Урок 1

Введение.

Вводное занятие. Знакомство с основами языка и основными типами данных. Xcode, Playground. Переменные, константы и коллекции данных. Дебаггер.

Введение в Swift

Причины появления Swift

- Objective C существует с 89-го года, обновился в 2006-м году и прошло очень много времени с момента обновления (10 лет).
- Повышение популярности платформы и упрощения процесса разработки.
- Повышение качества приложения.
- Корректная работа приложений очень сильно зависима от человеческого фактора (уровень квалификации разработчика).
- SWIFT упрощает разработку.

История Swift

2010 год начало разработки - это официальная версия от компании Apple.

2 июня 2014 года на конференции WWDC Swift был официально представлен вместе с бесплатным руководством по использованию языка объёмом в 500 страниц, доступным на сервисе «iBook Store».

Версия Swift 1.0 была выпущена 9 сентября 2014 года, вместе с «Gold Master» версией Xcode 6.0 для iOS.

8 июня 2015 года компания Apple объявила о выпуске новой версии Swift 2.0, которая получила более высокую производительность, новое API обработки ошибок, улучшения синтаксиса языка, а также функцию проверки доступности функций Swift для целевых ОС.

3 декабря 2015 года была выпущена бета-версия Swift 3.0 с поддержкой операционных систем OS X, iOS и Linux и лицензированная под открытой лицензий Apache 2.0.

Основные преимущества Swift перед Objective C

1. Мощные языковые преимущества. Развитые коллекции, генерики. Замыкания, кортежи - нововведения, которые очень сильно могут сократить код приложения.

2. Предельная строгая типизация. Swift требует точного соответствия типам. Самое строгое соответствие типа по сравнению с другими языками. Это помогает сократить количество ошибок. Пока все переменные не будут приведены к нужным типам, приложение не соберется. Это искусственная защита - создана для того, чтобы не дать возможности разработчику сделать ошибки и, как следствие, повысить качество своего кода.

3. Значительно более лаконичный синтаксис. Сокращение кода, не в ущерб читаемости. Уменьшение количества кода косвенным образов влияет на количество ошибок.

Перечень некоторых возможностей Swift

Возможность применять как ООП, так и функциональное программирование.

Замыкания (closures) - это самодостаточные блоки с определенным функционалом, которые могут быть переданы и использованы в вашем коде. Замыкания в Swift похожи на блоки в C и Objective-C и лямбды в других языках программирования.

Кортежи (tuples) - группируют несколько значений в одно составное значение. Значения внутри кортежа могут быть любого типа, то есть нет необходимости, чтобы они были одного и того же типа.

Генерики (generics) - общий тип, универсальный тип.

Развитые перечисления (enums) - перечисления определяют общий тип для группы связанных значений и позволяют работать с этими значениями в типобезопасном режиме в вашем коде.

Вычисляемые свойства - свойства, которые не хранят какого-либо значения, но вычисляют его при обращении. А также могут обработать установку нового значения.

Наблюдатели для свойств - willSet/didSet - блоки кода, выполняющиеся перед/после изменением свойства.

Протоколы - описание свойств и методов без реализации. Сам по себе протокол бесполезен, но он может быть имплементирован любым классом, структурой или перечислением, при этом тип, имплементирующий протокол, обязан реализовать все свойства и методы, описанные в нем. Кроме того, расширить протокол и наделить его реализацией по умолчанию. Таким образом появляется создавать нечто похоже на миксины.

Переопределение операторов - вы можете перезаписать существующие операторы или создать новые для любых типов данных.

Автоматический подсчет ссылок (ARC) - используется в swift для управления памятью вашего приложения. ARC - нечто среднее между ручным управлением памяти и сборщиком мусора. В отличии от CG, он почти не увеличивает потребление ресурсов вашим приложением, но он также не может полностью освободить разработчика от необходимости думать об управлении памятью. В большинстве случаев он просто работает, но, тем не менее, нужно всегда быть начеку и следить, чтобы в программе не появился цикл удержания, который приведет к утечки памяти. Циклы удержания могут быть очень коварны!

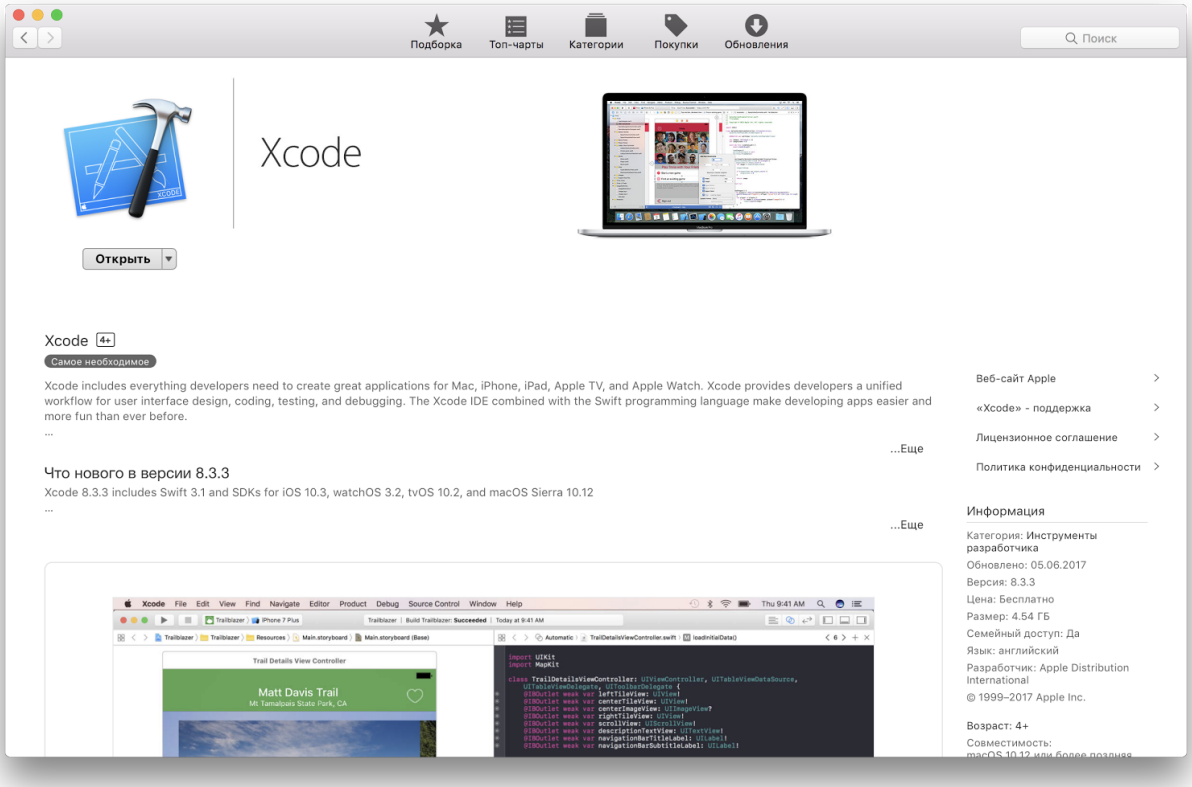
Знакомство со средой XCode

Установка Xcode

Прежде всего, нам понадобиться установить Xcode. Это Ide - среда для разработки приложений, предоставленная нам Apple.

Выполните пару простых шагов:

1. Откройте “App Store”.
2. Перейдите “Категории” -> “Инструменты разработчика”.
3. Выберите “Xcode”.
4. Нажмите загрузить.



Загрузка может занять значительное время, просто наберитесь терпения. Никогда не скачивайте xcode из неофициальных источников!

Аккаунт разработчика

Для того чтобы тестировать приложение на реальном устройстве, а не эмуляторе, необходимо зарегистрировать аккаунт разработчика. Это можно сделать по адресу developer.apple.com. При регистрации можно воспользоваться уже имеющимся appleID.

Для разработки приложения достаточно бесплатного аккаунта, но для массового тестирования и публикации в “App store” необходимо его оплатить. Стоимость 100$ в год.

Создание нового приложения в xcode

При запуске Xocde откроется стартовое окно. В правой его части будут отображаться последние открытые проекты, у вас оно может быть пустое. Слева отображаются варианты создания нового приложения:

1. “Get started with a playground” - создать новый файл “playground”. Этот файл предназначен для экспериментов с кодом. Он не связан с проектом, используйте его для изучения синтаксиса языка и исследования стандартных библиотек

2. “Create a new Xcode project” - создает проект с новым приложением.

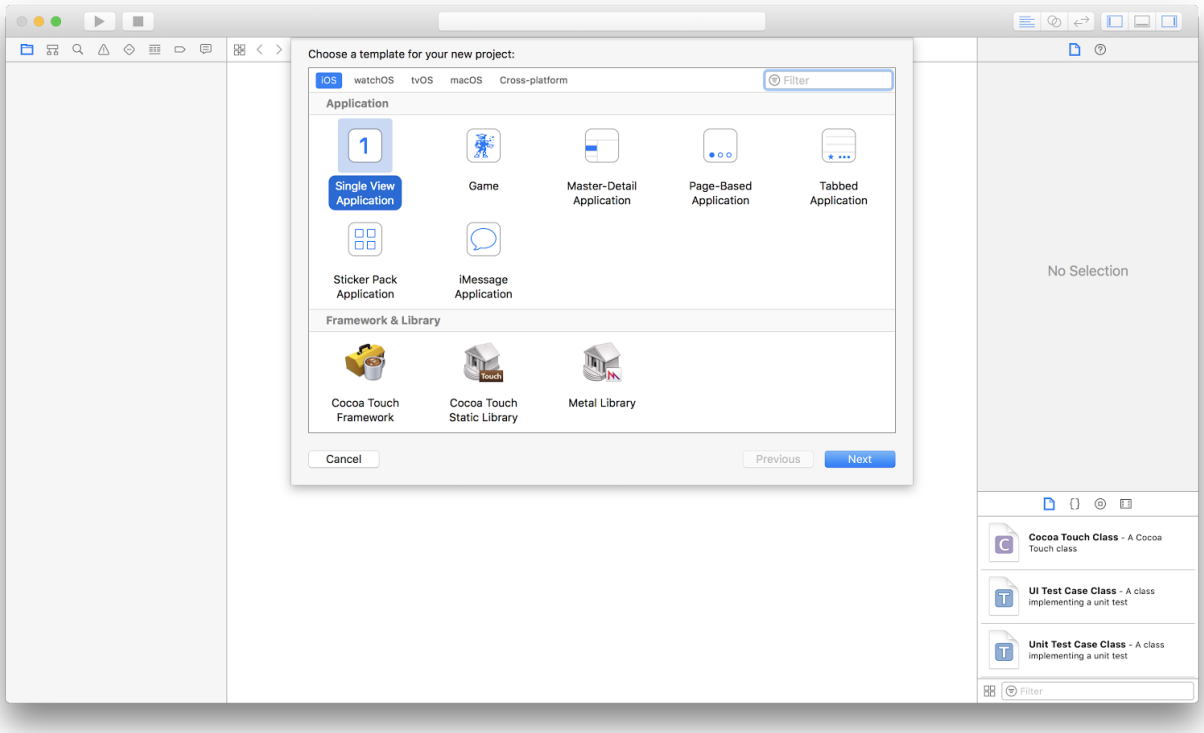
3. “Check out an existing project” - загружает уже созданный проект из системы контроля версий. Если вы не знаете, что такое “система контроля версий”, настоятельно рекомендую пройти бесплатный курс на портале geekbrains - “Git. Быстрый старт”

Выберете пункт - “Create a new Xcode project” и перейдите к процессу создания приложения. Первый шаг - это выбор шаблона. Шаблоны поделены на платформы, мы будем рассматривать только iOS. Каждый шаблон содержит один или пару начальных экранов вашего приложения.

Как правило, выбирается “Single View Application”, как самый простой, и меняется в зависимости от ваших нужд. Остальные шаблоны имеет смысл посмотреть в учебных целях, если у вас нет опыта разработки, то это хороший начальный пример.

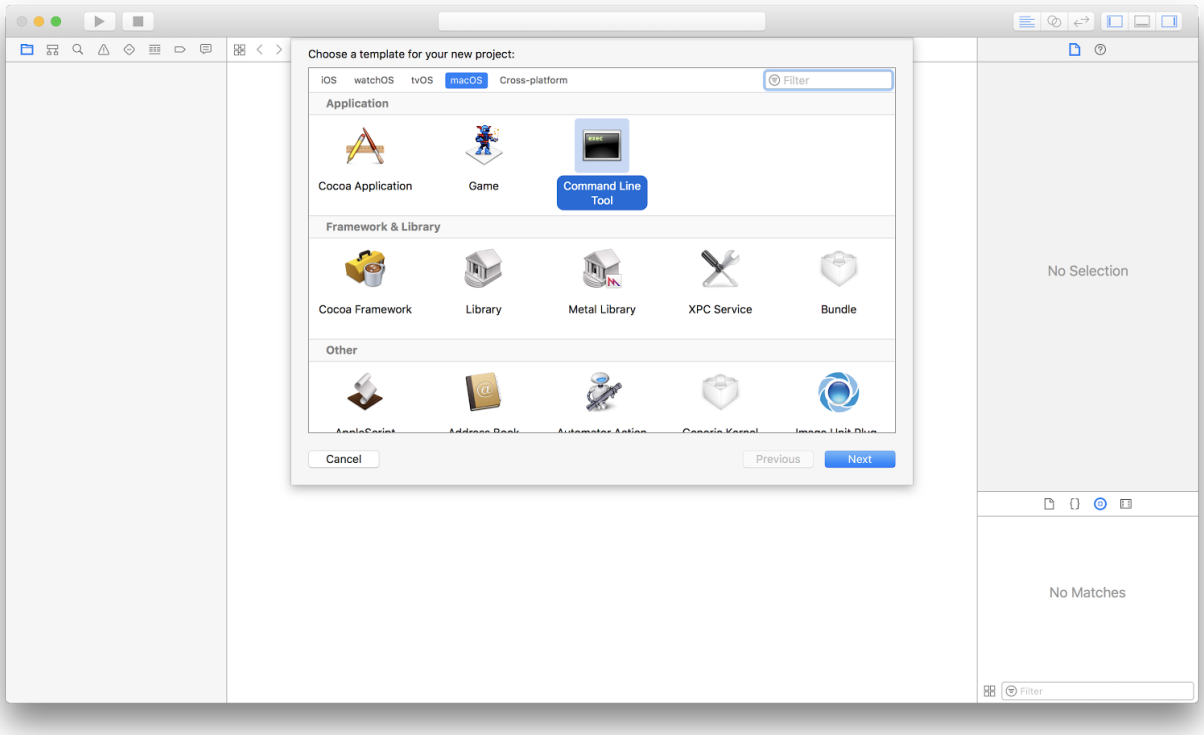
Краткое описание всех доступных шаблонов:

1. “Single View Application” - один пустой экран. Легко изменить под нужды конкретного приложения.
2. “Game” - самый особенный шаблон. Конфигурирует ваше приложения для разработки игры.
3. “Master-Detail Application” - несколько экранов с навигацией и поддержкой разделения экранов на две области на больших экранах.
4. “Page-Base Application” - пара экран с постраничным отображением информации как в приложении для чтения книг.
5. “Tabbed Application” - несколько экранов с навигацией основанной на вкладках.
6. “Sticker Pack Application” - набор стикеров для iMessage.
7. “iMessage Application” - приложение для iMessage.
8. “Cocoa Touch Framework” - используется, если необходимо создать библиотеку.

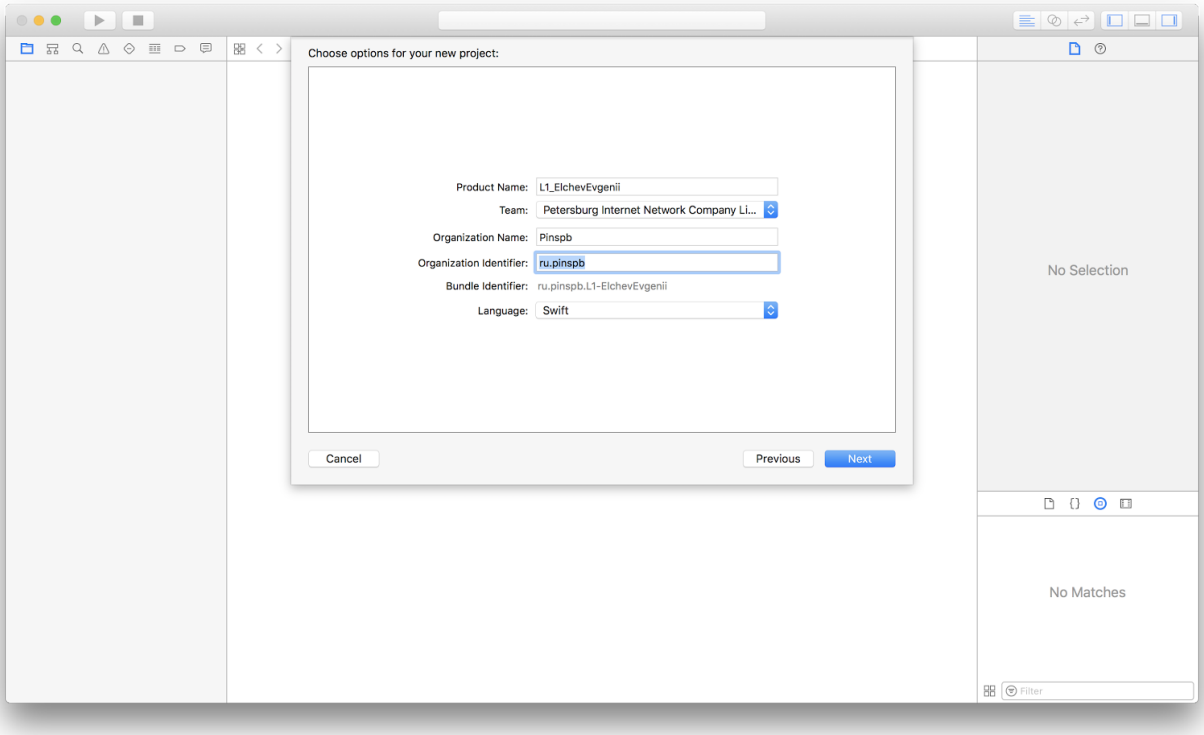


Создание первого приложения

Для наших уроков мы будем использовать “Command Line Tool” - шаблон. Он находится на вкладке “macOS”. Пусть вас не смущает такой выбор. Данный вид приложения поддерживает все возможности языка. Основное его отличия от iOS приложения состоит в том, что в нем нет графического интерфейса и для его запуска не требуется симулятор. В результате он запускается и выполняется очень быстро, что идеально подходит для наших задач.



На следующем шаге заполните все поля. В качестве имени приложения введите L1_ФИО. В качестве языка выберете swift. остальные поля можно заполнить по своему усмотрению.

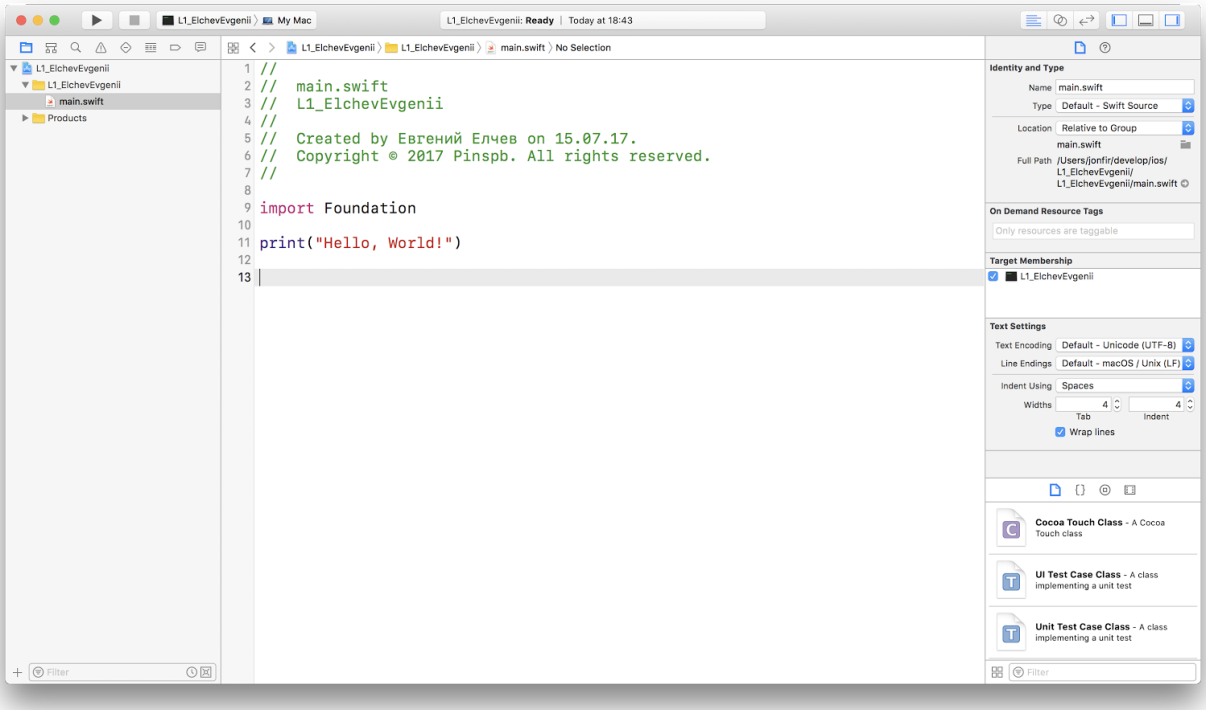


Следующий шаг - последний. Выберете папку, где будет сохранен ваш проект и нажмите “create”. Готово! Ваше первое приложение создано и готово к выполнению.

Знакомство с основными окнами среды

Xcode не просто редактор текста с подсветкой, это полноценная среда разработки (IDE). Вот ее ключевые возможности:

- управление вашим проектом;
- управление процессом сборки;
- управление процессом запуска;
- запуск и анализ тестов;
- работа с системами контроля версий. До версии 9 не самая ее сильная сторона и не рекомендована к использованию. Присмотритесь к бесплатному клиенту sourceTree;
- встроенный отладчик (Debugger);
- очень мощный механизм поиска;
- система рефакторинга, изменения кода на основе семантического анализа. Не поддерживает swift до версии xcode 9;
- встроенная документация по языку и стандартным фреймворкам
- менеджер ресурсов;
- редактор интерфейса;
- загрузка приложения в app store.



Давайте последовательно ознакомимся с основными элементами интерфейса нашей Ide.

В самом верху слева направо находятся:

1. Кнопка запуска проекта;
2. кнопка остановки проекта;
3. устройство на котором производится запуск;
4. строка состояния проекта
5. кнопка стандартного режима редактирования;
6. кнопка режима ассистента, переводит редактор в сдвоенный режим и отображает связанные с основный файлы;

7. кнопка режима версионирования, переводит редактор в сдвоенный режим и отображает изменения в произведенные со времени последнего коммита;
8. три кнопки включения/выключения отображения левой/правой/нижней панели.

Подсказка для тех, кто не знаком с основными принципами интерфейса macOS. Если при наведении на кнопку внизу вы видите черный треугольник, то попробуйте нажать на нее с зажатой клавишей “option”, вы увидите расширенный список действий этой кнопки. Поэкспериментируйте с элементами xcode!

Левая панель представляет набор различных навигаторов:

1. навигатор по файлам проекта;
2. навигатор по классам проекта;
3. навигатор поиска по проекту;
4. навигатор по ошибкам и предупреждениям;
5. навигатор по тестам;
6. навигатор отладки;
7. навигатор по точкам останова (breakpoint);
8. навигатор отчетов.

Не бойтесь переключаться между навигаторами во время разработки, это крайне полезные инструменты.

В центре находится редактор открытого файла. Его вид меняется в зависимости от типа файла. Он может отображать редактор интерфейса, текстовый редактор кода, форму изменения настроек проекта, таблицу редактирования plist файлов или редактор схемы coreData.

Правая панель также может меняться в зависимости от контекста. Но три вкладки там находятся постоянно:

1. вкладка свойств файла;
2. вкладка справки;
3. вкладка сниппетов находится внизу панели;

Если свойства файла вам практически никогда не понадобятся, то справкой пользуйтесь как можно чаще. Многие вопросы, которые у вас возникают, достаточно хорошо описаны инженерами apple.

Нижняя вкладка сдвоенная. Левая ее часть принадлежит окну дебаггера, там вы можете следить за значениями переменных. Она активна только во время пошаговой отладки. Правая часть окна консоли. Здесь отображаются ошибки, предупреждения возникающие во время выполнения программы и вывод вашей программы через методы “print” и “debugPrint”.

Не бойтесь исследовать вашу IDE. Можете создать тестовый проект и поэкспериментировать с различными вкладками и режимами. Очень важно овладеть инструментом, с помощью которого вам предстоить творить!

Синтаксис Swift, основные концепции

Константы и переменные

Переменная это именованная область в памяти, которую можно использовать для доступа к данным. Данные находящиеся в переменной называются значением переменной.

Константа это переменная значение которой нельзя изменять.

В swift используется два ключевых слова для объявления переменных. let - объявляет переменную как константу, это значит, что в дальнейшем ее значение нельзя будет изменить. var - объявляет обычную переменную, значение которой можно изменить. Компилятор строго следит за объявлением переменных и констант, если вы объявите переменную и не будете менять её значение, вам будет выведена соответствующая рекомендация о целесообразности переопределить её как константу. Константа в Swift фактически является константным указателем на константные данные - это касается как примитивные типов, так и объектных, и коллекций.

Основные типы данных

Swift имеет основные типы данных и также может использовать типы данных из языка Objective C.

Таблица основные типов данных:

Тип данных	**Описание**	**Диапазон**
Integer	Целочисленный тип	Диапазон соответствует разрядности ОС - Int32 или Int64
UInt	Целочисленный тип, только с положительными значениями.	Диапазон соответствует разрядности ОС - UInt32 или UInt64
Double	Представляет собой 64-битное число с плавающей точкой.	15 десятичных цифр
Float	Представляет собой 32-битное число с плавающей точкой.	6 десятичных цифр
Bool	Логический тип.	Может принимать значения true и false.
Character	Символьный тип.	Один символ
String	Строка	Любые символы

```swift
var a1: Double = Double.NaN
a1 = -5.323
let a2: Float = 4.23232
let a3: Int = Int.min
var a33: Int8 = Int8.max
a33 = Int8.min
var a34: Int16 = Int16.max
a34 = Int16.min
var a35: Int32 = Int32.max
a35 = Int32.min
var a36: Int64 = INT64_MAX
a36 = Int64.min
let a4: UInt = 12
let a41: UInt32 = 10
let aa110 = 10
let a6: Character = "b"
let a7: Bool = false
```

```
nan
-5.323
4.23232
-9223372036854775808
127
-128
32767
-32768
2147483647
-2147483648
9223372036854775807
-9223372036854775808
12
10
10
"b"
false
```

Коллекции

Коллекции это программный объект который содержит в себе набор значений. Так например если нам необходимо сохранить фио одного человека мы можем использовать переменную типа строка. Но если нам необходимо сохранить фио нескольких людей, например футбольной команды, мы можем использовать одну из коллекций называемую массив, типа строка.

```swift
let dreamTeam = [
    "Игорь Акинфеев",
    "Владимир Габулов",
    "Виктор Васин"
]
```

Таблица коллекций

Коллекция	Описание	Диапазон
Array<T>	Массив типа T	Ограничен памятью
Dictinary<K:T>	Словарь типа T с ключом K	Ограничен памятью
Set<T>	Множество типа T	Ограничен памятью

Все коллекции поддерживают операции вставки, удаления, поиска и сортировки элементов. Но каждая коллекция имеет свои отличительные черты и разное время выполнения этих операций. Будьте внимательны при выборе какой тип коллекции использовать в том или ином случае, вы можете как ускорить выполнение вашей программы, так и существенно его замедлить, если сделаете неверный выбор.

Настоятельно рекомендуется ознакомиться с понятием “о нотации” - способом описания относительного время выполнения операции в зависимости от числа элементов в коллекции, к которой применяется операции. Справочник основных вариантов сложности алгоритма:

- O(1) - операция выполняется быстро и время выполнения не зависит от размера коллекции;
- O(log n) - время выполнения зависит от размера коллекции, но не значительно;

- O (n) - линейная зависимость между размером коллекции и временем выполнения. Если размер коллекции увеличится в 2 раза, то и время выполнения увеличится в два раза. Не самая плохая зависимость, но при возможности ее следует избегать;
- O(n log n);
- O(n^2);
- O(2^n);
- O(n!).

Старайтесь выбирать коллекции так, чтобы операции, которые вы будете к ним применять, имели самую минимальную нотацию O(1) или хотя бы не превышали O(n).

Array

Массив - самая распространенная коллекция, она представлена почти во всех языках программирования. Это коллекция данных одного типа хранящихся в памяти друг за другом.

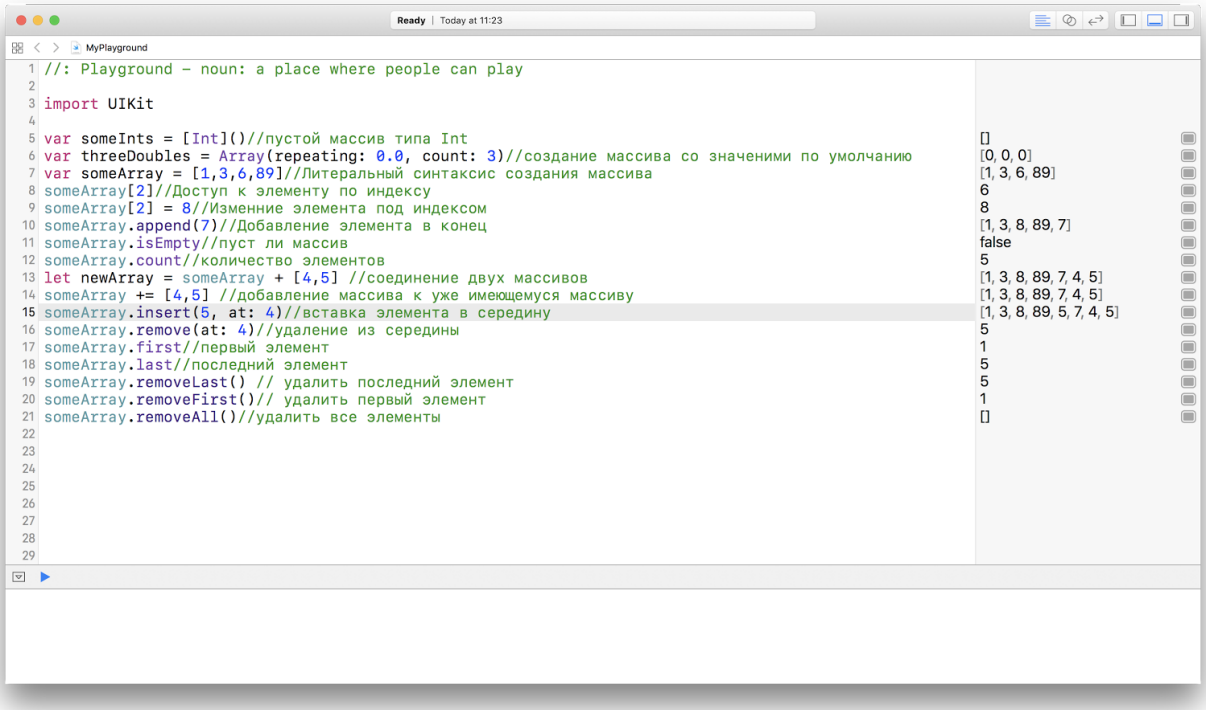
Массив всегда гарантирует сохранения порядка элементов. Это значит, что без вмешательства со стороны программиста элементы не могут менять свою позицию внутри коллекции, и первый элемент всегда будет первым, пока вы его сами не переместите.

Доступ к элементу коллекции осуществляется по индексу, смещению от начала, поэтому первый элемент в коллекции доступен по индексу 0 (без смещения), а второй 1 (смещение на один элемент).

Использовать эту коллекцию стоит в том случае, когда важно сохранять порядок элементов. Элементы коллекции можно отсортировать при необходимости по каким-либо параметрам.

Синтаксис	**Операция**	**Сложность**
array[0]	доступ по индексу	O(1)
array.index(of: T)	поиск элемента	O(n)
array.append(T)	вставка в конец	O(1)
array.insert(T, at: 0)	вставка в середину	O(n)
array.removeLast()	удаление последнего элемента	O(1)
array.remove(at: 0)	удаление в середине	O(n)

Кроме того swift поддерживает следующие методы:



Подумайте о том, какая сложность у каждой из этих операций.

Dictionary

Коллекция данных ключ значение, в учении о структурах данных часто носит название “хэш-таблица”. Коллекция поддерживает три основные операции: добавление новой пары, поиск пары по ключу и удаление пары из коллекции.

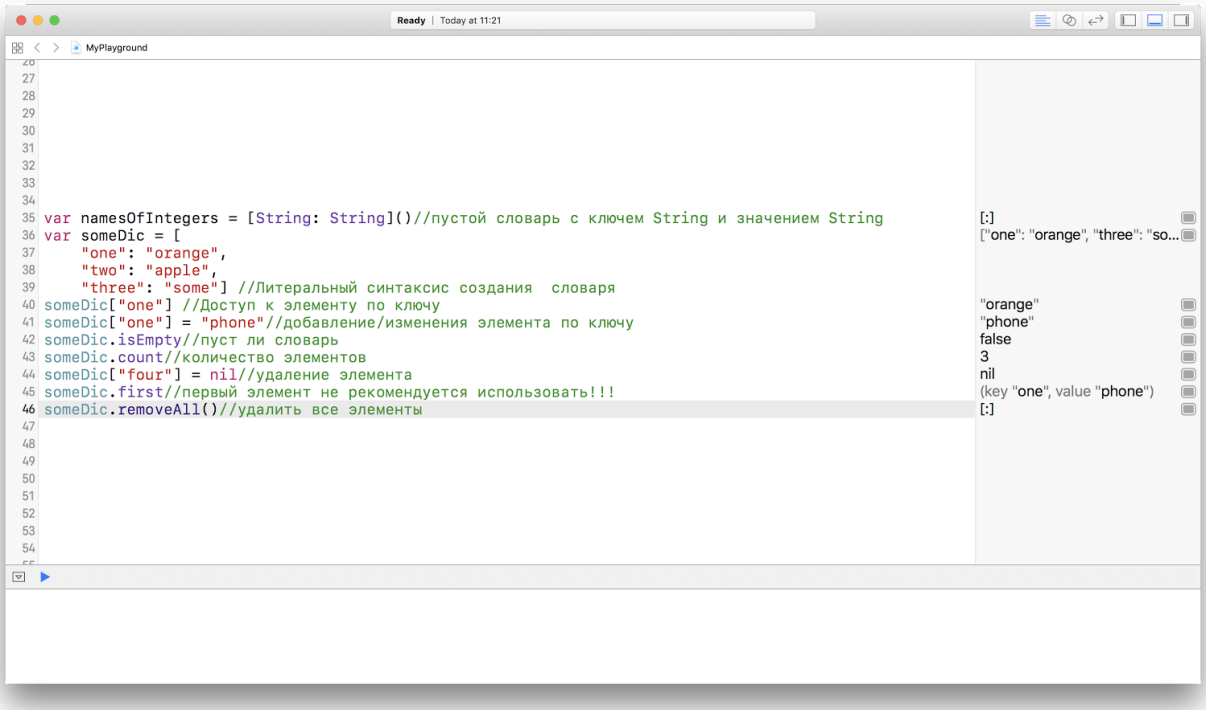
Коллекция не гарантирует сохранения порядка элементов. Это значит, что пары расположены внутри коллекции в неизвестном программисту порядке. Даже если вы создавали словарь, где на первом месте был ключ “one”, а на втором ключ “two”, то после инициализации эти ключи могут и, скорее всего, поменяют свои позиции. Это необходимо, чтобы оптимизировать время доступа к элементам.

Доступ к элементу коллекции осуществляется по ключу, и никогда не пытайтесь получить элемент по индексу, несмотря на то, что swift предоставляет подобный функционал.

Использовать эту коллекцию стоит в том случае, когда вам не приходится получить элемент коллекции по индексу, но часто приходится искать элемент по какому-либо определенному параметру, который можно использовать в качестве ключа.

Синтаксис	Операция	Сложность
...	доступ по индексу	не доступен
dictionary[K]	доступ по ключу	O(1)
dictionary[K] = T	вставка элемента	O(1)
dictionary[K] = nil	удаление элемента	O(1)

Кроме того, swift поддерживает следующие методы:



Set

Коллекция уникальных данных, т.е. элементы внутри коллекции не могут повторяться. Если вы попытаетесь добавить в коллекцию элемент, который уже содержится в ней, то добавления не произойдет, коллекция останется неизменной. Реализована в виде хэш-таблицы. Поддерживает математические операции с множествами.

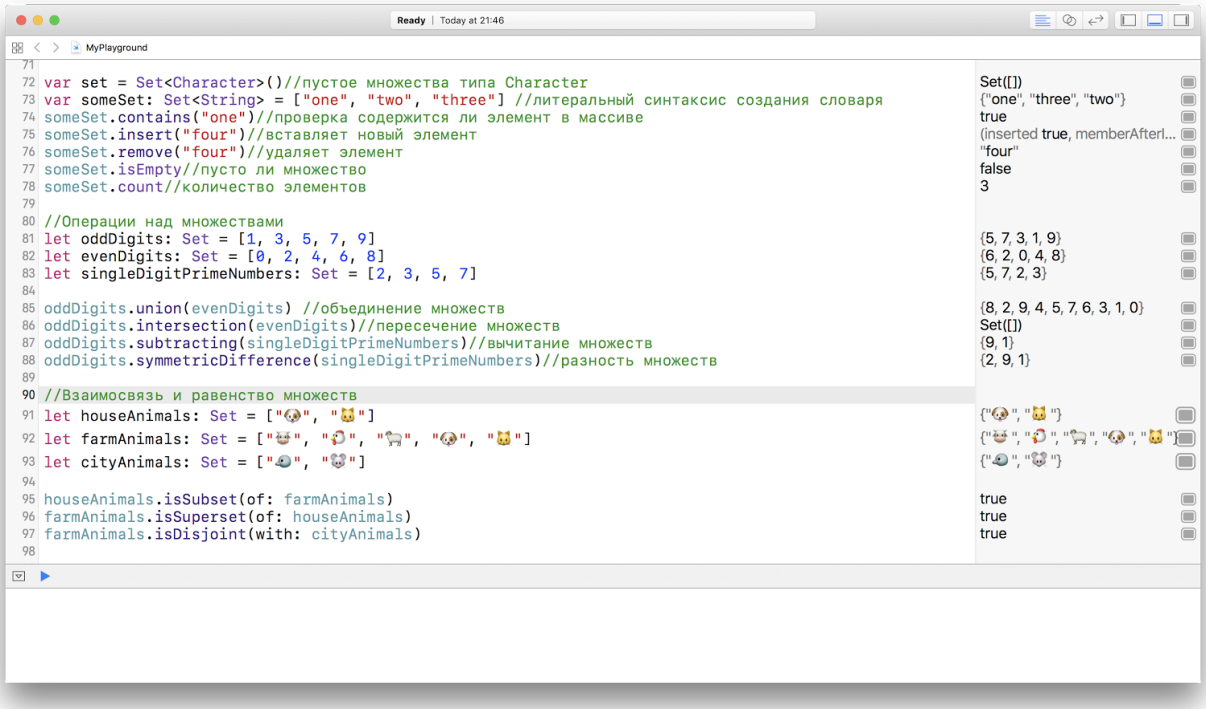
Коллекция не гарантирует сохранения порядка элементов.

Доступ к элементу коллекции осуществляется по самому элементу.

Использовать эту коллекцию стоит в том случае, когда вам не приходится получать элемент коллекции по индексу или ключу, но часто приходится проверять содержится ли элемент в коллекции и гарантировать уникальность элемента в коллекции. А также когда вам необходимо проводить операции со множествами над коллекциями.

Синтаксис	Операция	Сложность
...	доступ по индексу	не доступен
set.contains(T)	содержится ли элемент	O(1)
set.insert("four")	вставка элемента	O(1)
set.remove("four")	удаление элемента	O(1)

Кроме того, swift поддерживает следующие методы:



Преобразование типов

Предельно строгая типизация требует точного соответствия типам. Это означает что вы не можете сочетать в одном выражении значения разных типов, например сложить 4 и 4.0, хотя с точки зрения математики два этих значения равны, в нашем языке это целое число и число с плавающей точкой. В Swift самое строгое соответствие типам по сравнению с другими языками - это необходимо для сокращения количества ошибок в коде.

Пока все временные в одной операции не будут приведены к нужным типам - приложение не будет скомпилировано. Это означает, что складывать, вычитать и проводить прочие взаимодействия можно только с переменными одного типа. Это искусственная защита вашего приложения еще на этапе компиляции - как следствие, повышение качества кода и целевого приложения.

Swift может угадывать тип переменной при присвоении ей значения, но это не всегда бывает полезно, особенно в тех случаях, когда тип не очевиден.

// Преобразование типов	
var c1: Int = 25	25
var c2: Double = 10	10
var c3: Int = c1 + Int(c2)	35
var c4: Double = Double(c1) + c2	35
print(c4)	“35.0\n”
var c5: String = "c4=" + String(c4)	“c4=35.0”
var c6: Character = "\n"	“\n”
var c7: String = c5 + String(c6)	“c4=35.0\n”

Опциональный тип

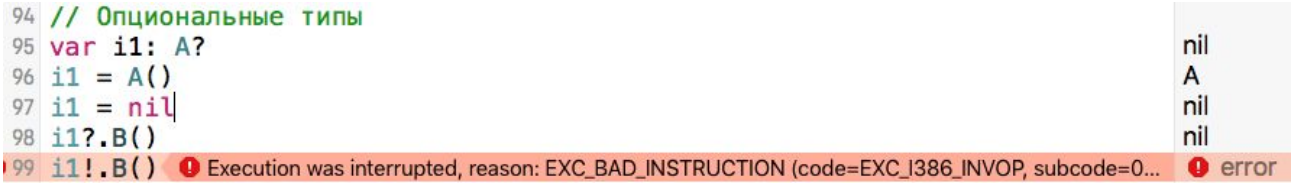
Любой тип данных в Swift может быть объявлен как опциональный. Это означает, что данная переменная может иметь значение своего типа, так и иметь значение nil. Опциональный тип

объявляется как обычный тип с добавлением знака вопроса в конце, например, опциональный строковый тип будет объявлен как String?

Опциональный тип является своего рода оберткой для переменной. Он гарантирует ей значение, как минимум, nil. Для работы с опциональной переменной необходимо сначала применить оператор принудительного извлечения, который обозначается знаком восклицания !

nil указывает на отсутствие каких-либо данных. Важно понимать, что даже пустая строка. без единого символа (“”), это все же строка, аналогично обстоит в ситуации с Int - 0 является значением, т.е. переменные содержащие “”, 0, 0.0 имеют значение, в отличии от переменных содержащих nil.

Важно помнить! При применении оператора принудительного извлечения опционального типа, если переменная будет иметь значение nil - приложение будет закрыто с ошибкой выполнения, т.е., попросту говоря, упадёт. Для безопасного извлечения опционального типа необходимо использовать опциональную привязку.



Опциональная привязка

Опциональная привязка очень похожа на принудительное извлечение, только лишь с той разницей, что при её использовании, если значение окажется равное nil, блок кода не будет выполнен и ошибок выполнения не будет.

Пример использования опциональной привязки:

var i44: Int? = nil
  if var value1 = i44 {
  print(value++)
}
	nil

Значение переменной равно nil - опциональная привязка не выполняется, при этом ошибок выполнения нет.

Когда мы меняем значение переменной на соответствующие ее типу опциональная привязка выполняется.

var i44: Int? = 6
  if var value1 = i44 {
  print(value1++)
}
	6
	
	“6\n”

Значение по умолчанию

При использовании опциональной переменной можно обойтись без извлечения, если предоставить значение по умолчанию через конструкцию ??

var a: Int? = nil let b = 4 + (a ?? 6)	10

var a: Int? = 10 let b = 4 + (a ?? 6)	14

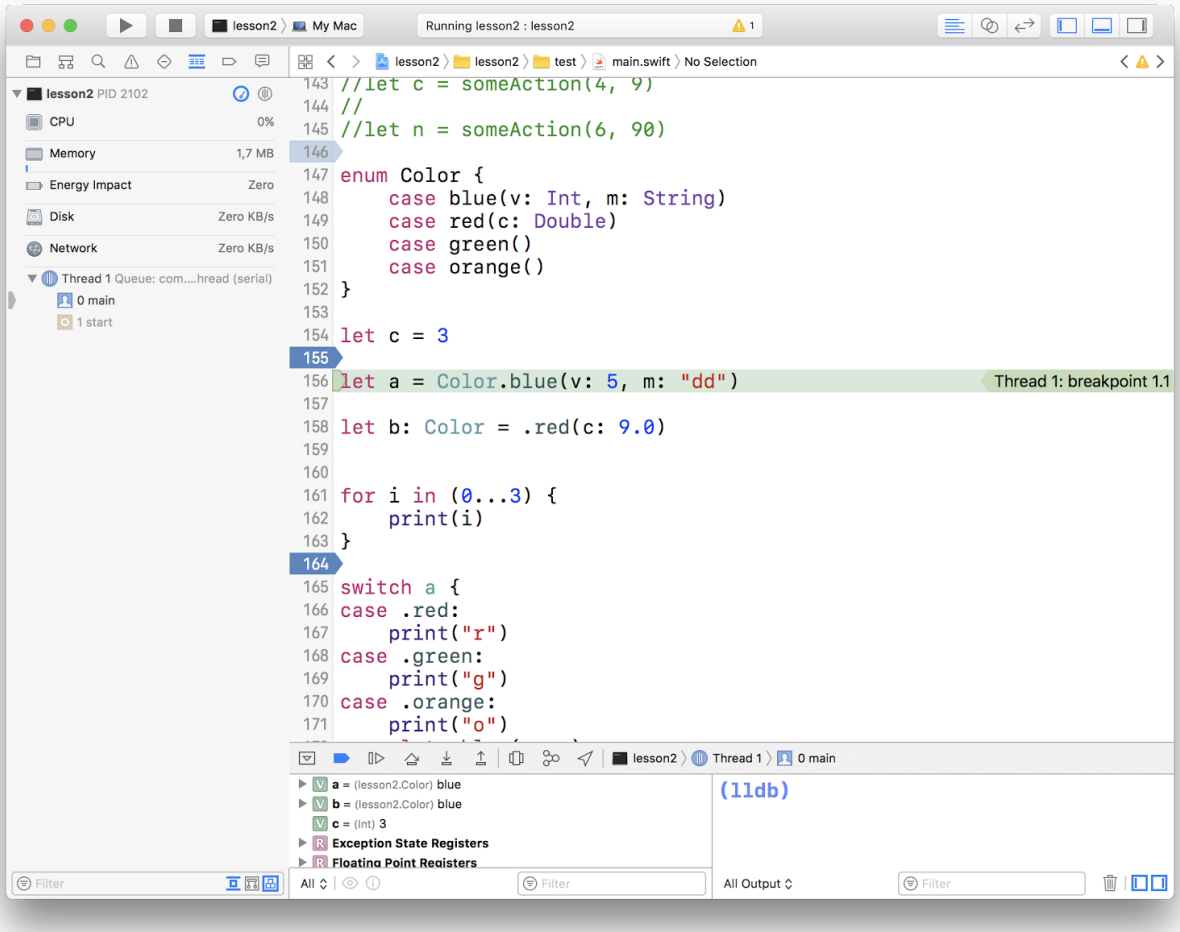
Отладчик (Debugger)

Как вы уже заметили, если вы допускаете критическую ошибку во время компиляции и выполнения, программа останавливается, и в консоли вы видите ошибку, что очень полезно, но не всегда понятно, что же привело к такой ошибке. Случаются даже более неприятные ситуации, программа работает, но делает не то, что вы задумывали, а вы никак не можете понять почему.

В этих ситуациях вам на помощь приходит режим отладки! В этом режиме ваша программа выполняется по шагам, строчка за строчкой. Вы можете проанализировать каждую команду в вашем коде, посмотреть результат ее выполнения и отслеживать изменение значений переменных.

Для того чтобы начать отладку, достаточно установить точку останова (breakpoint) на любой строке вашего кода. Делается это нажатием слева от строки кода. Вы можете установить несколько

breakpoint`ов, чтобы останавливать программу в различных местах. Повторное нажатие на breakpoint приведет к ее деактивации, и программа не будет останавливаться в этом месте.



После добавления breakpoint запустите программу, и, как только выполнение дойдет до строки, на которой она установлена, выполнение прекратится и активируется режим отладки. В этом режиме вы сами должны отдавать команды на выполнение следующего шага, без этого программа выполнятся не будет.

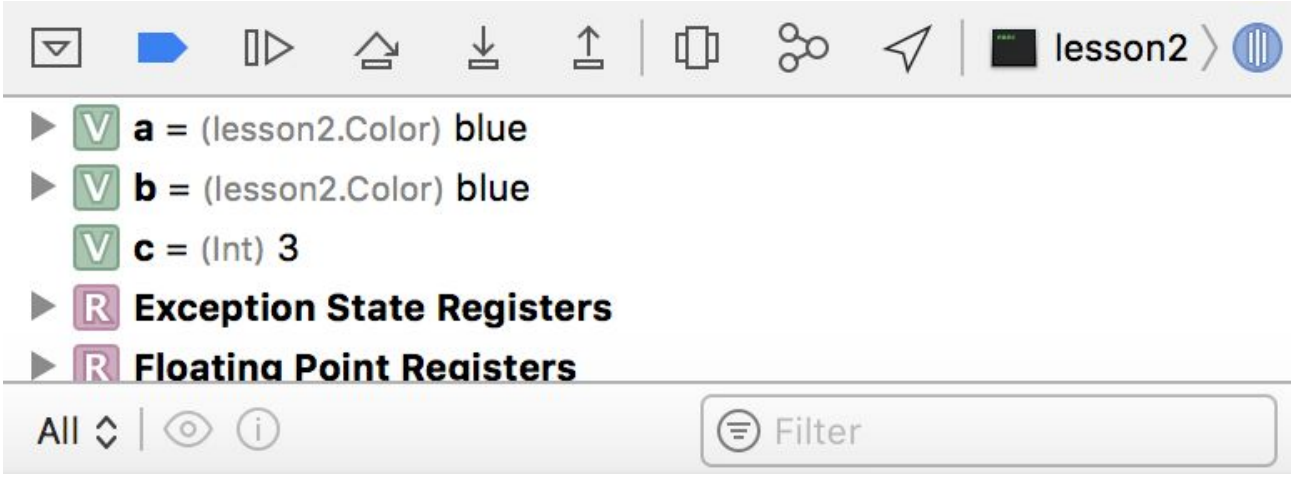
Строка, которая будет выполнена следующей, выделена зеленым. Это важно запомнить, так как вам может показаться, что подсвечивается уже выполненная строка.

Для управления выполнением у вас есть несколько кнопок, они расположены на нижней панели.

1. кнопка возврата к нормальному режиму. После ее нажатия процесс выполнения программы снова пойдет в нормальном режиме;
2. кнопка выполнения следующей команды без захода в функцию. При ее нажатии будет выполнена следующая строка кода, при этом, если вам на пути попадется вызов функции, ее выполнение будет принято за один шаг без раскрытия деталей;

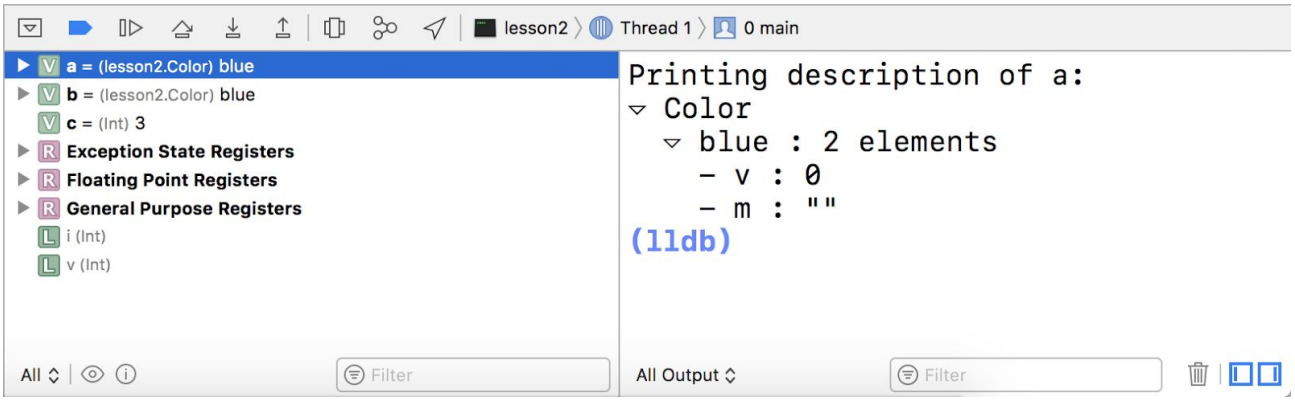
3. кнопка выполнения следующей команды с заходом в функцию. При ее нажатии будет выполнена следующая строка кода, при этом, если вам на пути попадется вызов функции, вы перейдете к ее описанию, сможете пошагово выполнить все ее инструкции;
4. кнопка выполнения следующей команды с выходом из функции. При ее нажатии, если вы уже находитесь внутри описания функции, вы немедленно покинете его и переместитесь к участку кода вызова этой функции.

Хотя сама по себе функция отслеживания порядка выполнения команд полезна, по настоящему незаменимой ее делает возможность наблюдать за состоянием переменных.



В панели дебаггера на нижней панели отображаются все переменные в текущей области видимости. Вы можете отслеживать их значения на текущем шаге выполнения. Если после выполнения следующего шага одна из переменных изменится, вы это увидите.

Если вам необходима более детальная информация о переменной, можно выбрать ее и нажать кнопку “Print description” (иконка с восклицательным знаком), слева в консоли вы получите ее описание.



Задачи для классной работы

1. Сложить две дроби между собой:

Решение:

```
304 var drob1 = 0.5
305 var drob2 = 1
306 var resulat1: Double = drob1 + Double(drob2)
```

```
0.5
1
1.5
```

2. Есть переменная типа Int и в ней записано шестизначное число. Задача поменять местами первую и последнюю цифры.

Решение:

```
var chislo: Int = 1234567                                          1234567
var tmpString = ""                                                 ""
let firstSimvol: Character = String(chislo).characters.first!      "1"
let lastSimvol: Character = String(chislo).characters.last!        "7"
tmpString = String(chislo)                                         "1234567"
tmpString.characters.removeLast()                                  "7"
tmpString.characters.removeFirst()                                 "1"
String(lastSimvol) + tmpString + String(firstSimvol)               "7234561"
```

Домашнее задание

Формат файла ДР: «1l_ФИ.playground»

1. Решить квадратное уравнение.
2. Даны катеты прямоугольного треугольника. Найти площадь, периметр и гипотенузу треугольника.
3. *Пользователь вводит сумму вклада в банк и годовой процент. Найти сумму вклада через 5 лет.

Дополнительные материалы

1. Официальный учебник по Swift
2. О нотация
3. Шпаргалка по сложности алгоритмов
4. Бесплатный курс по Git
5. Книга по системе контроля версий Git

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html#//apple_ref/doc/uid/TP40014097-CH5-ID309