Урок 1

Введение

Вводное занятие. Знакомство с языком. Основные типы данных. Xcode, Playground. Переменные, константы и коллекции данных. Дебаггер.

Введение в Swift

Причины появления Swift

- Язык Objective-C существует с 1989 года, последний раз обновлялся в 2006 году – более 10 лет назад.
- Рост популярности платформы и повышение качества приложения.
- Корректная работа приложений сильно зависит от человеческого фактора (квалификации разработчика).
- Swift упрощает разработку.

История Swift

Компания Apple начала разрабатывать Swift в 2010 году. 2 июня 2014 года этот язык был официально представлен на всемирной конференции разработчиков на платформах Apple (WWDC). Бесплатное 500-страничное руководство по его использованию было доступно на сервисе iBook Store.

Релиз Swift 1.0 состоялся 9 сентября 2014 года вместе с Gold Master-версией Xcode 6.0 для iOS.

8 июня 2015 года Apple выпустила новую версию Swift 2.0 с повышенной производительностью и новым API обработки ошибок. Улучшился синтаксис языка, появилась функция 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, лицензированная под открытой лицензий Apache 2.0. Новые версии Swift продолжают появляться и сейчас.

Основные преимущества Swift перед Objective-C

1. Мощные языковые преимущества. Развитые коллекции, генерики, замыкания, кортежи значительно сокращают код приложения.

2. Предельно строгая типизация по сравнению с другими языками. Пока все переменные не будут приведены к нужным типам, приложение не соберется. Эта искусственная защита позволяет снизить число ошибок разработчика и повысить качество кода.

3. Лаконичный синтаксис. Количество кода сокращается без ущерба читаемости. Это косвенно влияет и на количество ошибок.

Некоторые возможности Swift

Swift позволяет применять и объектно-ориентированное, и функциональное программирование.

Замыкания (closures) – это самодостаточные блоки с определенным функционалом, которые могут быть переданы и использованы в 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, не обязательно одного и того же.

Дженерики (generics) – общий, универсальный тип.

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

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

Наблюдатели для свойств willSet/didSet – блоки кода, выполняющиеся до/после изменения свойства.

Протокол – описание свойств и методов без реализации. Сам по себе протокол бесполезен, но он может быть имплементирован любым классом, структурой или перечислением. Тип, имплементирующий протокол, обязан реализовать все описанные в нем свойства и методы, расширить протокол и наделить его реализацией по умолчанию. Таким образом можно создавать нечто похожее на миксины.

Переопределение операторов – возможность перезаписать существующие операторы или создать новые для любых типов данных.

Автоматический подсчет ссылок (ARC) используется в Swift для управления памятью приложения. ARC – нечто среднее между ручным управлением памяти и сборщиком мусора (Garbage Collector). В отличие от GC, он почти не увеличивает потребление ресурсов приложением, но не может полностью освободить разработчика от необходимости управлять памятью. Нужно всегда следить, чтобы в программе не появился цикл удержания, который приведет к утечке памяти. Циклы удержания могут быть очень коварны!

Знакомство со средой Xcode

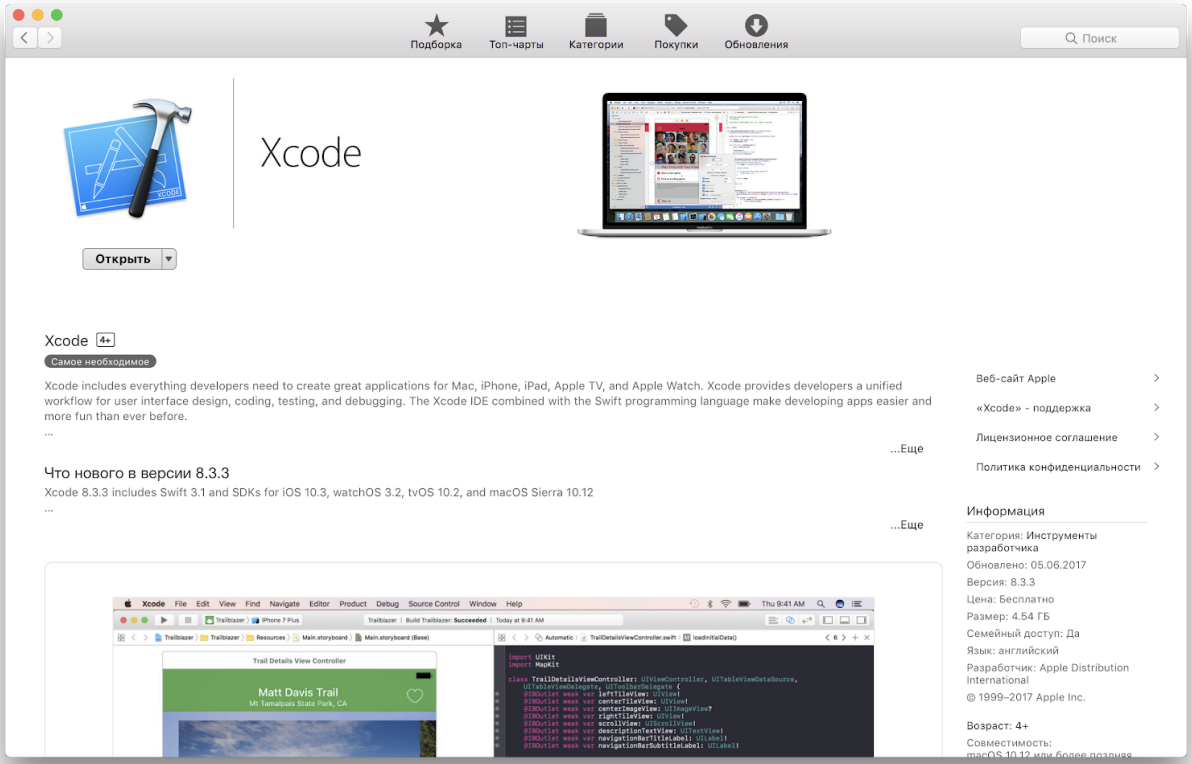
Установка Xcode

Прежде всего нам понадобится установить Xcode. Это IDE – интегрированная среда для разработки приложений, предоставленная Apple.

Выполните пару простых шагов:

1. Откройте App Store.
2. Перейдите в «Категории» → «Инструменты разработчика».
3. Выберите Xcode.
4. Нажмите «Загрузить».

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



Аккаунт разработчика

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 Apple ID.

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 Xcode откроется стартовое окно. В правой его части будут отображаться последние открытые проекты (у вас оно может быть пустым). Слева предлагаются варианты создания нового приложения:

1. Get started with a playground – создать новый файл «playground». Он предназначен для экспериментов с кодом и не связан с проектом. Используйте этот файл для изучения синтаксиса языка и исследования стандартных библиотек.

2. Create a new Xcode project – создать проект с новым приложением.

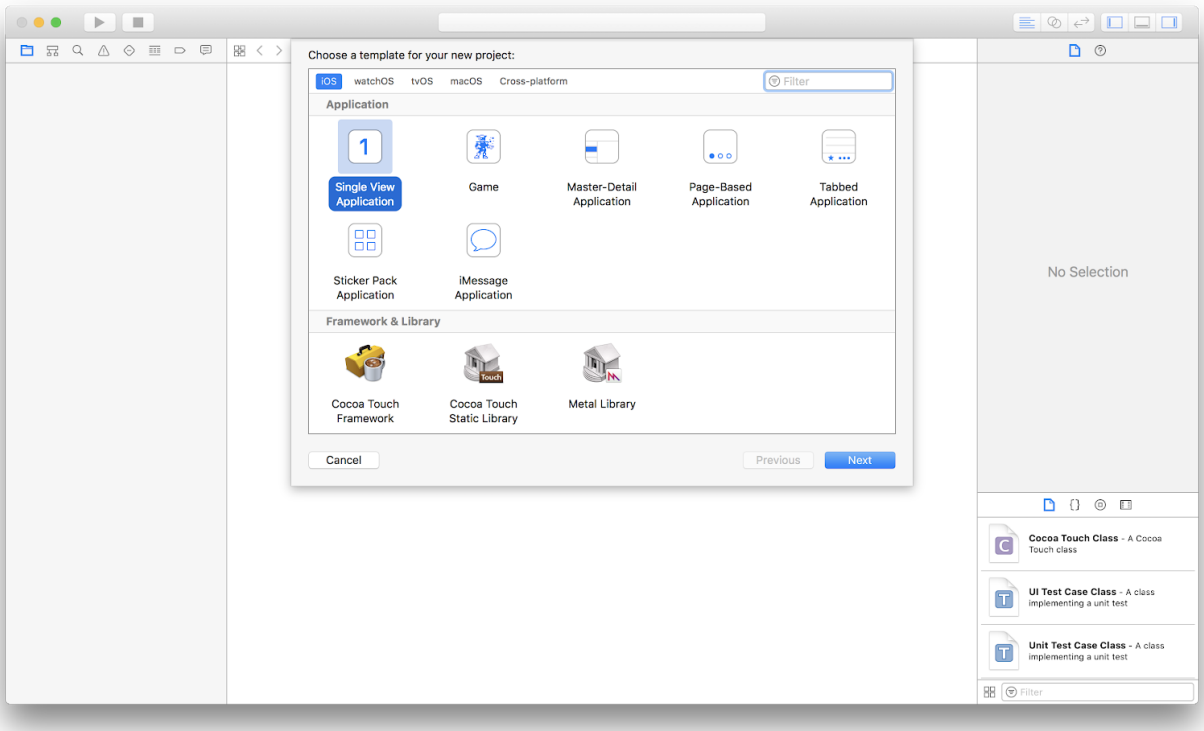
3. Check out an existing project – загрузить уже созданный проект из системы контроля версий. Если вы не знаете, что это такое, настоятельно рекомендую пройти бесплатный курс «Git. Быстрый старт» на портале GeekBrains.

Выберите 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, это хороший начальный пример.

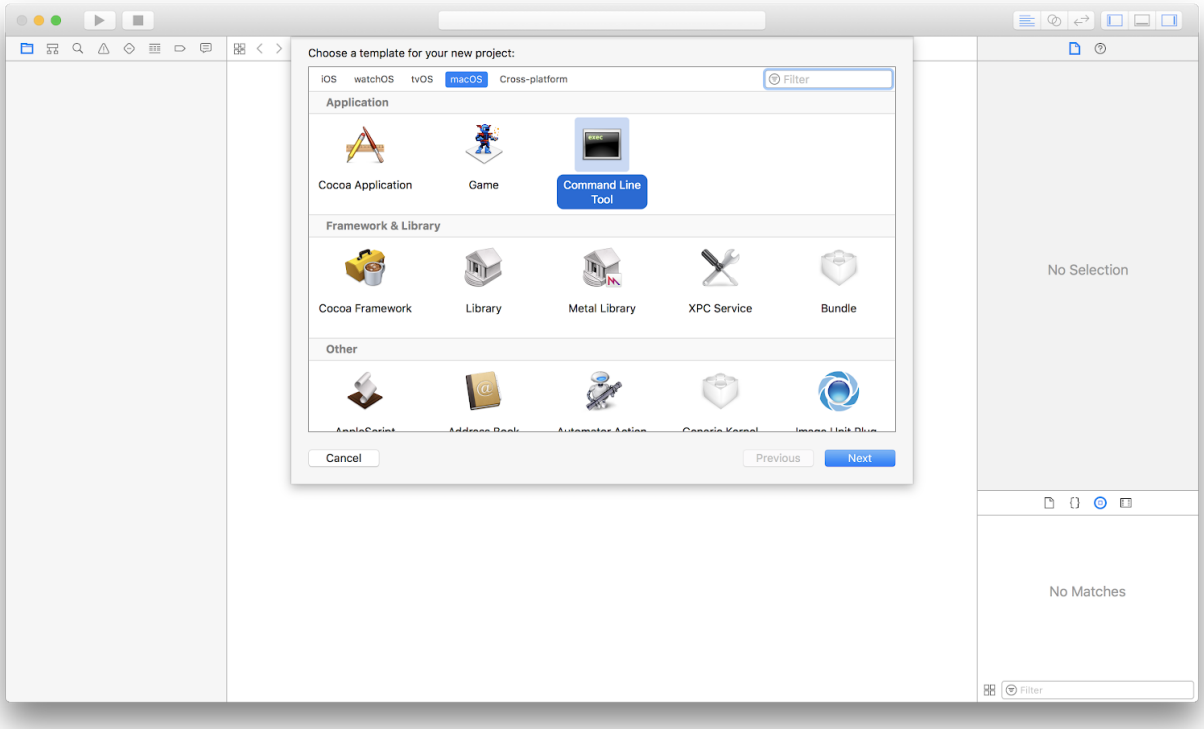
Краткое описание всех доступных шаблонов:

1. Single View Application – один пустой экран. Его легко изменить под нужды конкретного приложения.
2. Game – особенный шаблон, который конфигурирует ваше приложение для разработки игры.
3. Master-Detail Application – несколько экранов с навигацией и поддержкой разделения больших экранов на две области.
4. Page-Base Application – пара экранов 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 для создания библиотек.

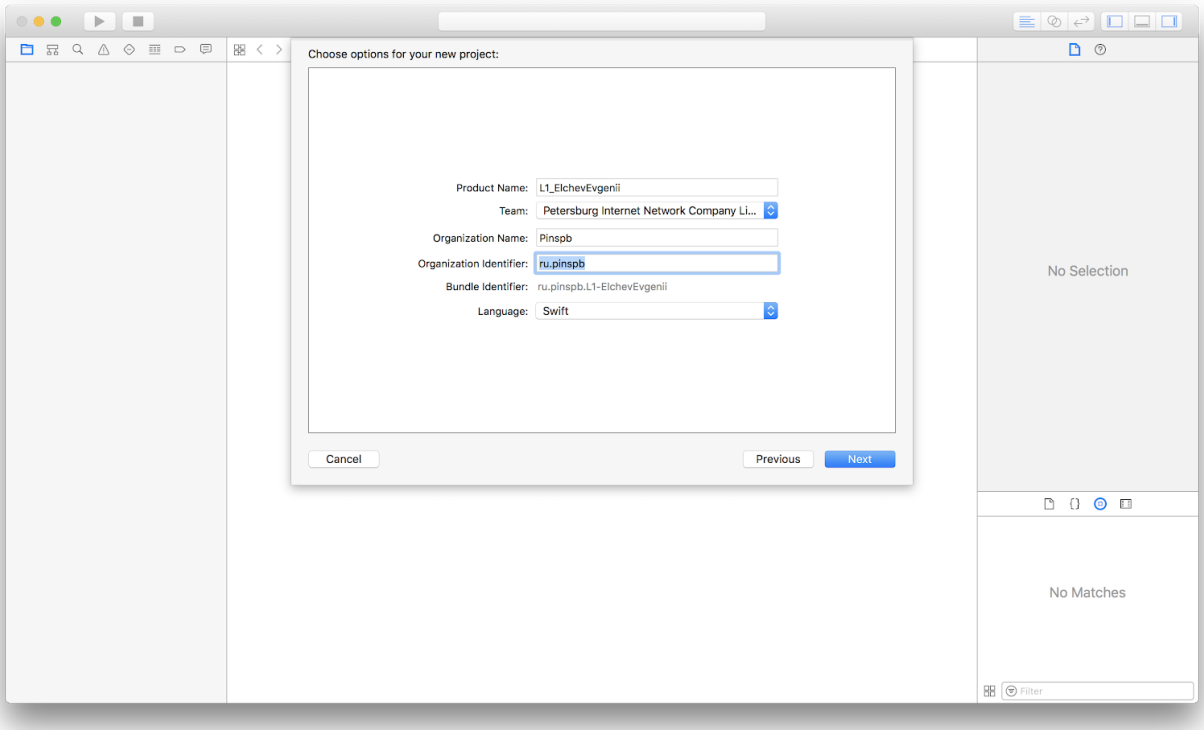


Создание первого приложения

Для уроков мы будем использовать шаблон Command Line Tool. Он находится на вкладке «macOS» и поддерживает все возможности языка. Основное его отличие от iOS-приложения 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 поле имени приложения введите L1_ФИО. В качестве языка выберите Swift. Остальные поля можно заполнить по своему усмотрению.



Последний шаг – выберите папку, где будет сохранен ваш проект, и нажмите Create.

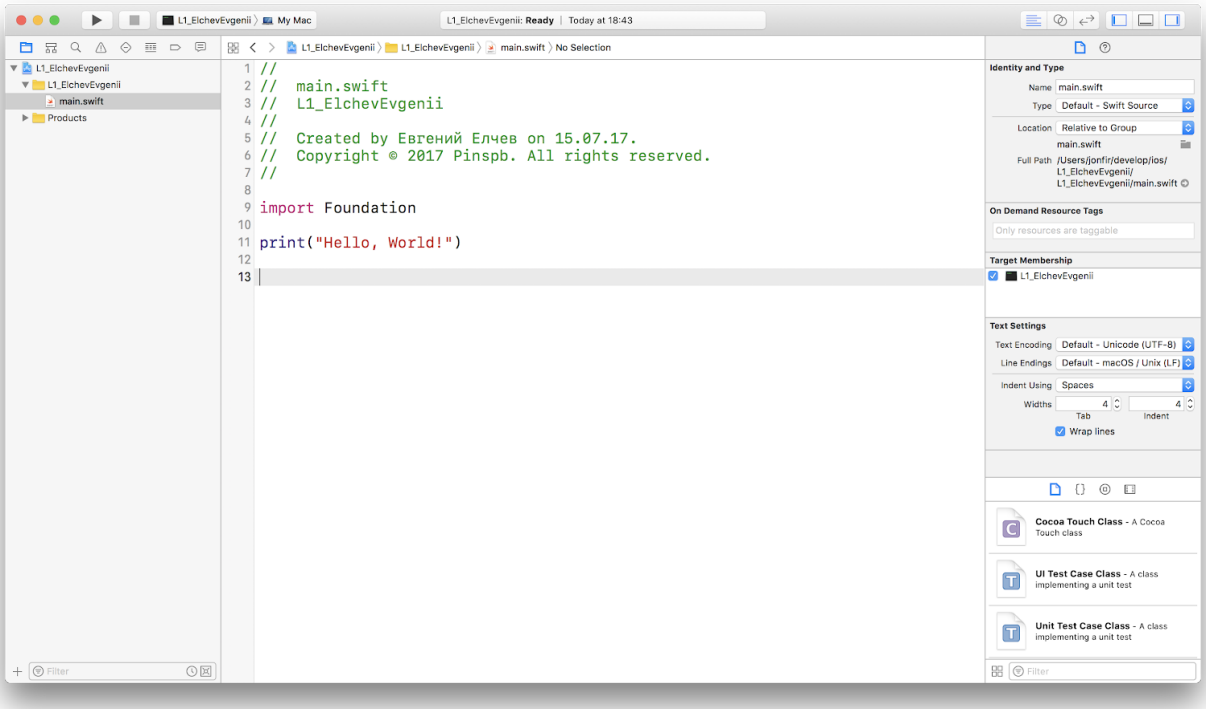
Готово! Ваше первое приложение создано и готово к выполнению.

Основные окна 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 рекомендуется к использованию. Вместо этого 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 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 файлы, связанные с основным).
7. Кнопка режима версионирования (переводит редактор в сдвоенный режим и отображает изменения, внесенные 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, попробуйте нажать на нее с зажатой клавишей «option», и 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 внизу панели.

Свойства файла вам почти не понадобятся, а вот справкой пользуйтесь как можно чаще. Многие вопросы, которые могут у вас возникнуть, 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 принадлежит окну консоли. Здесь отображаются ошибки, предупреждения, возникающие во время выполнения программы, и вывод вашей программы методами «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 предстоит 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 используются два ключевых слова для объявления переменных:

- let – объявляет переменную как константу. 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 ее значение, вы получите рекомендацию переопределить ее как константу. Константа в Swift фактически является указателем на константные данные – это касается и примитивных типов, и объектных, и коллекций.

Основные типы данных

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

Таблица основных 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	64-битное число с плавающей точкой	15 десятичных цифр
Float	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

```
3  var a1: Double = Double.NaN
4  a1 = -5.323
5  let a2: Float = 4.23232
6  let a3: Int = Int.min
7  var a33: Int8 = Int8.max
8  a33 = Int8.min
9  var a34: Int16 = Int16.max
10 a34 = Int16.min
11 var a35: Int32 = Int32.max
12 a35 = Int32.min
13 var a36: Int64 = INT64_MAX
14 a36 = Int64.min
15 let a4: UInt = 12
16 let a41: UInt32 = 10
17 let aa110 = 10
18 let a6: Character = "b"
19 let a7: Bool = false
```

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

Коллекции

Коллекция – это программный объект, который содержит в себе набор значений. Так, если нам необходимо сохранить ФИО одного человека, мы можем использовать переменную типа строка.

Чтобы сохранить ФИО нескольких людей, например футбольной команды, может использоваться одна из коллекций, называемая массив типа строка.

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

Таблица коллекций:

Коллекция	Описание	Диапазон
Array<T>	Массив типа T	Ограничен памятью
Dictionary<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 время выполнения этих операций может различаться. Будьте внимательны при выборе типа коллекции для каждого случая! Вы можете как ускорить выполнение программы, так и существенно его замедлить, если сделаете неверный выбор.

Настоятельно рекомендуем ознакомиться с «O-нотацией» – способом описания относительного времени выполнения операции в зависимости от числа элементов в коллекции. Справочник основных вариантов сложности алгоритма:

- O(1) – операция выполняется быстро, время выполнения не зависит от размера коллекции.
- O(log n) – время выполнения зависит от размера коллекции, но незначительно.
- O(n) – линейная зависимость между размером коллекции и временем выполнения. Если размер коллекции увеличится в 2 раза, то и время выполнения удвоится. Не самая плохая зависимость, но по 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 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.

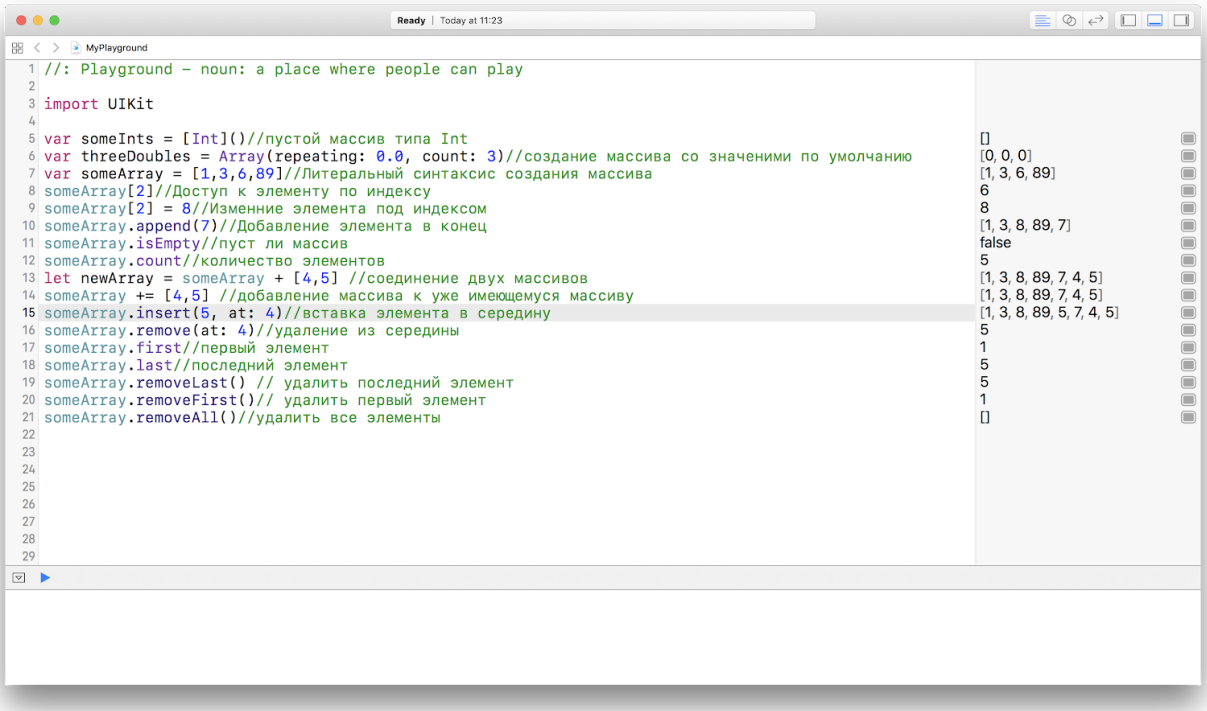
Массив гарантирует сохранение порядка элементов. Это значит, что без вмешательства со стороны программиста элементы не могут менять свою позицию внутри коллекции. Первый элемент всегда будет первым, пока вы его сами не переместите.

Доступ к элементу коллекции осуществляется по индексу смещения от начала. Первый элемент в коллекции будет доступен по индексу 0 (без смещения), второй – 1 (смещение на один элемент).

Использовать эту коллекцию стоит в том случае, когда важно сохранять порядок элементов. При необходимости их можно отсортировать 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, какая сложность у каждой из этих операций.

Dictionary

Коллекция данных «ключ-значение» в учении о структурах данных часто называется «хэш-таблица». Коллекция поддерживает три основные операции: добавление новой пары, поиск пары по ключу и удаление пары.

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

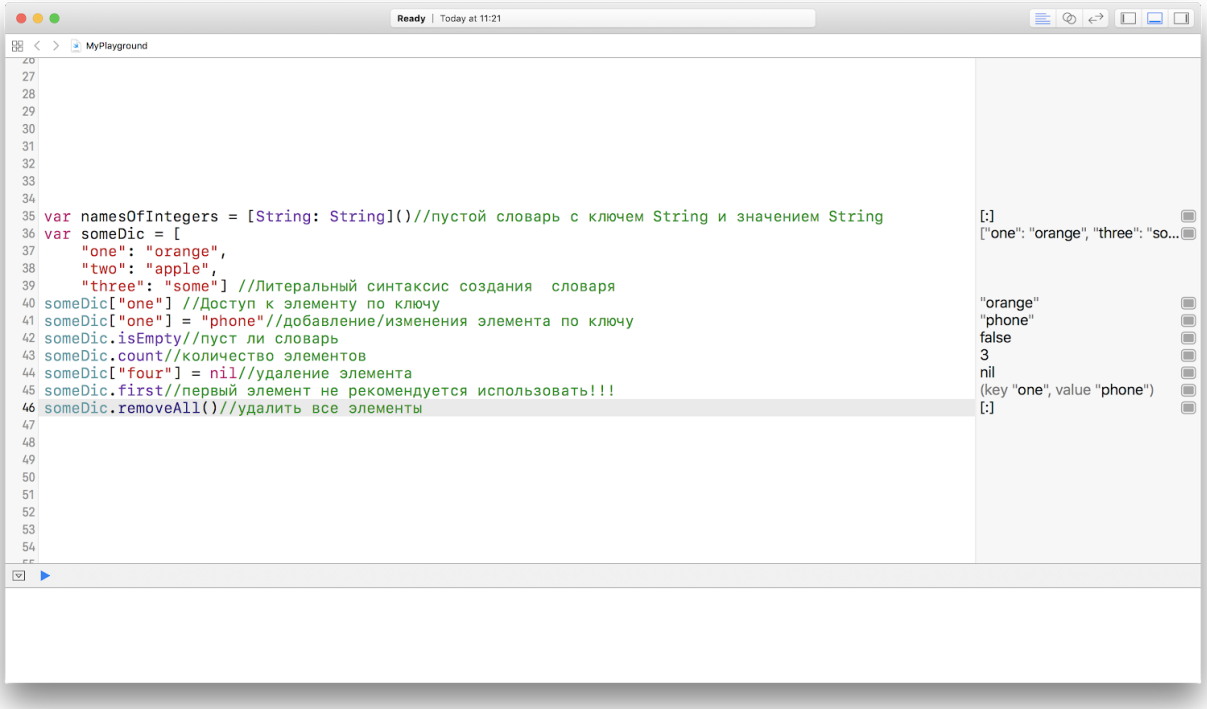
месте был ключ «one», а на втором – ключ «two», после инициализации эти ключи, скорее всего, поменяют свои позиции. Это необходимо, чтобы оптимизировать время доступа к элементам.

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

Использовать эту коллекцию стоит в том случае, когда вам часто приходится искать элемент по определенному параметру, который можно использовать в качестве ключа.

Синтаксис	Операция	Сложность
...	доступ по индексу	недоступен
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

Это коллекция уникальных данных, элементы внутри нее не могут повторяться. Если вы попытаетесь добавить в коллекцию элемент, который в ней уже содержится, то коллекция останется неизменной. Она реализована в виде хэш-таблицы и поддерживает математические операции с множествами.

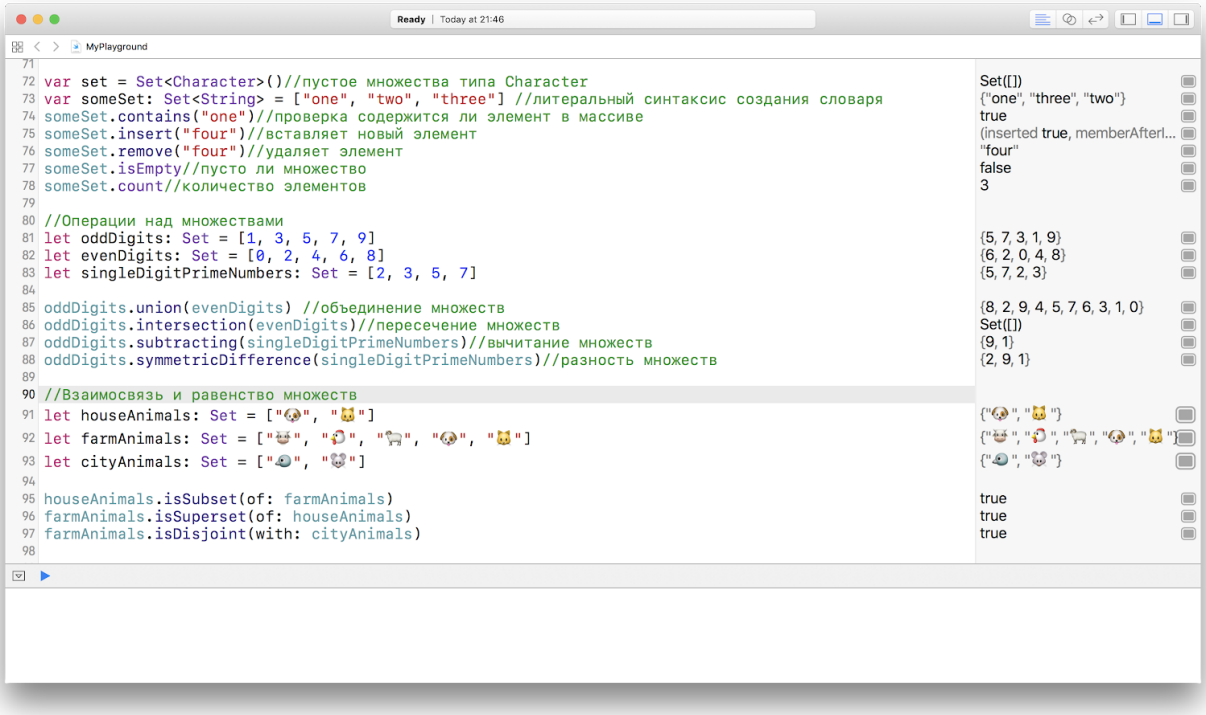
Коллекция не гарантирует сохранения порядка элементов.

Доступ к элементу коллекции осуществляется по самому элементу.

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

Синтаксис	Операция	Сложность
...	доступ по индексу	недоступен
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 переменные 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. Такая искусственная защита еще на этапе компиляции повышает качество кода и целевого приложения.

Swift может угадывать тип переменной при присвоении ей значения, но это не всегда полезно, особенно когда тип неочевиден.

```
// 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"
```

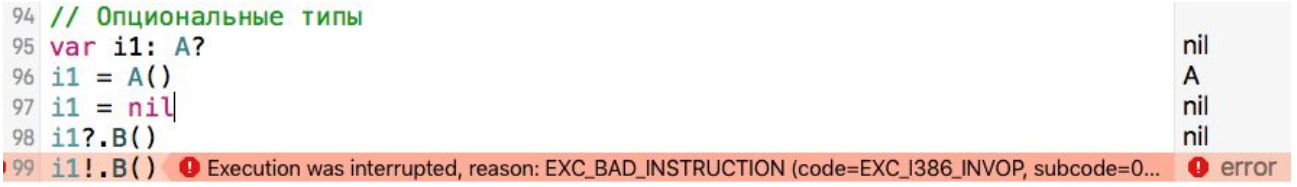
Опциональный тип

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

nil указывает на отсутствие каких-либо данных. Важно понимать, что даже пустая строка без единого символа (“”) – это все же строка. Аналогично обстоит ситуация с Int: 0 является значением. То есть переменные, содержащие “”, 0, 0.0, имеют значение, в отличие 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 – попросту говоря, упадёт. Для безопасного извлечения опционального типа необходимо использовать опциональную привязку.



Опциональная привязка

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

Пример использования опциональной привязки:

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

} 	

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

Когда мы меняем значение переменной на соответствующее 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

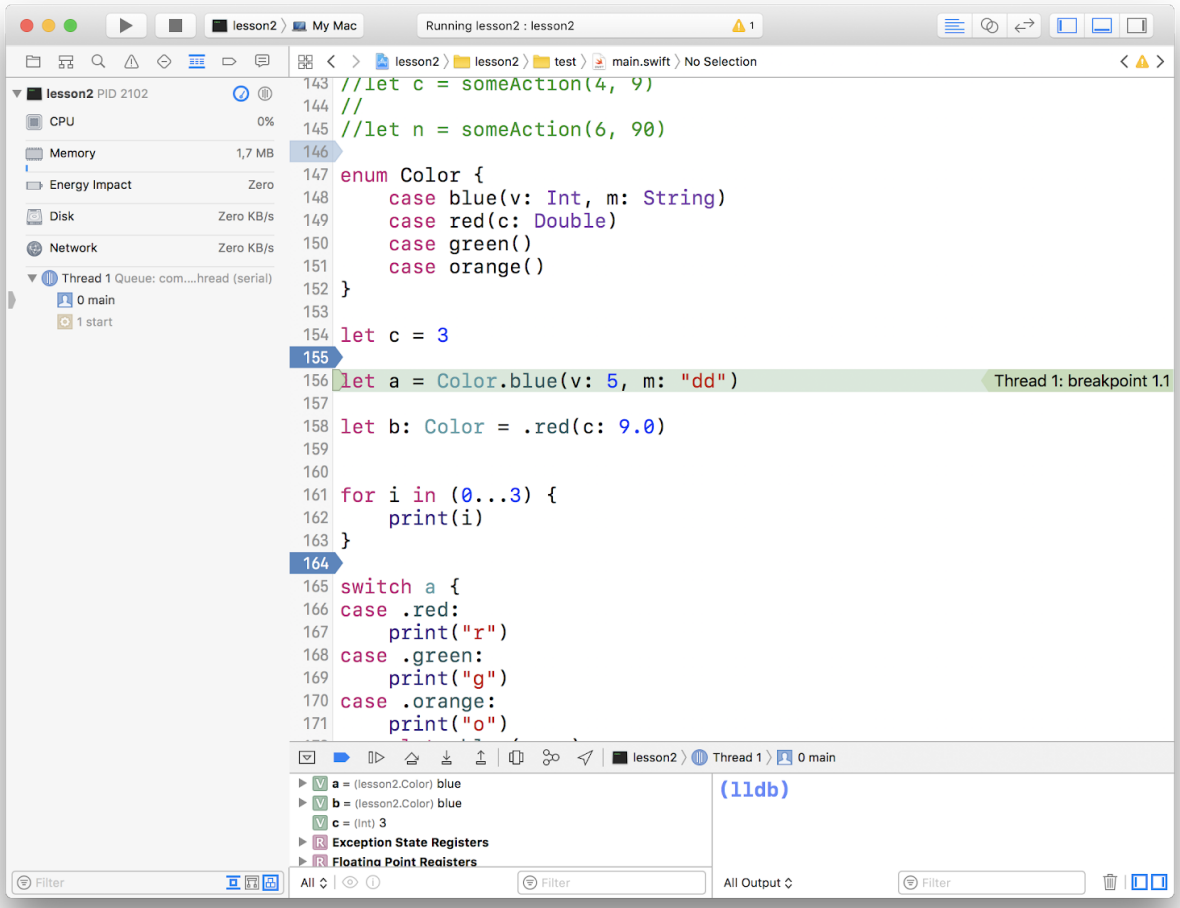
var a: Int? = 10 let b = 4 + (a ?? 6)	14

Отладчик (Debugger)

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

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

Чтобы начать отладку, достаточно задать 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. Как только выполнение дойдет до строки, на которой установлена точка, оно прекратится и активируется режим отладки. Дальше вы сами должны отдавать команды на выполнение следующего шага – без этого программа выполняться не будет.

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

Для управления у вас есть несколько кнопок, расположенных на нижней панели:

:

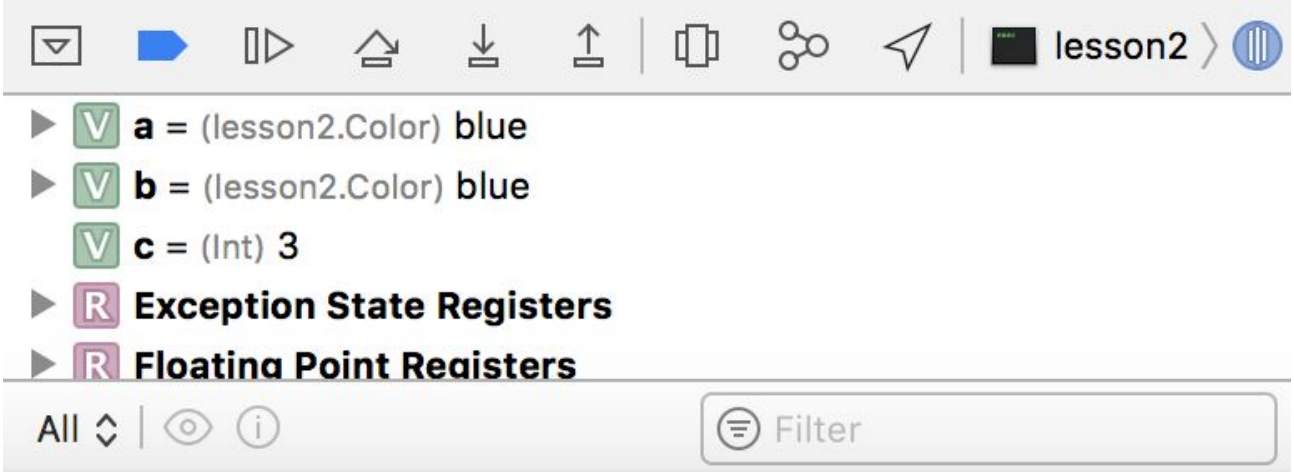
1. Кнопка возврата к нормальному режиму. После ее нажатия процесс выполнения программы снова пойдет в нормальном режиме.

2. Кнопка выполнения следующей команды без захода в функцию. При ее нажатии будет выполнена следующая строка кода, при этом, если 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 на пути попадется вызов функции, вы перейдете к ее описанию и сможете пошагово выполнить все ее инструкции.

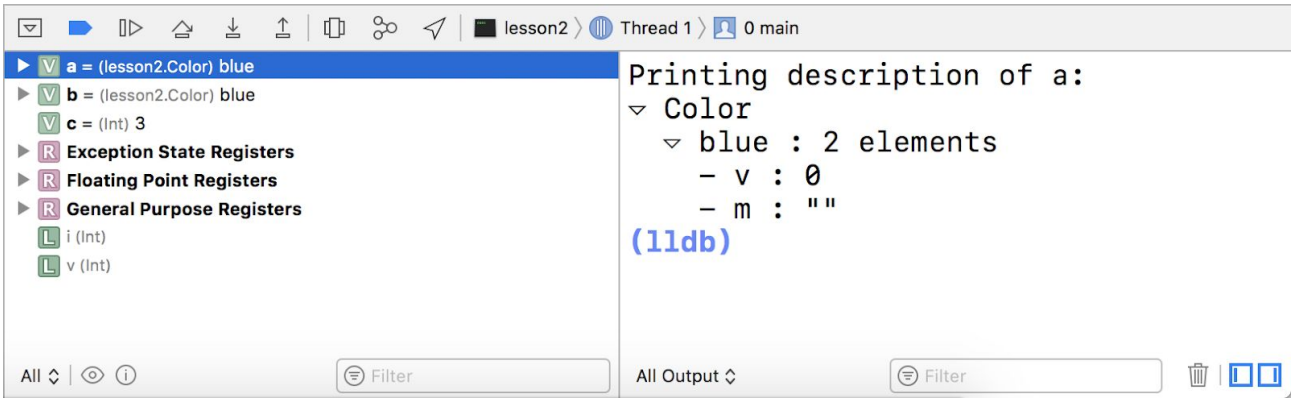
4. Кнопка выполнения следующей команды с выходом из функции. При ее нажатии, если вы уже находитесь внутри описания функции, вы немедленно покинете его и переместитесь к участку кода вызова этой функции.

Функция отслеживания порядка выполнения команд полезна сама по себе, но по-настоящему незаменимой ее делает возможность наблюдать за состоянием переменных.



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

Если 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.



Практические задания

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. Нужно 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.