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 воспроизведения демонстрационного примера 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 7 с App Store на вашем Mac (Xcode 6 НЕ будет работать)..
- Ссылку на видео лекций вы можете найти там, где вы нашли этот документ, то есть на iTunes U.

Обязательные пункты задания

1. Получите калькулятор, работающий как продемонстрировано на Лекции 1 и 2.

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 возможности ввести число с плавающей точкой. Это надо исправить. Разрешите вводить только правильные числа с плавающей точкой (например, “192.168.0.1” - неправильное число с плавающей точкой). Вам нужно добавить новую кнопку с точкой “.”. В этом задании не беспокойтесь о точности и значащих цифрах (включая нижеприведенные пункты задания).

3. Добавьте побольше кнопок с операциями к вашему калькулятору, так чтобы общее их число равнялось, по крайней мере, 12 (у вас может быть и больше, если вы этого хотите). Вы можете выбрать любые подходящие вам операции. Кнопки должны быть правильно и красиво организованы как в портретном режиме, так и в ландшафтном режиме на любых моделях iPhones.

4. Используйте цвета для придания вашему пользовательскому интерфейсу (UI) привлекательности. По крайней мере, цвета ващих кнопок с операциями должны отличаться от цвета кнопок цифровой клавиатуры, но в любом случае вы можете использовать любые цвета для придания хорошего вида вашему калькулятору.

5. Добавьте **String** свойство к вашему CalculatorBrain с именем description, которое возвращает описание последовательности введенных операторов и операций, которые приводят к значению, возвращаемому result. Ни символ “=“, ни символ “...” не должны появляться в этом описании.

6. Добавьте **Bool** свойство к вашему CalculatorBrain с именем isPartialResult, которое возвращает значение, является ли бинарная операция отложенной (если да, возвращает true, если нет, то false).

7. Используйте два вышеуказанных свойства для реализации метки UILabel на вашем пользовательском интерфейсе (UI), которая показывает последовательность операндов и операций, которые привели к тому, что показывается на дисплее display. Если isPartialResult, добавьте `...` в конец UILabel, иначе добавьте =. Если userIsInTheMiddleOfTypingANumber, вы можете оставить UILabel, показывающую то, что было там перед тем, как пользователь начал набирать число. Например ...

a. касаемся 7 + будет показано “7 + ...” (с 7, которая все еще на display)
b. 7 + 9 будет показано “7 + ...” (9 на display)
c. 7 + 9 = будет показано “7 + 9 =” (16 на display)
d. 7 + 9 = √ будет показано “√(7 + 9) =” (4 на display)
e. 7 + 9 √ будет показано “7 + √(9) ...” (3 на display)
f. 7 + 9 √ = будет показано “7 + √(9) =“ (10 на display)
g. 7 + 9 = + 6 + 3 = будет показано “7 + 9 + 6 + 3 =” (25 на display)
h. 7 + 9 = √ 6 + 3 = будет показано “6 + 3 =” (9 на display)
i. 5 + 6 = 7 3 будет показано “5 + 6 =” (73 на display)
j. 7 + = будет показано “7 + 7 =” (14 на display)
k. 4 × π = будет показано “4 × π =“ (12.5663706143592 на display)
l. 4 + 5 × 3 = будет показано “4 + 5 × 3 =” (27 на display)
m. 4 + 5 × 3 = будет показано “(4 + 5) × 3 =” если вы предпочитаете(27 на display)

8. Добавьте кнопку C, которая очищает все (ваш дисплей display, новую только ято добавленную метку UILabel и т.д.). Калькулятор после нажатия кнопки C должен быть в том же состоянии, что и при старте приложения.

Подсказки (Hints)

1. Существует `String` метод rangeOfString(`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 и так работает, но все-таки протестируйте его.

4. 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.

5. Вы можете использовать любые Unicode символы в качестве математических символов ваших операций. Например, x^2 и x^{-1} прекрасные математические символы.

6. Если вы установите text метки UILabel в nil или “” (пустая строка), то метка изменит свой размер и ее высота будет равна 0 (соответственно сдвинув оставщийся UI). Это может прводить ваших пользователей в замешательство. Если вы хотите, чтобы ваша метка UILabel оставалась пустой, но при этом высота ее не уменьшалась до нуля, то просто установите text метки UILabel в “ ” (пробел).

7. Если вы размещаете два (или более) элементов в Stack View и вы хотите, чтобы один из них (или более) были насколько возможно меньше, а остальные элементы использовали бы оставшееся пространство, вы можете это сделать, устанавливая Content Hugging Priority (CHP) для каждого элемента в Stack View. Большее число для CHP означает “сделай это как можно меньше, но так, чтобы оно все подходило к его содержимому”. Меньшее число для CHP означает “располагая этот элемент, вы можете растянуть его на сколько хотите”. The CHP для заданных кнопки оио метки (или чего-то еще) можно установить в Инспекторе Размера (Dimensions Inspector) в Области Утилит (правая сторона) в Xcode. UIButtons имеют по умолчанию **CHP** равный **250**. UILabels имеют по умолчанию **CHP** равный **251**. Если у вас имеется кнопка и метка в том же самом Stack View, то по умолчанию кнопка будет огромной, а метка маленькой. Если вы хотите достичь противоположной ситуации (маленькая кнопка,большая метка), то вам нужно изменить CHP для кнопки с **250** до **252** (или больше, но так, чтобы это было больше, чем **251** для метки). Или вы можете изменить CHPдля метки до **249** (или ниже). Это подсказка, а не обязательное заданиek. Вполне возможно, что при выполнии домашнего задания вам вообще не придется прибегать к CHP. Если вас это как-то смущает, то вы можете это игнорировать.

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 Dictionary

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 до тех пор, пока оно не начнет функционировать правильно согласно поставленной задачи.

Приведем наиболее общие соображения, по которым задание может быть отклонено:

- 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 , то его ждет неудача ! 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. Может оказаться, что раздел Strings and Characters Руководства по Swift будет очень полезно для решения этой проблемы.

2. Измените вычислямую переменную displayValue экземпляра класса и сделайте ее Optional Double, а не Double. Ее значение будет nil, если содержимое display.text нельзя интепретировать как Double. Установка его значения в nil должно очищать display. Вам нужно модифицировать код для displayValue.

3. Изучите документацию класса NSNumberFormatter с тем, чтобы использовать его для форматирования display, который должен показывать **6** цифр после десятичной точки (вместо показа всех цифр, представляющих Double). Это уничтожит необходимость использования **Autoshrink** в display. Использование класса NSNumberFormatter поможет избавиться от лишних “.0” , подсоединяемых к целочисленным значениям (например, показываем “**4**”, в не “**4.0**” при извлечении квадратного корня из **16)**. Выможете это использовать и для описания description в классе CalculatorBrain.

4. Определите одну из ваших кнопок для операции, которая бы “генерировала бы случайное число между 0 и 1”. Кнопка для этой операции не является константой (так как меняет свое значение при каждом нажатии). Она не является и унарной операцией (так как ни с чем не оперирует).