In Model:

let brain: Brain

let stupidBrain: Brain

init() {

brain = Brain(precision: 0)

brain.setPrecision(precision)

stupidBrain = Brain(precision: 0)

stupidBrain.setPrecision(stupidPrecision)

**if** memoryValue == "" {

brain.memory = **nil**

stupidBrain.memory = **nil**

} **else** {

brain.memory = Number(memoryValue, precision: precision)

stupidBrain.memory = Number(memoryValue, precision: stupidPrecision)

}

}

Unchanged: copyToPastBin, fromPastBin and speedTest

**func** updateDisplayData() {

**let** temp = brain.last.getDisplayData(

forLong: !screenInfo.isPortraitPhone,

lengths: lengths,

forceScientific: forceScientific,

showAsInteger: showAsInteger,

showAsFloat: showAsFloat)

DispatchQueue.main.async {

**self**.displayData = temp

}

}

**func** haveResultCallback() {

**if** brain.last.isNull {

DispatchQueue.main.async {

**self**.showAC = **true**

**self**.precisionDescription = **self**.precision.useWords

}

} **else** {

DispatchQueue.main.async {

**self**.showAC = **false**

}

}

updateDisplayData()

**for** key **in** C.keysAll {

**if** brain.isValidNumber {

keyInfo[key]!.enabled = **true**

} **else** {

**if** C.keysThatRequireValidNumber.contains(key) {

keyInfo[key]!.enabled = **false**

} **else** {

keyInfo[key]!.enabled = **true**

}

}

}

// check mr

keyInfo["mr"]!.enabled = brain.memory != **nil**

}

**func** pendingOperatorCallback(op: String?) {

/// In the brain, we have already asserted that the new op is different from previous

/// Set the previous one back to normal?

**if** **let** previous = previous {

DispatchQueue.main.async {

**if** C.keysWithPendingOperations.contains(previous) {

**self**.keyInfo[previous]!.colors = C.scientificColors

} **else** {

**self**.keyInfo[previous]!.colors = C.operatorColors

}

}

}

/// Set the colors for the pending operation key

**if** **let** op = op {

DispatchQueue.main.async {

**if** C.keysWithPendingOperations.contains(op) {

**self**.keyInfo[op]!.colors = C.pendingScientificColors

} **else** {

**self**.keyInfo[op]!.colors = C.pendingOperatorColors

}

}

}

previous = op

}

**func** pressed(\_ \_symbol: String) {

**let** symbol = ["sin", "cos", "tan", "asin", "acos", "atan"].contains(\_symbol) && !rad ? \_symbol+"D" : \_symbol

**switch** symbol {

**case** "2nd":

secondActive.toggle()

**self**.keyInfo["2nd"]!.colors = secondActive ? C.secondActiveColors : C.secondColors

**case** "Rad":

hasBeenReset = **false**

rad = **true**

**case** "Deg":

hasBeenReset = **false**

rad = **false**

**case** "plusKey":

**break**

**default**:

**if** !isCalculating {

**if** symbol == "AC" {

hasBeenReset.toggle()

} **else** {

hasBeenReset = **false**

}

Task {

DispatchQueue.main.async { **self**.isCalculating = **true** }

**await** asyncOperation(symbol)

**if** ["mc", "m+", "m-"].contains(symbol) {

**if** **let** memory = brain.memory {

**let** temp = memory.getDisplayData(

forLong: **true**,

lengths: Lengths(precision),

forceScientific: **false**,

showAsInteger: showAsInteger,

showAsFloat: showAsFloat,

maxDisplayLength: precision)

DispatchQueue.main.sync {

memoryValue = temp.**left** + (temp.right ?? "")

}

} **else** {

DispatchQueue.main.sync {

memoryValue = ""

}

}

}

}

}

}

}

**func** asyncOperation(\_ symbol: String) **async** {

brain.operation(symbol)

DispatchQueue.main.async { **self**.isCalculating = **false** }

}