Assignment I: Calculator (Калькулятор)

Цель:

Цель задания состоит в воспроизведении демонстрационный примера, данного на лекции, и небольшого его усовершенствования. Очень важно, чтобы вы понимали что вы делаете на каждом шаге восроизведения демонстрационного примера с лекции, так как вторая часть задания (этот документ) заключается в расширении возможностей вашего калькулятора.
Другая цель задания состоит в приобретении опыта создания проектов в Xcode и набора кода с нуля. Не используйте copy / paste любого кода откуда угодно. Печатайте и смотрите, что Xcode делает, когда вы что-то делаете.

Обязательно посмотрите ниже раздел подсказок (**Hints** section)!

Материалы

- Прежде чем вы начнете выполнять задание вам необходимо скачать и установить Xcode 6 с App Store на вашем Mac.

Обязательные пункты задания

1. Получите калькулятор, работающий как продемонстрировано на Лекции 1 и 2. Часть Autolayout (Разметка), приведенная в конце лекции - это дополнительное и необязательное задание, но дает вам возможность приобрести хороший опыт использования Autolayout. Если вы не будете делать Autolayout, то убедитесь, что вы позиционировали все так, что интерфейс был виден на любой модели iPhone (то есть располагался в левом верхнем уголе экрана).

2. Ваш калькулятор (calculator) уже умеет работать с числами с плавающей точкой (например, если вы последовательно будете нажимать на клавиши 3 ⏎ 4 ÷ то он покажет правильный результат 0.75), тем не менее, пока нет ввода чисел с плавающей точкой. Это надо исправить. Разрешите вводить только правильные числа с плавающей точкой (напримр, “192.168.0.1” - неправильное число с плавающей точкой). Вам нужно добавить новую кнопку с точкой “.”. В этом задании не беспокойтесь о точности.

3. Добавьте следующие 4 кнопки с операциями:
   - sin : вычисляет sin операнда на вершине стэка.
   - cos : вычисляет cos операнда на вершине стэка.
   - sqrt : вычисляет √ (квадратный корень) операнда на вершине стэка.
   - π: вычисляет (ассоциируется) с величиной π. Например: 3 π × должно показать на **дисплее (display)** вашего калькулятора величину π, умноженную на 3, точно также 3 ⏎ π ×, и π 3 × должны показывать величину π, умноженную на 3.

4. Добавьте метку UILabel на ваш пользовательский интерфейс, которая показывает историю ввода пользователем каждого операнда и *операции*. Найдите подходящее место для этой текстовой метки.

5. Добавьте кнопку “C”, которая будет чистить все (ваш **дисплей ввода**, новую метку, которую вы только что добавили). При нажатии кнопки “C”. калькулятор должен находится в таком же состоянии как при старте”.

6 . Избегайте проблем, перечисленных в разделе **Оценка (Evaluation)**. Этот список может расти от задания к заданию, так что всегда проверяйте его.

Подсказки (Hints)

Эти подсказки не являются обязательными пунктами задания. Вы можете их не выполнять. Однако, следование этим подсказкам , возможно, немного облегчит выполнение задания (assignment) (хотя никаких гарантий!).

1. Существует `String` метод **`rangeOfString(substring: String )`**, который может оказаться полезным для выполнения той части задания, которая касается плавающей точки. Он возвращает `Optional`. Если переданный `String` аргумент не может быть найдет в получателе, тогда возвращает **nil** (В противном случае не беспокойтесь о том, что он возвращает в этом задании).

2. Требование работать с плавающей точкой, возможно, может потребовать единственной строки кода. Заметьте, что то, что вы читаете в данный момент - это подсказка, а не обязательное задание.

3. Будьте внимательны когда пользователь начинает ввод числа с нажатия десятичной точки, например, он хочет ввести в калькулятор число ".5". Возможно, что ваше решение просто работает, но все-таки протестируйте его.

4. sin() и cos() - это стандартные функции, которые доступны в Swift для вычисления sin и cos.

5. Значение π доступно с помощью выражения M_PI: `let x = M_PI.`

6. Вы можете думать о π как об операнде или как об *операции* (такой новый тип операции, которая не имеет аргументов и не нуждается в операндах, расположенных в стэке, но возвращает значение). Это на ваше усмотрение. Но в любом случае было бы прекрасно, если бы вы добавили другие константы в ваш Calculator, написав минимальное количество строк кода.

6. Экономия очень ценна в кодировании: самый легкий путь избежать ошибок при кодировании - это вообще не писать ни одной строчки кода. Это задание требует очень, очень мало строк кода, так что если вы начинаете писать десятки строк кода, то вы явно движетесь не в том направлении.

Что нужно изучать

Это частичный список концепций, в которых это задание увеличивает ваш опыт работы или продемонстрирует ваши знания.

- Xcode
- Swift
- Target/Action
- Outlets
- UILabel
- UIViewController
- Classes
- Functions and Properties (instance variables)
- let versus var
- Optionals
- Computed или Stored properties
- String and Array
- Autolayout (extra credit)

Оценка (Evaluation)

Во всех заданиях требуется написание качественного кода, на основе которого строится приложение без ошибок и предупреждений (without warnings or errors), следовательно вы должны тестировать полученное приложение (application) до тех пор, пока оно не начнет функционировать правильно согласно поставленной задачи.

Приведем наиболее общие соображения, по которым задание может быть отклонено (marked down):

- Приложение не создается (Project does not build).
- Приложение не создается без предупреждений (Project does not build without warnings).
- Один или более пунктов в разделе **Обязательные задания (Required Tasks)** не выполнены.
- Не понята фундаментальная концепция задания (A fundamental concept was not understood).
- Код - небрежный или тяжелый для чтения (например, нет отступов и т.д.).
- Ваше решение тяжело (или невозможно) прочитать и понять из-за отсутствия комментариев, из-за плохого наименования методов и переменных, из-за непонятной структуры и т.д.

Часто студенты спрашивают: “Сколько комментариев кода нужно сделать?” Ответ - ваш код должен легко и полностью быть понятным любому, кто его читает. Вы можете предполагать, что читатель знает SDK, но вам не следует предполагать, что он уже знает решение проблемы.

Дополнительные задания (Extra Credit)

Мы постарались создать Дополнительные задания так, чтобы они расширили ваши познания в том, что мы проходили на этой неделе. Попытка выполнения по крайней мере некоторых из них приветствуется в плане получения максимального эффекта от этого курса.

1. Дайте возможность пользователю нажимать кнопку "backspace" если он ввел неверную цифру. Это вовсе не кнопка "undo," так что если пользователь нажал неверную кнопку с операцией (wrong operation button), то его ждет неудача (out of luck)! Вам решать как вы будете разруливать случай, когда пользователь кнопкой "backspace" полностью удалил число и все еще находится в процессе ввода числа (in the middle of entering), но оставлять **дисплей (display)** абсолютно пустым было бы не очень дружелюбно (not very user-friendly).
   Возможно вам покажется полезным использование глобальных функций **countElements** и **dropLast** для этой задачи. Обе могут брать **String** как их единственный аргумент. Первая из этих функций - это то, о чем вы можете думать как о "length" (длине), а другая удаляет последний символ из **String** (и возвращает результат этого действия). Возможно вы будете озадачены, если alt-кликните на этих функциях. Типы аргументов и возвращаемые значения потребуют некоторых значительных объяснений, которые мы не можем здесь привести, но, если кратко, то **String** - это в действительности коллекция символов (a collection of characters), которая может быть проиндексирована и "нарезана" (sliced) на подколлекции символов (sub-collections of characters). Вот почему **countElements** (которая берет коллекцию (collection)) и **dropLast** (которая берет "нарезанную" (sliceable) вещь) работают со строками **String**.

2. Когда пользователь нажимает кнопку операции (operation button), разместите знак равно = в конце текстовой метки **UILabel**, которая добавляется в Обязательном задании (#4). Таким образом пользователь сможет понять, является ли число на **дисплее калькулятора** результатом вычислений или это число только что ввел пользователь. Не позволяйте знаку "=" многократно появляться в конце метки **UILabel**.

3. Добавьте +/- операцию, которая меняет знак числа на **дисплее (in the display)**. Будьте более внимательны с этим. Если пользователь находится в середине ввода числа, вы возможно, захотите изменить знак этого числа и позволить ему

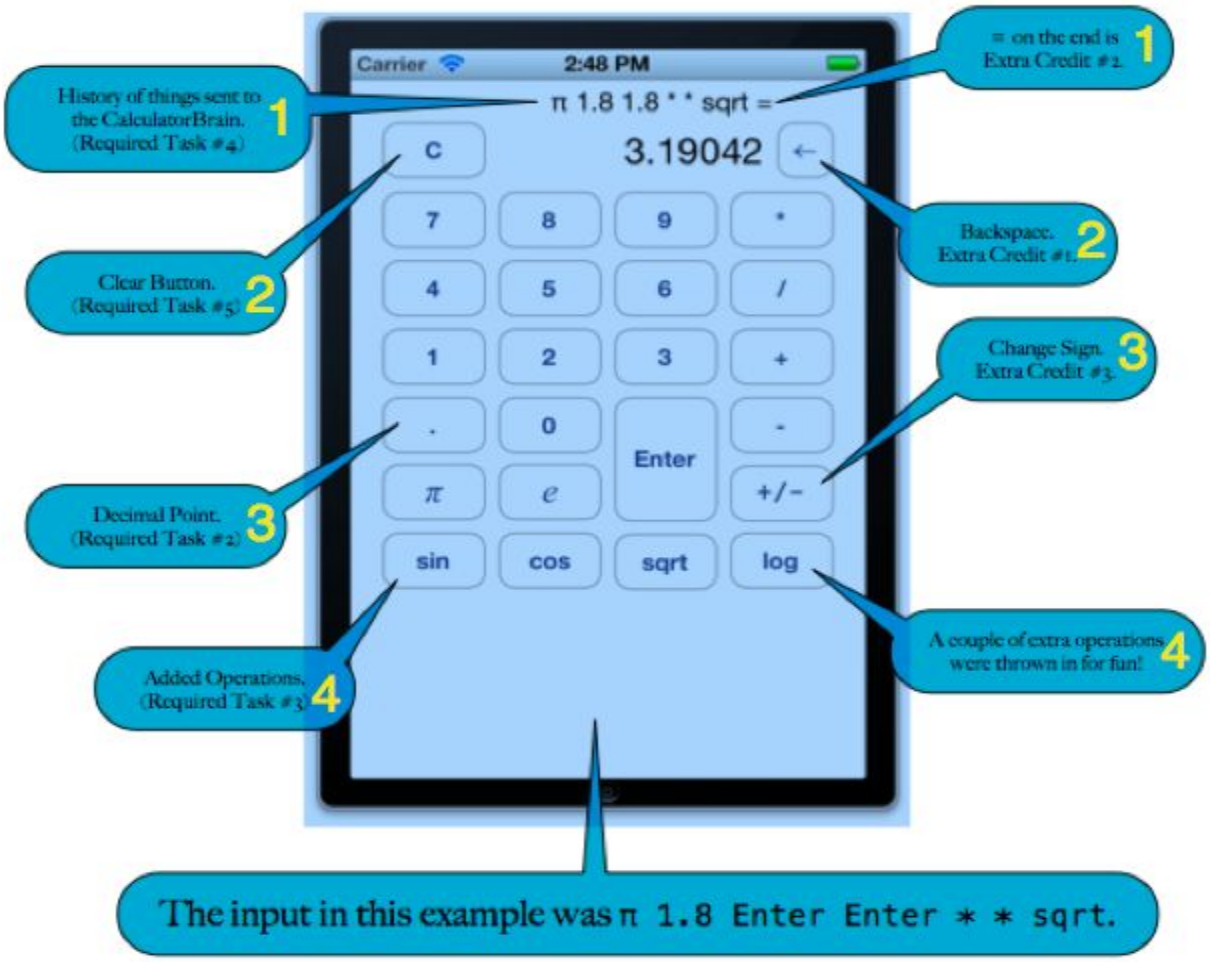
продолжать его ввод, не прибегайте к “принудительному” “enter” как в других операциях. Но если вы не находитесь в середине ввода числа, то это будет работать просто как любая другая унарная операция (например, sqrt).

4. Измените вычисляемую переменную экземпляра класса displayValue так, чтобы она была Optional Double, а не Double. Его значение должно быть nil если содержимое display.text не может быть интерпретировано как Double (вам нужно использовать документацию для понимания кода NSNumberFormatter). Установка этого значения в nil должна очищать display.

5. Используйте Autolayout, чтобы заставить ваш калькулятор выглядеть хорошо на любой модели iPhone и как портретной, так и ландшафтной ориентации (пока не беспокойтесь об iPads). Просто используйте CTRL -перетягивание к концам вашего экрана для позиционирования display, вы можете делать CTRL -перетягивание между вашими UILabels (и/или вашей кнопкой C) для установки вертикальных /горизонтальных расстояний между ними. Используйте голубые направляющие линии! Возможно, хорошая идея переустановить вашу Autolayout (с помощью кнопки в нижнем правом углу), а затем использовать CTRL-перетягивание для добавления правил (constraints) к элементам, которые не являются частью сетки, состоящей из цифровой клавиатуры и кнопок с операциями, а затем использовать для настройки кнопки в самом правом положении внизу чтобы передвинуть их (после того как вы передвинули их в нужное место по отношению в вашим UILabel(s), и т.д., используйте пунктирные голубые линии!).

Примерный вид экрана

Этот экран представлен здесь только для примера. Он не находится в разделе “Обязательнын задачи”. Фактически он включен сюда по требованию прошлых студентов. Однако не позволяйте виду экрана, представленного ниже, сдерживать вашу креативность!



Ввод в этом примере имеет вид: π 1.8 Enter Enter * * sqrt.

Надписи слева (сверху вниз):

1. история элементов, посланных в Calculator (Обязательное задание № 4);
2. кнопка “Очистить”(Clear)(Обязательное задание № 5);
3. десятичная точка (Обязательное задание № 2);
4. добавленные операции (Обязательное задание № 3);

Надписи справа(сверху вниз):

5. = в конце - это дополнительное задание № 2;
6. кнопка BackSpace - дополнительное задание № 1;
7. смена знака - дополнительное задание № 3;

8. добавлена пара дополнительных операций.