

Содержание (сводка)

	Введение	21
1	Знакомство со Swift. <i>Приложения, системы и не только!</i>	31
2	По имени Swift. <i>Swift на практике</i>	57
3	Коллекции и управление. <i>Зацикленные на данных</i>	81
4	Функции и перечисления. <i>Повторное использование кода</i>	121
5	Замыкания. <i>Необычные гибкие функции</i>	155
6	Структуры, свойства и методы. <i>Типы, определяемые пользователем, и не только</i>	181
7	Классы, акторы и наследование. <i>О пользе наследования</i>	211
8	Протоколы и расширения. <i>Протокольные церемонии</i>	235
9	Опциональные типы, распаковка, обобщение и другое. <i>Неизбежные опциональные типы</i>	267
10	Знакомство со SwiftUI. <i>Пользовательские интерфейсы</i>	293
11	Практическое применение SwiftUI. <i>Круги, таймеры, кнопки – выбирайте!</i>	337
12	Приложения, веб-программирование и все такое. <i>Собирая все вместе</i>	365

Содержание (настоящее)

Введение

Ваш мозг и Swift. Вы пытаетесь изучить что-то новое, а ваш мозг хочет оказать вам услугу и как можно быстрее забыть выученное. Он думает: «Лучше оставить место для чего-то поважнее, например, от каких диких животных стоит держаться подальше или почему на сноуборде не стоит кататься нагишом». Так как же заставить ваш мозг думать, что ваша жизнь зависит от знания Swift?

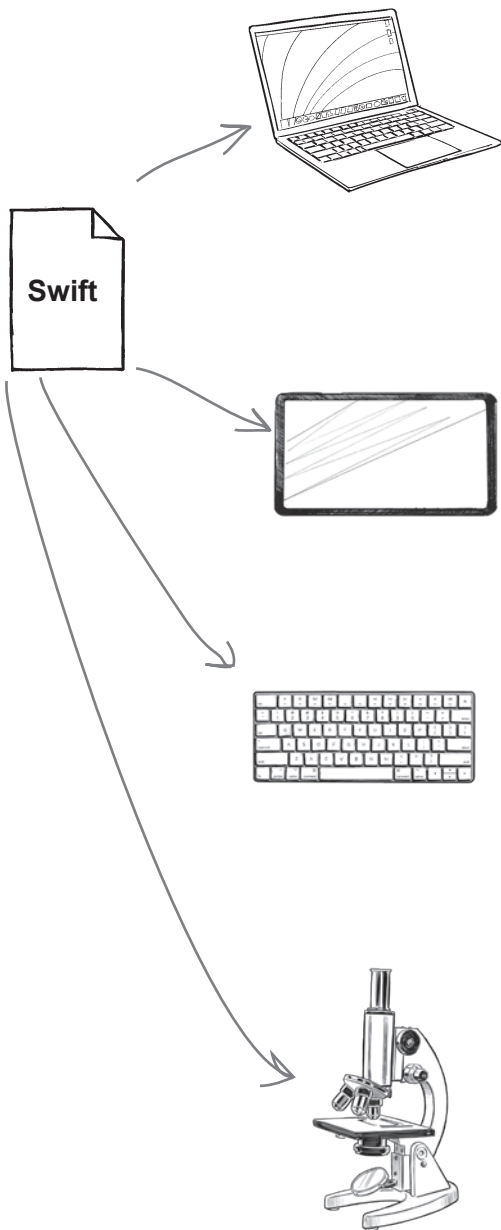
Для кого написана эта книга?	22
Мы знаем, о чем вы думаете	23
Метапознание: наука о мышлении	25
Вот что сделали мы	26
Что можете сделать вы	27
Примите к сведению	28

Знакомство со Swift

1

Приложения, системы и не только!

Swift — язык программирования, на который можно положиться. Вам не будет стыдно познакомить с ним вашу семью. Он безопасен, надежен, быстр, доступен и несложен. И хотя Swift получил наибольшую известность **как язык программирования для платформ Apple**, таких как iOS, macOS, watchOS и tvOS, проект с открытым кодом Swift также работает в Linux и Windows и постепенно набирает популярность как язык системного программирования, а также как серверный язык. На нем можно строить все что угодно, от мобильных приложений до игр, веб-приложений, фреймворков. Итак, за дело!



Swift — универсальный язык	32
Стремительная эволюция Swift	34
Стремление в будущее	35
Как вы будете писать код Swift	36
Путь, лежащий перед вами	38
Установка Playgrounds	39
Создание среды Playground	41
Использование среды Playground для написания кода Swift	42
Основные структурные элементы	44
Пример Swift	50
Поздравляем, вы сделали свои первые шаги в Swift!	55

По имени Swift

2 Swift на практике

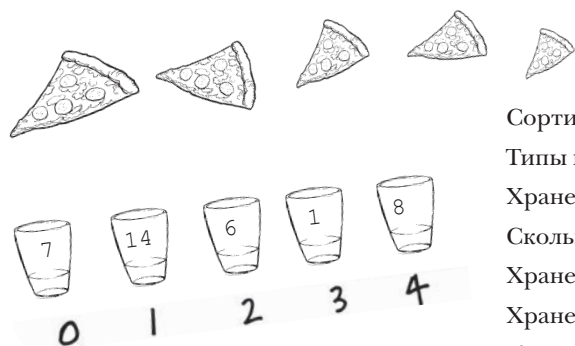
Вы уже знаете азы Swift. Но пришло время изучить основные элементы языка более подробно. Вы узнали достаточно, чтобы вас воспринимали серьезно, пора употребить новые знания на практике. Мы применяем Playgrounds для написания кода, использования команд, выражений, переменных и констант — основных структурных элементов Swift. В этой главе мы заложим основу вашей будущей карьеры программиста Swift. Вы освоите систему типов Swift и изучите основы представления текста в строковом виде. Не будем терять времени — еще чуть-чуть, и вы начнете писать код Swift.



Из чего строятся программы	58
Базовые операторы	59
Математические вычисления	60
Выражайтесь яснее	61
Имена и типы	64
Не все данные являются числами	68
Определение строковых переменных	70
Строковая интерполяция	76

3 Коллекции и управление Зацикленные на данных

Вы уже знаете о выражениях, операторах, переменных, константах и типах Swift. Пришло время собрать воедино все, что говорилось ранее, и на этой основе исследовать некоторые более сложные структуры данных и операторы Swift: **коллекции** и **управляющие команды**. В этой главе мы поговорим о сохранении коллекций данных в переменных и константах, о структурировании данных, обработке данных и работе с данными с использованием управляющих команд. Позднее в книге будут рассмотрены другие способы сбора и структурирования данных, а пока начнем с **массивов**, **множеств** и **словарей**.



Сортировка пиццы	82
Типы коллекций Swift	83
Хранение значений в массиве	84
Сколько элементов в массиве? И есть ли в нем элементы?	86
Хранение значений в множестве	87
Хранение значений в словаре	89
Кортежи	91
Хороший псевдоним пригодится каждому	92
Управляющие команды	94
Команды if	95
Команды switch	96
Построение команды switch	100
Операторы диапазонов	102
Более сложные команды switch	103
Многократное выполнение кода в циклах	104
Построение цикла for	105
Построение цикла while	108
Построение цикла repeat-while	109
Решение проблемы сортировки пиццы	110
Мы прошли большой путь!	112

4

Функции и перечисления

Повторное использование кода

Функции в языке Swift позволяют упаковать некоторое поведение или единицу работы в блок кода, который может вызываться из других частей вашей программы. Функции могут быть автономными, а могут определяться как часть класса, структуры или перечисления, где они обычно называются методами. При помощи функций можно разбить сложные задачи на меньшие части, более удобные и легкие в тестировании. Функции занимают центральное место в формировании структуры программ в Swift.



Функции Swift как средство повторного использования кода	123
Встроенные функции	124
Что можно узнать по встроенным функциям?	125
Когда могут пригодиться функции	128
Написание тела функции	129
Использование функций	130
Функции работают со значениями	132
С возвращением (из функций)	134
Переменное количество параметров	136
Что можно передать функции?	139
У каждой функции есть тип	140
Типы функций как типы параметров	144
Несколько возвращаемых типов	146
Функции не обязаны существовать автономно	148
Switch с перечислениями	151

Замыкания

5

Необычные гибкие функции

Функции полезны, но иногда нужно больше гибкости. Swift позволяет использовать **функцию как тип** — так же, как вы используете целое число или строку. **Это означает, что вы можете создать функцию и присвоить ее переменной.** После того как функция будет присвоена переменной, ее можно вызывать через эту переменную или передавать другим функциям в параметре. Когда вы создаете и используете функцию подобным образом, это называется **замыканием**. Замыкания очень полезны, потому что они способны сохранять **ссылки** на константы и переменные из контекста, в котором они были определены. Это называется **замыканием по значению**, отсюда и название.

Мы закроем вопрос с замыканиями.

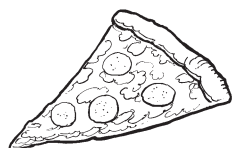
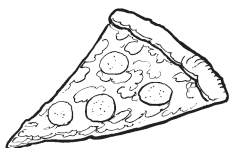


Знакомьтесь: простое замыкание	156
Замыкания с параметрами	158
Вычисление сводного значения	166
Свертки с использованием замыканий	167
Сохранение значений из внешней области видимости	170
Выходящие замыкания: нетривиальный пример	172
Автозамыкания обеспечивают гибкость	175
Сокращенные имена аргументов	178

6 Структуры, свойства и методы

Типы, определяемые пользователем, и не только

При работе с данными часто требуется определять собственные виды данных. Структуры, также нередко обозначаемые ключевым словом **struct**, позволяют создавать **типы данных, определяемые пользователем** (подобно тому, как `String` и `Int` являются типами данных), посредством **объединения других типов**. Использование структур для представления данных, с которыми работает ваш код Swift, позволяет отступить на шаг и подумать над взаимодействием данных, передаваемых в вашем коде. **Структуры могут содержать переменные и константы** (внутри структур они называются **свойствами**) и **функции** (называемые **методами**). Добавим немного порядка в ваш мир и займемся углубленным изучением структур.



Сделаем пиццу во всей красе...	183
Инициализатор ведет себя точно так же, как функция	189
Статические свойства повышают гибкость структур	192
Функции в структурах	196
Методы	196
Изменение свойств с использованием методов	197
Вычисляемые свойства	199
Get- и set-методы для вычисляемых свойств	203
Реализация set-метода	204
Строки Swift в действительности являются структурами	206
Для чего нужны отложенные свойства	207
Использование отложенных свойств	208

7

Классы, акторы и наследование

О пользе наследования

Структуры показали, насколько полезным может быть построение типов, определяемых пользователем. Но у Swift в запасе есть и другие средства, включая *классы*. Классы похожи на структуры: они позволяют создавать *новые типы данных, содержащие свойства и методы*. Кроме того что они являются *ссылочными типами*, то есть экземпляры конкретного класса совместно используют одну копию своих данных (в отличие от структур, которые являются типами-значениями, чьи данные копируются), *классы поддерживают наследование*. Наследование позволяет построить один класс на базе другого класса.

Структура с другим именем (и это имя — класс)	212
Наследование и классы	214
Переопределение методов	218
Финальные классы	222
Автоматический подсчет ссылок	226
Изменяемость	227

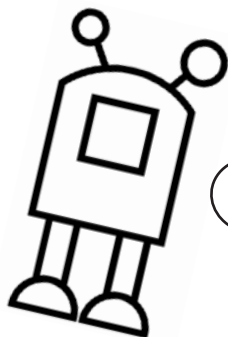
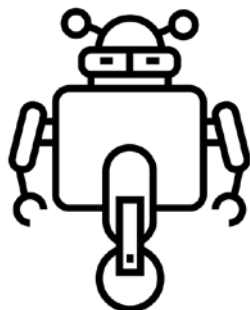


8 Протоколы и расширения

Протокольные церемонии

Вы все знаете о классах и наследовании, но у Swift еще есть немало средств структурирования программ. Знакомьтесь: протоколы и расширения. Протоколы в Swift позволяют определить шаблон с перечнем методов и свойств, необходимых для некоторой цели или некоторого блока функциональности. Протокол включается классом, структурой или перечислением, в которых содержится его фактическая реализация. Типы, которые предоставляют необходимую функциональность, называются поддерживающими этот протокол. Расширения позволяют легко добавлять новую функциональность в существующие типы.

Фабрика роботов	238
Наследование протоколов	243
Изменяющие методы	245
Протоколы как типы и коллекции	248
Вычисляемые свойства в расширениях	252
Расширение протокола	256
Полезные протоколы и вы	257
Поддержка протоколов Swift	260



Может, мы и не протокольные роботы, но мы узнаем хороший протокол, когда увидим его!

Опциональные типы, распаковка, обобщение и другое

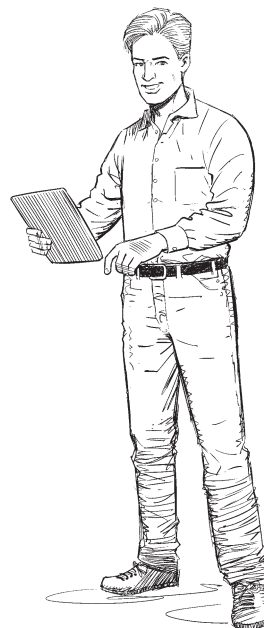
9

Неизбежные опциональные типы

Обработка несуществующих данных может быть весьма непростым делом. К счастью, в Swift для этого существует решение: опциональные типы. В Swift опциональный тип позволяет работать со значением или выполнить действия при отсутствии значения. Это одно из многих проявлений безопасности при проектировании Swift. Ранее вы уже встречались с опциональными типами в коде, а теперь мы изучим их более подробно. Опциональные типы улучшают безопасность Swift, потому что с ними снижается риск написания кода, который перестает работать при отсутствии данных, или возврата значения, которое в действительности значением не является.



Обработка отсутствующего значения	269
Для чего могут понадобиться опциональные типы	270
Опциональные типы и обработка отсутствующих данных	273
Распаковка опциональных типов	274
Распаковка опциональных типов с ключевым словом guard	277
Принудительная распаковка	278
Обобщения	288
Очередь с обобщениями	289
Новый тип Queue	290



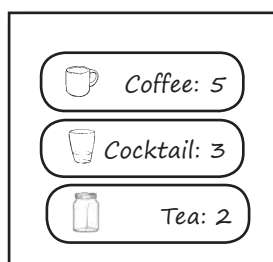
10

Знакомство со SwiftUI

Пользовательские интерфейсы

Пришло время применить на практике все приемы, возможности и компоненты Swift, о которых вы узнали в книге: мы займемся построением пользовательских интерфейсов. В этой главе мы сведем все воедино для построения первого настоящего пользовательского интерфейса. Он будет строиться на основе **SwiftUI, UI-фреймворка для платформ Apple**. Мы по-прежнему будем использовать Playgrounds (по крайней мере на первом этапе), но все, что здесь будет делаться, заложит фундамент для реальных приложений iOS. Приготовьтесь: в этой главе будет много кода и новых концепций. Вдохните поглубже и переверните страницу, чтобы с головой погрузиться в *SwiftUI*.

*У вас будет время выпить,
но только после того, как вы
освоите SwiftUI...*



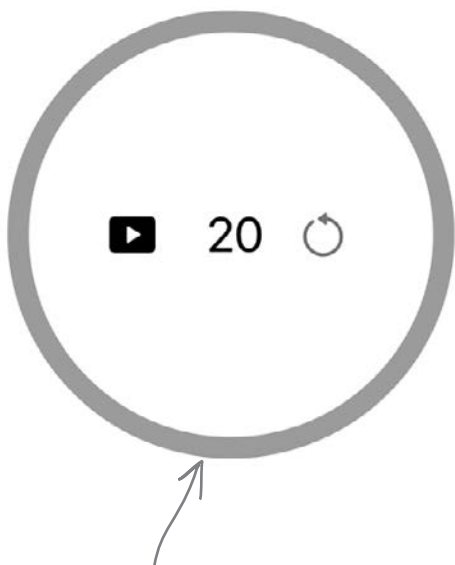
А что это вообще такое – UI-фреймворк?	294
Ваш первый пользовательский интерфейс SwiftUI UI	296
Строительные блоки пользовательских интерфейсов	300
Создаем список (и неоднократно возвращаемся к нему, чтобы довести до совершенства)	301
Пользовательские интерфейсы с состоянием	303
Как работают кнопки	304
Давайте посмотрим, насколько далеко мы зашли	307
ЗАДАЧА: Написать приложение на базе SwiftUI	311
Создание в Xcode нового проекта SwiftUI для iOS	312
Ваша среда Xcode должна выглядеть примерно так	314
Создание нового типа для хранения элемента списка задач	315
Однозначная идентификация каждого элемента списка задач	318
Создание пользовательского интерфейса приложения	319
Запустите приложение и посмотрите, что произойдет...	325
Реализация сохранения списка задач	326
Значит, это и есть UI-фреймворк?	330

11

Практическое применение SwiftUI

Круги, таймеры, кнопки — выбирайте!

SwiftUI вовсе не ограничивается кнопками и списками. В интерфейсах также можно использовать геометрические фигуры, анимации и многое другое! В этой главе будут рассмотрены некоторые **расширенные возможности построения пользовательских интерфейсов в SwiftUI** и их связывания с источниками данных, содержимое которых не генерируется пользователем (как в случае со списком задач). SwiftUI позволяет строить **пользовательские интерфейсы** с обработкой **событий**, поступающих из разных источников. Мы будем работать в Xcode, среде разработки от компании Apple, а основное внимание будет уделяться приложениям для iOS, но все, что вы узнаете в этой главе, в равной степени применимо к SwiftUI для iPadOS, macOS, watchOS и tvOS. Вперед, глубины SwiftUI ожидают вас!



Создайте свой собственный круг!

Что интересного можно сделать с UI-фреймворком?	338
Создание нового проекта SwiftUI для iOS	341
Пользовательский интерфейс и функциональность Executive Timer	343
Создание основных элементов приложения	344
Составляющие пользовательского интерфейса	345
Настройка пользовательского интерфейса приложения	346
Программирование составляющих пользовательского интерфейса	347
Объединение трех элементов	350
Завершающие штрихи	352
Запустите приложение и посмотрите, что произойдет...	353
Представления с вкладками	354
Создайте представления, которые вам нужны	354
Постройте представление TabView, содержащее ваши представления	354
Создание нового представления ContentView с вкладками	360
Создание вкладок и TabView	361
Запустите новую версию Executive Timer	363

12

Приложения, Веб-программирование и Все такое Собирая все вместе

Вы значительно продвинулись в изучении Swift. Вы освоили Playgrounds и Xcode. Было понятно, что когда-нибудь нам придется **попрощаться**, и сейчас этот момент настал. Расставаться нелегко, но мы знаем, что вы справитесь. В этой главе — последней, в которой мы будем вместе с вами (в этой книге), — мы еще раз **пройдемся по многим концепциям**, которые вы изучили, и совместно построим несколько приложений. Мы убедимся в том, что ваши навыки Swift закреплены, и дадим некоторые рекомендации относительно того, **что делать дальше**, — своего рода домашнее задание, если хотите. Это будет интересно, и мы расстанемся на высокой ноте.



Путешествие должно завершиться...	366
Построение заставки	371
Пошаговая сборка экрана заставки	372
Совместное использование состояния	378
На помощь приходит старый знакомый...	379
Построение приложения с несколькими представлениями, совместно использующими состояние	380
Построение приложения с двумя представлениями	381
ObservableObject	381
Первое представление	382
Второе представление	383
И снова первое представление	384
Первое представление (продолжение)	385
AsyncImage с наворотами	390
Varog: веб-фреймворк для Swift	394
Отправка данных средствами Varog	397