            if text[i] == ")" { depth -= 1 }
            if depth == 0 { return i }
            i = text.index(after: i)
        }
        return nil
    }

    func format(_ value: Double) -> String {
        let str = value.truncatingRemainder(dividingBy: 1) == 0
            ? String(Int(value))
            : String(value)
        return str.replacingOccurrences(of: ".", with: ",")
    }

    func title(for button: CalcButton) -> String {
        switch button {
        case .number(let v): return v
        case .operation(let o): return o.rawValue
        case .function(let f): return f.rawValue
        case .constant(let c): return c.rawValue
        case .leftParen: return "("
        case .rightParen: return ")"
        case .decimal: return ","
        case .equal: return "="
        case .clear: return "C"
        case .history: return "≡"
```

~~~~swift
        }
    }
}

struct HistoryView: View {
    let history: [String]

    var body: some View {
        NavigationStack {
            List(history.reversed(), id: \.self) {
                Text($0)
            }
            .navigationTitle("Verlauf")
        }
    }
}

struct CalcButtonStyle: ButtonStyle {
    let button: CalcButton

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.white)
            .background(color)
            .cornerRadius(14)
            .scaleEffect(configuration.isPressed ? 0.95 : 1)
    }

    var color: Color {
        switch button {
        case .number, .decimal: return .gray.opacity(0.7)
        case .operation: return .orange
        case .function, .constant, .leftParen, .rightParen: return .purple
        case .equal: return .blue
        case .clear: return .red
        case .history: return .black.opacity(0.7)
        }
    }
}
```
~~~~