Assignment I: 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.

Do not copy/paste any of the code from anywhere. Type it in and watch what Xcode does as you do.

Required Tasks

1. Get the Calculator working as demonstrated in lecture.

2. Your calculator should work 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.

3. Add the following operations to your Calculator:

   a. `sin`: calculates the sine of the top operand on the stack

   b. `cos`: calculates the cosine of the top operand on the stack

   c. `π`: calculates (well, conjures up) the value of π. For example, `3 × π` should put three times the value of π into the display on your calculator.

4. 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.

Required Tasks

4. Add a `UILabel` to your UI which shows a history of every operand *and operation* input by the user. Place it at an appropriate location in your UI.
5. 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. 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. Be sure to test this case.
2. `sin()` and `cos()` are standard functions that are available in Swift to perform those operations for you.
3. The value of π is available via the expression `Double.pi`. E.g. `let x = Double.pi`
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 assignment requires very, very few lines of code, so if you find yourself writing dozens of lines of code, you are on the wrong track.

Extra Credit

1. Add a `+/-` operation which changes the sign of the number in the `display`. Be careful with this one. If the user is in the middle of entering a number, you probably want to change the sign of that number and allow typing to continue, not force an enter like other operations do. On the other hand, if the user is not in the middle of typing a number, then this operation would work just like any other unary operation.
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` (you'll need to use the documentation to understand the `NSNumberFormatter` code). *Setting* its value to `nil` should clear the `display` out.