

```
1 //
2 // MultiplicationMaestroApp.swift
3 // MultiplicationMaestro
4 //
5
6 import SwiftUI
7
8 @main
9 struct MultiplicationMaestroApp: App {
10     var body: some Scene {
11         WindowGroup {
12             ContentView()
13         }
14     }
15 }
```

```
1 //
2 // ResultVisibility.swift
3 // MultiplicationMaestro
4 //
5
6 import Foundation
7
8 // An enumeration is like a Bool, or boolean value, in that
9 // an enumeration has a defined list of possible values.
10 // The advantage is that with an enumeration, we can control
11 // what the possible values are.
12
13 // ResultVisibility is used to determine what results
14 // should be visible in the list of results below the
15 // question and answer interface.
16 enum ResultVisibility: String {
17     case all = "All"
18     case incorrect = "Incorrect"
19     case correct = "Correct"
20 }
```

```
1 //
2 // Result.swift
3 // MultiplicationMaestro
4 //
5
6 import Foundation
7
8 // Tracks the result of answering a single question
9 struct Result: Identifiable {
10
11     // MARK: Stored properties
12     let id = UUID()
13     let multiplicand: Int
14     let multiplier: Int
15     let inputGiven: String
16     let answerCorrect: Bool
17
18     // MARK: Computed properties
19     var correctProduct: Int {
20         return multiplicand * multiplier
21     }
22
23 }
```

```

1 //
2 // ContentView.swift
3 // MultiplicationMaestro
4 //
5
6 import SwiftUI
7
8 struct ContentView: View {
9
10     // MARK: Stored properties
11
12     // Values to be multiplied
13     @State var multiplicand = Int.random(in: 1...12)
14     @State var multiplier = Int.random(in: 1...12)
15
16     // Holds the user's input
17     @State var inputGiven = ""
18
19     // Tracks whether the input has even been checked yet
20     @State var answerChecked: Bool = false
21
22     // Tracks whether the provided answer is correct or not
23     @State var answerCorrect: Bool = false
24
25     // Tracks the results of all questions answered so far
26     @State var results: [Result] = []
27
28     // Tracks what results should be visible currently
29     @State private var selectedResultVisibility: ResultVisibility = .all
30
31     // MARK: Computed properties
32
33     // Gets updated as new values are randomly generated
34     var correctProduct: Int {
35         return multiplicand * multiplier
36     }
37
38     // The main user interface
39     var body: some View {
40
41         VStack(spacing: 0) {
42
43             // Operation, values to be multiplied
44             HStack {
45                 Text("x")
46
47                 Spacer()
48
49                 VStack(alignment: .trailing) {
50                     Text("\(multiplicand)")
51                     Text("\(multiplier)")
52                 }
53             }
54
55             // Horizontal line
56             Divider()
57
58             // Result and input area
59             HStack {
60
61                 ZStack {
62                     Image(systemName: "checkmark.circle")
63                         .foregroundColor(.green)
64                     // Only show this when the answer given is correct

```

```

65         //          CONDITION          true false
66         .opacity(answerCorrect == true ? 1.0 : 0.0)
67
68     Image(systemName: "x.square")
69         .foregroundColor(.red)
70         // Show this when both of the following situations are true:
71         // 1. Answer has been checked.
72         // 2. Answer was not correct.
73         // Necessary since if we show this only when an answer is incorrect,
74         // with no other conditions, it would show as soon as a new
75         // question is generated.
76         //          CONDITION1 AND CONDITION2          true false
77         .opacity(answerChecked == true && answerCorrect == false ? 1.0 : 0.0)
78     }
79
80     Spacer()
81
82     TextField("", text: $inputGiven)
83         // Ensure input is right-aligned
84         .multilineTextAlignment(.trailing)
85 }
86
87 // Buttons to control program
88 ZStack {
89
90     // Allow input to be checked
91     Button(action: {
92
93         // If we've gotten to this point, the answer has at least been checked
94         answerChecked = true
95
96         // Convert the provided input (String) into integer (Int) if possible
97         guard let answerGiven = Int(inputGiven) else {
98             // User gave invalid input (e.g.: typed 'mangos' rather than 5)
99             answerCorrect = false
100            // Save this result
101            saveResult()
102            // Stop checking the answer
103            return
104        }
105
106        // Is the integer given actually correct?
107        if answerGiven == correctProduct {
108            answerCorrect = true
109        } else {
110            answerCorrect = false
111        }
112
113        // Save this result
114        saveResult()
115
116    }, label: {
117        Text("Check Answer")
118            .font(.largeTitle)
119    })
120    // Only show this button when an answer has not been checked
121    .opacity(answerChecked == false ? 1.0 : 0.0)
122    .padding()
123    .buttonStyle(.bordered)
124
125    // Allow new question to be generated
126    Button(action: {
127
128        // Generate a new question
129        multiplicand = Int.random(in: 1...12)

```

```

130     multiplier = Int.random(in: 1...12)
131
132     // Reset properties that track what's happening with the current question
133     answerChecked = false
134     answerCorrect = false
135
136     // Reset the input field
137     inputGiven = ""
138
139     }, label: {
140         Text("New question")
141             .font(.largeTitle)
142     })
143     .padding()
144     .buttonStyle(.bordered)
145     // Only show this button when an answer has been checked
146     .opacity(answerChecked == true ? 1.0 : 0.0)
147
148 }
149
150 // Control filtering of prior tasks
151 VStack {
152     // Label for picker
153     Text("Filter by...")
154         .font(Font.caption.smallCaps())
155         .foregroundColor(.secondary)
156
157     // Picker to allow user to select what tasks to show
158     Picker("Filter", selection: $selectedResultVisibility) {
159         Text(ResultVisibility.all.rawValue)
160             .tag(ResultVisibility.all)
161         Text(ResultVisibility.incorrect.rawValue)
162             .tag(ResultVisibility.incorrect)
163         Text(ResultVisibility.correct.rawValue)
164             .tag(ResultVisibility.correct)
165     }
166     .pickerStyle(.segmented)
167     .padding(.horizontal)
168 }
169 .padding(.bottom)
170
171 // Show results of prior questions attempted
172 List(filter(results, by: selectedResultVisibility)) { result in
173     HStack {
174         Text("\(result.multiplicand)")
175         Text("x")
176         Text("\(result.multiplier)")
177         Text("=")
178         Text("\(result.inputGiven)")
179         Text("( \(result.correctProduct) )")
180             .opacity(result.answerCorrect == false ? 1.0 : 0.0)
181         Spacer()
182         ZStack {
183             Image(systemName: "checkmark.circle")
184                 .foregroundColor(.green)
185                 .opacity(result.answerCorrect == true ? 1.0 : 0.0)
186
187             Image(systemName: "x.square")
188                 .foregroundColor(.red)
189                 .opacity(result.answerCorrect == false ? 1.0 : 0.0)
190         }
191     }
192     .font(.title)
193 }
194

```

```

195     }
196     .padding(.horizontal)
197     .font(.system(size: 72))
198
199 }
200
201 // MARK: Functions
202
203 // Save the result of a question that has been answered
204 func saveResult() {
205
206     // Create a result to save based on current question state
207     let newResult = Result(multiplicand: multiplicand,
208                            multiplier: multiplier,
209                            inputGiven: inputGiven,
210                            answerCorrect: answerCorrect)
211
212     // Ensure most recent result is always at top of the list
213     results.insert(newResult, at: 0)
214
215 }
216
217 // Filter the list of results to be shown
218 func filter(_ listOfResults: [Result], by visibility: ResultVisibility) -> [Result] {
219
220     // When the user wants to see all results, just return the list provided
221     if visibility == .all {
222         return listOfResults
223     } else {
224
225         // Create an empty list of results
226         var filteredResults: [Result] = []
227
228         // Iterate over the list of results, and build a new list
229         // that only includes the selected type of result
230         for currentResult in listOfResults {
231
232             if visibility == .correct && currentResult.answerCorrect == true {
233                 filteredResults.insert(currentResult, at: 0)
234             } else if visibility == .incorrect && currentResult.answerCorrect == false {
235                 filteredResults.insert(currentResult, at: 0)
236             }
237
238         }
239
240         // Return the filtered list of results
241         return filteredResults
242     }
243
244 }
245 }
246
247 struct ContentView_Previews: PreviewProvider {
248     static var previews: some View {
249         ContentView()
250     }
251 }

```