Часть II. Базовые возможности Swift

Добро пожаловать во вселенную Swift 5!

Пришло время перейти к изучению этого прекрасного языка программирования. С каждым пройденным разделом, с каждой изученной страницей и каждым выполненным заданием вы будете все ближе к своей цели — к разработке потрясающих приложений и, возможно, к хорошему заработку.

Swift — крайне интересный язык программирования. Если ранее вы работали с другими языками программирования, то уже скоро заметите их сходство с детищем Apple. Данная часть книги расскажет вам о базовых понятиях, которые предшествуют успешному программированию, и обучит основам синтаксиса и работы с различными фундаментальными механизмами, которые легли в основу всей системы разработки программ на языке Swift.

Как и многие другие языки, Swift активно использует переменные и константы для хранения значений, а для доступа к ним служат идентификаторы (имена). Более того, Swift вывел функциональность переменных и констант на новый качественный уровень. И скоро вы в этом убедитесь. По ходу чтения книги вы узнаете о многих потрясающих нововведениях, которые не встречались вам в других языках, но на отсутствие которых вы, возможно, сетовали.

- ✓ Глава 4. Отправная точка
- ✓ Глава 5. Фундаментальные типы данных

4 Отправная точка

У каждого из вас свой уровень подготовки и разный опыт за плечами, каждый прошел свой собственный путь независимо от других. Все это накладывает определенные сложности при написании учебного материала: для кого-то он может быть слишком простым, а для кого-то чрезвычайно сложным! Найти золотую середину порой не так просто.

ПРИМЕЧАНИЕ В первую очередь эта книга ориентирована на начинающих разработчиков. Помните, что вы всегда можете пропустить то, что уже знаете. Не стоит выражать свое недовольство из-за наличия излишнего, с вашей точки зрения, материала.

4.1. Как компьютер работает с данными

С момента изобретения микропроцессора основные принципы работы персонального компьютера не претерпели существенных изменений. Он все так же орудует нулями и единицами, проводя с ними совсем нехитрые операции. Возможно, вы удивитесь, но современные электронные устройства не такие уж и умные, как нам рассказывают в рекламе: весь смарт-функционал определяется инженерами, проектирующими схемы, и программистами.

В отличие от обыкновенного калькулятора, компьютер предназначен для выполнения широкого (можно сказать, неограниченного) спектра задач. В связи с этим в его аппаратную архитектуру уже заложены большие функциональные возможности, а вся логика работы определяется программным обеспечением, создаваемым программистами.

В общем случае при максимальном упрощении можно выделить три основных функциональных уровня, обеспечивающих работу компьютера (рис. 4.1).

Рис. 4.1. Функциональные уровни компьютера

Аппаратный уровень представляет собой набор типовых блоков (физического оборудования), самыми важными из которых являются центральный процессор (*CPU*, Central Processing Unit) и оперативная *память*.

CPU — сердце любого широкофункционального электронного устройства (персональные компьютеры, планшеты, смартфоны и т. д.). Его изобретение стало настоящей революцией, ознаменовавшей собой появление современных вычислительных машин. Кстати, компьютеры называются вычислительными машинами неспроста. Их работа заключается в обработке чисел, в вычислении различных значений через выполнение простейших арифметических и логических операций, называемых инструкциями. Все данные в любом компьютере в итоге превращаются в числа. Причем неважно, говорим мы о фотографии, любимой песне или книге — для компьютера все это представляется в виде чисел.

CPU как раз и предназначен для работы с числами. Он получает на вход несколько значений и в зависимости от переданной команды выполняет с ними заданные операции.

В ходе этой работы в качестве временного хранилища данных выступает оперативная память. CPU постоянно производит с ней обмен данными.

Память — это основное средство хранения данных, используемое программами в ходе их функционирования. Память компьютера состоит из миллионов отдельных ячеек, каждая из которых может хранить информацию. Также у каждой ячейки присутствует свой уникальный адрес.

Рассмотрим принцип работы оперативной памяти. На рис. 4.2 приведен схематичный вид памяти с ячейками № 1669–1680. В настоящее время эти ячейки пусты, то есть не имеют записанной в них информации.

Рис. 4.2. Память с набором ячеек

ПРИМЕЧАНИЕ Приведенные схемы и описания максимально упрощены. На самом деле компьютер на аппаратном уровне оперирует исключительно двумя цифрами: 0 и 1. Он знает всю их мощь и по полной их использует.

Приведенные далее примеры хранения и обработки десятичных чисел показаны для лучшего восприятия вами учебного материала. Повторюсь, что в действительности эти числа переводятся в двоичную систему, после чего происходит их запись в память и обработка в процессоре в виде последовательностей 0 и 1.

Предположим, что вы запустили программу «Калькулятор». При ее использовании возникает необходимость временно хранить в памяти различные числовые значения. Каждое значение при этом, скорее всего, будет занимать не одну, а целую группу ячеек (логически объединенных). Каждая такая группа будет иметь уникальное имя, она может быть названа *хранилищем данных*. Данные об используемых в программе хранилищах записываются в специальный реестр. При этом указывается имя хранилища, а также адреса используемых ячеек.

Предположим, было создано два хранилища с именами «Группа 1» и «Группа 2». В первом содержится числовое значение 23, а во втором — 145 (рис. 4.3).

В будущем для того, чтобы получить значение 23 (хранящееся в «Группе 1»), нет необходимости сканировать все ячейки оперативной памяти на предмет записанных в них данных — достаточно лишь обратиться к «Группе 1» по имени и получить записанное в ее ячейках значение.

Рис. 4.3. Реестр с двумя хранилищами данных

Если вернуться к рис. 4.1, то вторым идет уровень **операционной системы**. В ходе разработки вы будете не так часто задумываться о нем.

В общем случае операционная система — это посредник между вашей программой и аппаратной частью. Благодаря ОС, приложения могут использовать все возможности компьютера: выполнение математических операций, передача данных по Wi-Fi и Bluetooth, отображение графики, воспроизведение песен группы Queen и многое-многое другое.

Операционная система предоставляет интерфейсы, которые могут использоваться в ходе работы программного обеспечения. Хотите сложить два числа? Не вопрос! Напишите соответствующую команду, и ОС сама передаст ваши байты куда следует, после чего вернет результат.

Для того чтобы отдавать различные команды на выполнение каких-либо операций, как раз и нужен третий уровень (см. рис. 4.1).

«**Программный**» уровень предполагает использование языков программирования для создания приложений. Все довольно просто: разработчик пишет команды, ОС обрабатывает их, после чего передает их для выполнения на аппаратный уровень. Каждая команда на языке программирования может подразумевать под собой множество циклов работы с памятью и выполнение сотен инструкций в процессоре.

В общем виде **программирование** — это автоматизация процессов для решения определенных задач с помощью компьютера. В ходе написания программы разработчик манипулирует данными и определяет реакцию программы на них.

Каждая программа взаимодействует с данными, которые могут быть получены из различных источников: загружены из файлов на жест-

ком диске, введены пользователями, получены из Сети и т. д. Часто эти данные нужно не просто получить, обработать и отправить в указанное место, но и обеспечить их хранение с целью использования в будущем.

Так, например, получив ваше имя (предположим, что вы ввели его при открытии приложения), программа запишет его в память и при необходимости будет загружать оттуда, чтобы отобразить в соответствующем месте графического интерфейса.

4.2. Базовые понятия

В предыдущем разделе, когда мы говорили об оперативной памяти, то упоминали понятие «хранилище данных».

Хранилище данных — это виртуальный объект, обладающий некоторым набором свойств, к примеру: записанное значение, количество ячеек, адрес в оперативной памяти, тип информации (числовой, строковый) и т. д.

Практически весь процесс программирования состоит в том, чтобы создавать (определять) объекты, устанавливать (инициализировать) их значения, запрашивать эти значения и производить с ними операции.

В программировании Swift при работе с хранилищами данных выделяют два важнейших понятия: объявление и инициализация.

Объявление — это создание нового объекта (хранилища данных).

Инициализация — это присвоение объявленному объекту определенного значения.

В примере выше были объявлены хранилища данных с именами «Группа 1» и «Группа 2», после чего их проинициализировали значениями `23` и `145`.

Рассмотрим простейший арифметический пример: значение «Группы 1» (`23`) умножить на значение «Группы 2» (`145`) (листинг 4.1).

Листинг 4.1.

```
23 * 145
```

Данный математический пример является выражением, то есть законченной командой на языке математики. В нем можно выделить одну операцию умножения и два операнда, с которыми будут производиться действия (`23` и `145`).

Чтобы произвести данную операцию умножения, нужно выполнить следующую последовательность действий:

- Получить значение «Группы 1».
- Получить значение «Группы 2».
- Произвести операцию умножения значения «Группы 1» на значение «Группы 2».
- Объявить хранилище данных с именем «Группа 3» для хранения результата умножения.
- Проинициализировать хранилищу «Группа 3» результат операции.

В листинге 4.1 указателем на то, что необходимо произвести операцию умножения, служит оператор * (умножение). В результате выполнения выражения будет получено новое значение `3335`, записанное в группу ячеек с именем «Группа 3» (рис. 4.4).

Рис. 4.4. Результат умножения двух чисел помещен в новое хранилище

Помимо этого, хочется обратить внимание на то, что хранилища данных могут содержать в себе не только цифры, но и другие виды информации (текстовую, графическую, логическую и др.). Виды информации в программировании называются типами данных. При объявлении хранилищ вы **всегда** будете сообщать программе, данные какого типа она должна хранить (целые числа, дробные числа, текст, рисунок и т. д.).

Любая программа — это набор выражений или, другими словами, набор четких команд, понятных компьютеру. Например, выражение «Отправь этот документ на печать» укажет компьютеру на необходимость выполнения некоторых действий в определенном порядке: загрузка документа, поиск принтера, отправка документа принтеру

и т. д. Выражения могут состоять как из одной, так и из нескольких команд, как, например: «Отправь этот файл, но перед этим преобразуй его из docx в rtf».

Выражение — завершенная команда, выполняющая одну или несколько операций. Выражение может состоять из множества операторов и операндов.

Оператор — это минимальная автономная функциональная единица (слово или группа слов), выполняющая определенную операцию.

Операнд — это значение, с которым оператор производит операцию.

Все зарезервированные языком программирования наборы символов называются **ключевыми словами**.

Ранее мы рассматривали пример умножения двух чисел `23` и `145` с последующим объявлением нового хранилища. Всего было выделено пять отдельных шагов, которые, в свою очередь, и являлись полноценным выражением.

Если рассмотреть это выражение со стороны синтаксиса языка программирования, то его можно представить в виде следующей строки (листинг 4.2).

Листинг 4.2

```
Создать_хранилище Группа 3 = Группа 1 * Группа 2
```

Обратите внимание, как легко и просто с помощью одного выражения производятся необходимые нам операции.

4.3. Введение в операторы

В Swift выделяют следующие основные виды операторов:

- **Простые операторы**, выполняющие операции с некоторыми значениями (операндами). В их состав входят унарные и бинарные операторы.
  - **Унарные операторы** выполняют операцию с одним операндом (например, `-a`). Они могут находиться перед операндом, то есть быть префиксными (например, `!b`), или после него, то есть быть постфиксными (например, `i?`).
  - **Бинарные операторы** выполняют операцию с двумя операндами (например, `1+6`). Оператор, который располагается между операндами, называется инфиксным.

- **Структурные операторы** влияют на ход выполнения программы. Например, останавливают выполнение программы при определенных условиях или указывают программе, какой блок кода должен быть выполнен при определенных условиях.

В будущем в ходе создания приложений вы будете регулярно применять различные операторы, а также при необходимости создавать собственные.

Обратимся к Swift и Xcode Playground. Перед вами уже должен быть открыт созданный ранее проект.

ПРИМЕЧАНИЕ Если вы закрыли предыдущий playground-проект, то создайте новый. Для этого откройте Xcode, в появившемся меню выберите пункт «Get started with a playground», после чего выберите Blank и укажите произвольное название нового проекта. Перед вами откроется рабочее окно Xcode playground (рис. 4.5).

Рис. 4.5. Окно нового playground

Взгляните на код в Xcode Playground (листинг 4.3).

Листинг 4.3

```
import UIKit
var str = "Hello World"
```

Предназначение первой строки мы подробно рассмотрим позже, а сейчас вкратце: она осуществляет подключение модуля, расширяющего функциональные возможности языка.

Рассмотрим следующую строчку кода (листинг 4.4).

Листинг 4.4

```
var str = "Hello World"
```

В ней объявляется хранилище данных, после чего инициализируется текстовое значение `Hello World`. Данное выражение можно разбить на отдельные шаги (рис. 4.6).

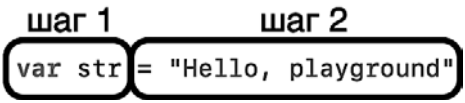

Рис. 4.6. Выражение состоит из отдельных шагов

Шаг 1: С помощью ключевого слова (оператора) `var` объявляется новое хранилище данных с именем `str`.

Шаг 2: В оперативную память записывается текстовое значение `Hello World`, после чего создается указатель в хранилище данных на этот участок памяти. Или, другими словами, в созданное хранилище `str` записывается значение `Hello World`. Этот процесс и называется инициализацией значения с помощью оператора `=` (присваивания).

Если представить текущую схему оперативной памяти, то она выглядит так, как показано на рис. 4.7.

Рис. 4.7. Результат создания нового хранилища данных

В данном выражении используется два оператора. На шаге 1 — это оператор `var`, на шаге 2 — оператор `=`. В двух следующих разделах рассмотрим подробнее их работу, но начнем с оператора инициализации.