# Assignment I: Calculator

## Ziele

Ziel des Assignments ist die Demo aus der Vorlesung nachzubauen und dann ein paar Erweiterungen hinzuzufügen. Es ist wichtig, dass Sie jeden Schritt beim Erstellen der Demo verstehen, so dass Sie gut vorbereitet sind, die Erweiterungen durchzuführen.

Ein weiteres Ziel ist den Umgang mit Xcode zu verstehen. Dazu gehört auch das Anlegen eines neuen Projektes. Nutzen Sie kein Copy & Paste, sondern tippen Sie den Code ein und beobachten Xcode während Sie dies tun.

Das Testat für dieses Assignment bekommen Sie im Praktikum. Fangen Sie rechtzeitig mit der Vorbereitung an (am besten direkt nach der Vorlesung). Beachten Sie, dass nicht nur eine laufende Lösung berücksichtigt wird, Sie sollten Ihre Lösung auch vollständig verstanden haben. Nach erfolgtem Testat, laden Sie Ihre Lösung in Moodle hoch.

Stellen Sie sicher auch die Tipps Sektion weiter unten zu lesen.

Lesen Sie ebenfalls aufmerksam die Sektion Testat weiter unten, um sicherzustellen dass Sie alle Anforderungen erfüllen.

## Material

- Sie müssen die (freie) Entwicklungsumgebung Xcode 8 über den App Store auf Ihrem Mac (oder VM) installieren (frühere Versionen von Xcode funktionieren **nicht**). Im Labor ist dies bereits geschehen. Es ist dringend empfohlen, dies so schnell wie möglich zu machen, am besten direkt nach der ersten Vorlesung.

# Aufgaben

1. Bauen Sie den Taschenrechner nach, wie in der Vorlesung gezeigt.
2. Ihr Taschenrechner funktioniert bereits mit Gleitkommazahlen (d.h., wenn Sie `3 ÷ 4 =` eintippen, zeigt er 0.75 korrekt an). Jedoch gibt es bisher keine Möglichkeit eine Gleitkommazahl direkt *einzugeben*. Beheben Sie dies, indem Sie erlauben eine valide Gleitkommazahl eingeben zu können (d.h. "`192.168.0.1`" ist **keine** valide Gleitkommazahl!). Dazu brauchen Sie einen "`.`" Button in Ihrem Taschenrechner.
3. Fügen Sie weitere Operation Buttons zu Ihrem Taschenrechner hinzu, so dass dieser mindestens ein Dutzent Operationen hat (mehr, wenn Sie möchten). Sie können beliebige Operationen auswählen. Die Buttons müssen jedoch ein sauber ausgerichtetes Layout im Portrait und Landscape Mode auf allen iPhone 6's und 7's haben.
4. Verwenden Sie Farbe, damit Ihr UI gut aussieht. Mindestens müssen Ihre Operation Buttons eine andere Farbe als normale Keypad Buttons haben. Ansonsten können Sie Farben beliebig verwenden.
5. Fügen Sie ein `Bool` Property mit dem Namen `resultIsPending` zu Ihrem `CalculatorBrain` hinzu, welches zurückgibt ob eine Binäroperation ausgeführt wird.
6. Fügen Sie ein `String` Property mit dem Namen `description` zu Ihrem `CalculatorBrain` hinzu, welches die Beschreibung einer Sequenz von Operanden und Operationen zurückgibt, die beschreibt wie der Wert welcher von `result` (oder das Zwischenergebnis falls `resultIsPending`) zurückgegeben wurde, berechnet wurde. Der Character `=` (Gleichheitszeichen) sollte niemals in `description` auftauchen, ebenso wie `...` (Auslassungspunkte).
7. Implementieren Sie ein `UILabel` in Ihrem UI welches die Sequenz von Operanden und Operationen zeigt, die zum Ergebniss führte (oder führen wird, wenn `resultIsPending`) und im `display` angezeigt (oder angezeigt werden wird, wenn `resultIsPending`) wird. Wenn `resultIsPending` `true` ist, fügen Sie `...` ans Ende des `UILabel`s an, andernfalls fügen Sie `=` an. Wenn der `userIsInTheMiddleOfTyping` kann das `UILabel` das anzeigen was bereits dort stand, bevor der User mit der Eingabe begonnen hat. Beispiele...
   a. Eingabe von 7 + zeigt "7 + ..." (mit 7 noch im display)
   b. 7 + 9 zeigt "7 + ..." (mit 9 noch im display)
   c. 7 + 9 = zeigt "7 + 9 =" (16 im display)
   d. 7 + 9 = √ zeigt "√(7 + 9) =" (4 im display)
   e. 7 + 9 = √ + 2 = zeigt "√(7 + 9) + 2 = (6 im display)
   f. 7 + 9 √ zeigt "7 + √(9) ..." (3 im display)
   g. 7 + 9 √ = zeigt "7 + √(9) =" (10 im display)
   h. 7 + 9 = + 6 = + 3 = zeigt "7 + 9 + 6 + 3 =" (25 im display)
   i. 7 + 9 = √ 6 + 3 = zeigt "6 + 3 =" (9 im display)
   j. 5 + 6 = 7 3 zeigt "5 + 6 =" (73 im display)
   k. 4 × π = zeigt "4 × π =" (12.5663706143592 im display)
8. Fügen Sie einen `C` Button hinzu, der alles löscht (Ihr `display`, das neue `UILabel` welches Sie hinzugefügt haben, jede ausstehende binäre Operation, etc.). Idealerweise versetzt dies Ihren Calculator in den gleichen Zustand wie nach dem Start.

# Tipps

1. Die `String` Methode `contains(String)` ist Ihnen vielleicht eine große Hilfe für die Gleitkommazahl Aufgabe in diesem Assignment.
2. Die Gleitkommazahl Aufgabe kann in einer einzigen Zeile Code (eine öffnende oder schließende Klammer allein in einer Zeile wird nicht als Codezeile betrachtet) implementiert werden. Beachten Sie, dass dies nur ein Tipp und keine Anforderung ist.
3. Testen Sie ebefalls den Fall (und behandeln ihn entsprechend), wenn der User beginnt eine neue Zahl einzugeben und zwar durch tippen des Dezimalpunktes (z.B. tippt der User den Dezimalpunkt wenn `userIsInTheMiddleOfTyping` `false` ist).
4. Der einfachste Weg eine Bug-freie Zeile Code zu schreiben, ist die Zeile gar nicht zu schreiben. Das gesamte Assignment kann mit ein wenigen Dutzent Zeilen Code (vielleicht sogar zwei Dutzent) gelöst werden. Wenn Sie also mehr als 100 Zeilen Code schreiben, dann haben Sie wahrscheinlich einen falschen Lösungsansatz gewählt.
5. Sie können jedes beliebige Unicode Zeichen als mathematisches Symbol verwedenden. So sind z.B. $x^2$ und $x^{-1}$ valide mathematische Symbole.
6. Wenn Sie den Text eines `UILabel`s auf `nil` oder eine leeren String setzen, wird ein resize auf 0 Height durchgeführt (bewegt das restliche UI entsprechend umher). Dies ist wahrscheinlich unangenehm für die User. Wenn Sie ein `UILabel` leer aussehen lassen wollen, aber nicht Höhe 0, dann setzen Sie dessen `text` einfach auf " " (Space).
7. Vielleicht möchten Sie einen Call zu `performPendingBinaryOperation()` für Ihren Fall `binaryOperation` hinzufügen (so dass 6 × 5 × 4 × 3 = funktioniert).
8. Falls Sie besorgt sind, dass Sie etwas durch Ihre Änderungen zerstört haben, behalten Sie eine Version des Calculators mit dem Code aus der Demo und vergleichen Sie dies mit Ihrer modifizierten Version.
9. Keine der Aufgaben verlangt von Ihnen den Umgang mit Input der Fehler verursacht (z.B. Division-by-Zero oder Wurzel einer negativen Zahl). Nehmen Sie an, dass diese Fälle bei der Abnahme nicht getestet werden.
10. Ein einfacher Weg sich `description` des `CalculatorBrain` zu denken ist als eine `String` Repräsentation, welche zu dem geführt hat was im `accumulator` ist. Wenn Sie dies so behandeln, dann müssen Sie lediglich jedes mal diese String Repräsentation erstellen, wenn Sie den Wert des `accumulator` setzen. Der einzige "Haken" ist, wenn Sie eine `pendingBinaryOperation` haben (diesen Fall müssen Sie also gesondert behandeln).
11. Wenn Sie die Vorgehensweise im obigen Tipp nutzen um `description` zu implementieren, dann haben Sie zwei Variablen (`accumulator` und seine `String` Repräsentation), deren Werte immer zur gleichen Zeit gesetzt werden und die niemals asynchron zueinander sein sollten. Swift hat eine großartige Datenstruktur für Variablen die zusammen gehören: ein Tuple. Dies haben wir wahrscheinlich noch nicht in der Vorlesung besprochen, erwägen Sie aber dessen Nutzung, wenn Sie herausfinden können, wie sie funktioniert.
12. Hier ist noch etwas, was wir wahrscheinlich noch nicht besprochen haben, Sie aber selbst herausfinden können: verwenden Sie den `??` Operator (Defaulting den Wert eines Optionals) mindestens einmal in Ihrer Lösung.

# Lernziele

Hier ist eine unvollständige Liste von Konzepten, die Sie beim Lösen des Assignments vertiefen bzw. lernen sollen.

1. Xcode
2. Swift
3. Target/Action
4. Outlets
5. `UILabel`
6. `UIViewController`
7. Funktionen und Properties (Instanzvariablen)
8. Computed vs. Stored Properties
9. `let` vs. `var`
10. `enum`, `struct` und `class`
11. Optionals
12. `String` und `Dictionary`
13. Tuples (hoffentlich)
14. `??` Operator (hoffentlich)

# Testat

Für alle Testate in diesem Semester ist das Ziel qualitativ hochwertigen Code zu schreiben, der ohne Warnings und Errors baut. Ebenfalls sollte die so entstandene App auf Funktion und Fehler gestestet werden und dies iterativ so lange, bis alle Fehler beseitigt sind und die App so funktioniert, wie gedacht.

Hier sind die am meisten vorkommenden Gründe, weshalb Sie vielleicht kein Testat bekommen:

- Projekt baut nicht
- Eine oder mehrere Aufgaben wurden nicht zufriedenstellend gelöst
- Ein fundamentales Konzept wurde nicht verstanden
- Code ist optisch mangelhaft und schwer zu lesen (z.B. ist das Einrücken nicht konsistent, etc.)
- Ihre Lösung ist schwer (oder unmöglich) für jemanden zu lesen, durch mangelhafte Kommentare, schlechte Variablen/Methoden Namen, schlechte Lösungsstruktur, zu lange Methoden, etc.
- Die Swift API Design Guidelines wurden nicht eingehalten
- Globale Variablen/Methoden, d.h. private API wurde nicht mittels des Schlüsselwortes `private` deklariert

Häufig fragen Studenten wie viele Kommentare im Code nötig sind. Die Antwort ist einfach. Der Code muss einfach und vollständig lesbar sein. Sie können davon ausgehen, dass die iOS API und der Code aus der Vorlesung bekannt sind. Sie sollten nicht davon ausgehen, dass bereits eine Lösung für die Aufgabe bekannt ist.