Assignment 1: Calculator

Objective

The goal of this assignment is to recreate the demonstration given in lecture and then make some small enhancements. It is important that you understand what you are doing with each step of recreating the demo from lecture so that you are prepared to do the enhancements.

Another goal is to get experience creating a project in Xcode and typing code in from scratch. Do not copy/paste any of the code from anywhere. Type it in and watch what Xcode does as you do.

This assignment must be submitted using the submit script described here by the start of lecture next Wednesday (i.e before lecture 4). You may submit it multiple times if you wish. Only the last submission will be counted. For example, it might be a good idea to go ahead and submit it after you have reproduced what was shown in lecture and gotten that part working (even before you attempt the enhancements). If you wait until the last minute to try to submit and you have problems with the submission script, you'll likely have to use one of your valuable free late days.

Be sure to review the Hints section below!

Also, check out the latest in the Evaluation section to make sure you understand what you are going to be evaluated on with this assignment.

Materials

- You will need to install the (free) program Xcode 7 using the App Store on your Mac (Xcode 6 will NOT work). It is highly recommended that you do this immediately so that if you have any problems getting Xcode to work, you have time to get help from Piazza and/or the TAs in their office hours.
- A link to the video of the lectures can be found in the same place you found this document.

Required Tasks

1. Get the Calculator working as demonstrated in lectures 1 and 2.
2. Your calculator already works with floating point numbers (e.g. if you touch `3 ÷ 4 =`, it will properly show 0.75), however, there is no way for the user to *enter* a floating point number directly. Fix this by allowing *legal* floating point numbers to be entered (e.g. "`192.168.0.1`" is **not** a legal floating point number!). You will have to add a new "`.`" button to your calculator. Don't worry too much about precision or significant digits in this assignment (including in the examples below).
3. Add some more operations buttons to your calculator such that it has at least a dozen operations total (it can have even more if you like). You can choose whatever operations appeal to you. The buttons must arrange themselves nicely in portrait and landscape modes on all iPhones.
4. Use color to make your UI look nice. At the very least, your operations buttons must be a different color than your keypad buttons, but otherwise you can use color in whatever way you think looks nice.
5. Add a `String` property to your `CalculatorBrain` called `description` which returns a description of the sequence of operands and operations that led to the value returned by `result`. "=" should never appear in this description, nor should "…".
6. Add a `Bool` property to your `CalculatorBrain` called `isPartialResult` which returns whether there is a binary operation pending (if so, return `true`, if not, `false`).
7. Use the two properties above to implement a `UILabel` in your UI which shows the sequence of operands and operations that led to what is showing in the `display`. If `isPartialResult`, put . . . on the end of the `UILabel`, else put =. If the `userIsInTheMiddleOfTypingANumber`, you can leave the `UILabel` showing whatever was there before the user started typing the number. Examples …
   a. touching 7 + would show "7 + …" (with 7 still in the display)
   b. 7 + 9 would show "7 + …" (9 in the display)
   c. 7 + 9 = would show "7 + 9 =" (16 in the display)
   d. 7 + 9 = √ would show "√(7 + 9) =" (4 in the display)
   e. 7 + 9 √ would show "7 + √(9) …" (3 in the display)
   f. 7 + 9 √ = would show "7 + √(9) =" (10 in the display)
   g. 7 + 9 = + 6 + 3 = would show "7 + 9 + 6 + 3 =" (25 in the display)
   h. 7 + 9 = √ 6 + 3 = would show "6 + 3 =" (9 in the display)
   i. 5 + 6 = 7 3 would show "5 + 6 =" (73 in the display)
   j. 7 + = would show "7 + 7 =" (14 in the display)
   k. 4 × π = would show "4 × π =" (12.5663706143592 in the display)
   l. 4 + 5 × 3 = would show "4 + 5 × 3 =" (27 in the display)
   m. 4 + 5 × 3 = could also show "(4 + 5) × 3 =" if you prefer (27 in the display)

8. Add a `C` button that clears everything (your `display`, the new `UILabel` you added above, etc.). The Calculator should be in the same state as it is at application startup after you touch this new button.

Hints

1. The `String` method `rangeOfString(String)` might be of great use to you for the floating point part of this assignment. It returns an Optional. If the passed `String` argument cannot be found in the receiver, it returns `nil` (otherwise don't worry about what it returns for now).
2. The floating point requirement can be implemented in a single line of code. Note that what you are reading right now is a Hint, not a Required Task. Still, see if you can figure out how to implement it in one or two lines (a curly brace on a line by itself is not considered a "line of code").
3. Be careful of the case where the user starts off entering a new number by touching the decimal point, e.g., they want to enter the number `.5` into the calculator. It might well be that your solution "just works" but be sure to test this case.
4. Economy is valuable in coding. The easiest way to ensure a bug-free line of code is not to write that line of code at all. This entire assignment (not including Extra Credit) can be done in a few dozen lines of code, so if you find yourself writing more than 100 lines of code, you might be on the wrong track.
5. You can use any Unicode characters you want as your mathematical symbols for your operations. For example, x^2 and x^{-1} are perfectly fine mathematical symbols.
6. If you set a `UILabel`'s `text` to `nil` or `""` (the empty string), it will resize to have zero height (shifting the rest of your UI around accordingly). You may find this disconcerting for your users. If you want a `UILabel` to appear empty, but not be zero height, simply set its `text` to be `" "` (space).
7. If you put two (or more) things in a Stack View and you want one (or more) of them to be as small as possible and the rest to use up the remaining space, you can do this by setting the Content Hugging Priority (CHP) of each of the things in the Stack View accordingly. A high(er) number for CHP means "make this be as small as it can be and still fit its contents". A low(er) number means "while laying this out, you can stretch it as much as you want". The CHP for a given button or label (or whatever) can be set in the Dimensions Inspector in the Utilities (right side) panel in Xcode. `UIButton`s have a default CHP of 250. `UILabel`s have a default CHP of 251. So if you had a button and a label in the same stack view, by default the button would be huge and the label would be small. So if wanted the opposite (small button, large label), you'd want to change the button's CHP from 250 to 252 (or higher, but just so that it's higher than the label's 251). Or you could change the label's CHP to 249 (or lower). This is a Hint, not a Required Task. It is quite possible (even likely) that you will implement your entire homework assignment without paying any attention to CHP at all. So if this is confusing you, ignore it.

Things to Learn

Here is a partial list of concepts this assignment is intended to let you gain practice with or otherwise demonstrate your knowledge of.

1. Xcode
2. Swift
3. Target/Action
4. Outlets
5. `UILabel`
6. `UIViewController`
7. Classes
8. Functions and Properties (instance variables)
9. `let` versus `var`
10. Optionals
11. Computed vs. Stored properties
12. `String` and `Dictionary`

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- Project does not build without warnings.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Code is visually sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, long methods, etc.

Often students ask "how much commenting of my code do I need to do?" The answer is that your code must be easily and completely understandable by anyone reading it. You can assume that the reader knows the SDK, but should not assume that they already know the (or a) solution to the problem.

Extra Credit

We try to make Extra Credit be opportunities to expand on what you've learned this week. Attempting at least some of these each week is highly recommended to get the most out of this course.

1. Implement a "backspace" button for the user to touch if they hit the wrong digit button. This is not intended to be "undo," so if the user hits the wrong operation button, he or she is out of luck! It is up to you to decide how to handle the case where the user backspaces away the entire number they are in the middle of typing, but having the `display` go completely blank is probably not very user-friendly. You will find the Strings and Characters section of the Swift Reference Guide to be very helpful here.
2. Change the computed instance variable `displayValue` to be an Optional `Double` rather than a `Double`. Its value should be `nil` if the contents of `display.text` cannot be interpreted as a `Double`. *Setting* its value to `nil` should clear the `display` out. You'll have to modify the code that uses `displayValue` accordingly.
3. Figure out from the documentation how to use the `NSNumberFormatter` class to format your `display` so that it only shows 6 digits after the decimal point (instead of showing all digits that can be represented in a `Double`). This will eliminate the need for Autoshrink in your `display`. While you're at it, make it so that numbers that are integers don't have an unnecessary ".0" attached to them (e.g. show "4" rather than "4.0" as the result of the square root of sixteen). You can do all this for your `description` in the `CalculatorBrain` as well.
4. Make one of your operation buttons be "generate a random number between 0 and 1". This operation button is not a constant (since it changes each time you invoke it). Nor is it a unary operation (since it does not operate on anything).