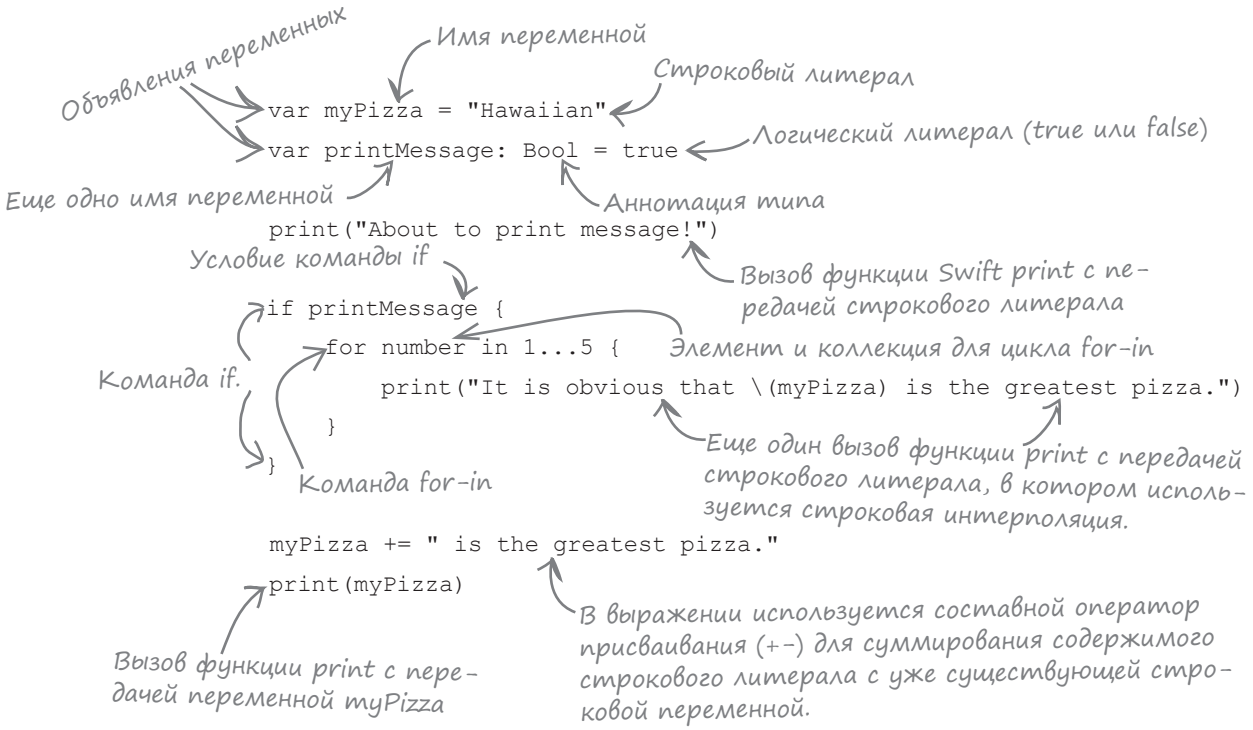
Swift на практике

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

Каждая программа, которую вы пишете на Swift, *составляется* из разных элементов. В главах этой книги вы узнаете, что это за элементы и как объединить их, чтобы программа на языке Swift сделала именно то, что вам нужно.



Ключевые моменты

- Программы на языке Swift строятся из множества разных элементов.
- Наиболее фундаментальные структурные элементы — выражения, команды и объявления.
- Команда определяет действие, выполняемое вашей программой Swift.
- Выражение возвращает результат, то есть некоторое значение.
- Объявление вводит в программу Swift новое имя, например имя переменной.
- Переменные предназначены для хранения некоторого типа.
- Типы определяют, какие данные хранятся в некоторой области памяти.
- Литералы представляют значения, записанные непосредственно в программе (например, `5`, `true` или `"Hello"`).

Базовые операторы

Первый элемент, который мы рассмотрим, — **оператор**. Оператор представляет собой знак или последовательность символов, используемые для изменения, проверки или объединения значений, с которыми вы работаете в своей программе.

Операторы могут использоваться для выполнения математических вычислений, логических операций, присваивания значений и т. д. В Swift поддерживаются все операторы, встречающиеся в большинстве языков программирования, а также некоторые операторы, относительно уникальные для Swift... или по крайней мере используемые в Swift уникальным образом. Вы больше узнаете об операторах в процессе чтения книги.

Операторы бывают *унарными*, *бинарными* и *тернарными*:

Это всего лишь означает, что операторы могут работать с одним значением (унарные операторы), двумя значениями (бинарные операторы) или тремя значениями (тернарные операторы).

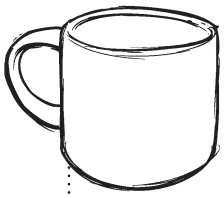
«-» — унарный оператор. Он применяется к одному значению, в данном случае 1.

```
-1
11 + 2
a ? b : c
```

«+» — бинарный оператор. Он работает с двумя значениями (в данном случае 11 и 2).

? и : — компоненты тернарного оператора. Он работает с тремя значениями (в данном случае a, b и c). Он будет рассмотрен позднее.

Оператор — знак или последовательность символов, предназначенные для изменения, проверки или объединения значений.



Расслабьтесь

Если вы не знаете все возможные операторы и их обозначения, не паникуйте.

По мере расширения ваших знаний и опыта использования Swift многие операторы будут казаться вам совершенно естественными (а многие уже кажутся, например математические операторы). Но со многими операторами этого не произойдет.

Для программиста абсолютно нормально искать справочную информацию об операторах, ключевых словах и синтаксисе, даже если он программировал на этом языке годами или десятилетиями. Нет ничего плохого в том, чтобы лишний раз освежить память.

Математические вычисления

В ходе программирования вы часто выполняете математические вычисления. Swift не подведет вас в том, что касается математики. Поддерживаются все классические операторы, включая операторы для *сложения*, *вычитания*, *деления*, *умножения* и *вычисления остатка*.

Каждая из этих строк, содержащих оператор, называется выражением.

```
100 + 50
92 - 8
8 / 4
9 * 10
42 % 2
```

Складывает 100 и 50 бинарным оператором сложения +.

Вычитает 8 из 92 бинарным оператором вычитания -.

Делит 8 на 4 бинарным оператором деления /.

*Умножает 9 на 10 бинарным оператором умножения *.*

Еще не забыли школьный курс математики? Вспомните порядок выполнения операций!

Оператор вычисления остатка. Возвращает остаток от деления одного числа на другое число.

Упражнение

Создайте новую среду Playground и займитесь вычислениями! Операторы можно комбинировать в одном выражении, как вас учили в школе.

Попробуйте умножить какое-нибудь число на 42, затем разделить результат на 3 и вычесть 4.

Попробуйте вычислить результат 4 + 5 * 5.

Проверьте полученное число на четность оператором вычисления остатка (если остаток от деления на 2 равен нулю, то это четное число). Как вы думаете, какой результат вы получите?

Ответ на с. 65.

Выражайтесь яснее

Набор значений и операторов, который дает некоторый результат, называется выражением.

На самом деле любой код, возвращающий значение, называется выражением. Даже если производимое значение полностью совпадает с выражением, это все равно выражение.

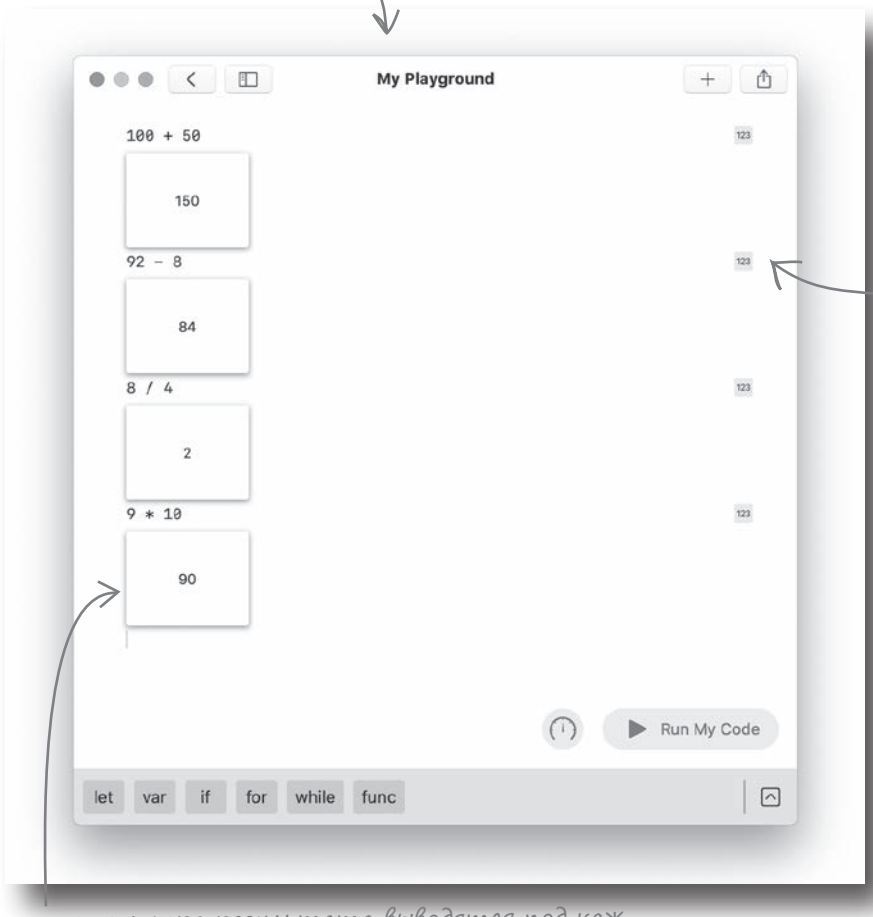
```
917 - 17 + 4
```

Это выражение, потому что это набор значений ((917, 17 и 4) и операторов (- и +), который генерирует результат (904).

Выражайтесь яснее

Если вы введете эти **выражения** в среде Playground и выполните их, вы увидите их значение (то есть результат, полученный при *вычислении* выражения):

```
100 + 50
92 - 8
8 / 4
9 * 10
```



Так как у каждого из этих выражений существует результат, для них можно добавить области просмотра.

Области просмотра результата выводятся под каждым выражением, для которого они были добавлены.

Не используйте смешанный стиль включения пробелов...

...выберите один стиль и придерживайтесь его. Таким образом, ваши выражения и операторы могут иметь вид `8/4` *или* `8 / 4`*, но только не* `8 /4` *или* `8/ 4`.

Ключевые моменты

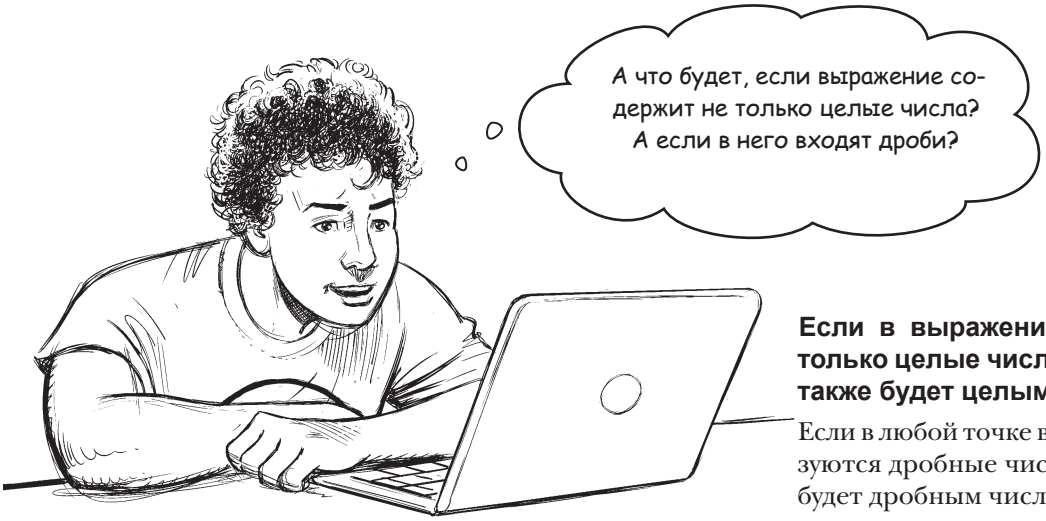
- Программы Swift строятся из команд.
- Простая команда строится из выражений и объявлений.
- Выражение представляет собой код, при вычислении которого вы получаете результат.
- Команда не возвращает результат.
- Объявление вводит в программу новую переменную, константу, функцию, структуру, класс или перечисление.
- Операторы — знаки или последовательности символов, используемые для изменения, проверки или объединения значений в Swift.
- Некоторые часто используемые операторы выполняют математические вычисления.
- Другие операторы позволяют выполнять присваивание, сравнивать значения или проверять результат операций.

Кто и что делает?

Подкрепите свои знания базовых операторов, используемых в выражениях Swift, и соедините каждое выражение в левой части с его результатом в правой части.

`7 + 3`	`2`
`9 - 1`	`10`
`10 / 5`	`20`
`5 * 4`	`8`
`9 % 3`	`0`
`-11`	`-11`

Ответы на с. 65.



Если в выражении используются только целые числа, то и результат также будет целым числом.

Если в любой точке выражения используются дробные числа, то и результат будет дробным числом.

Предположим, вы хотите разделить 10 на 6. Казалось бы, при выполнении следующей команды результат должен быть равен приблизительно `1,666666666666667`:

```
10 / 6
```

Тем не менее результат будет равен `1`, потому что результат может быть только целым числом, а дробь округляется до ближайшего целого значения.

С другой стороны, если вы прикажете Swift использовать дробные вычисления:

```
10.0 / 6.0
```

получится более точный ответ `1.666666666666667`, потому что Swift в этом случае работает исключительно с дробными числами.

В Swift дробные числа также часто обозначаются своими типами double и float.

Возможно, вы также слышали о других типах — строках и логических значениях.

В Swift также поддерживается еще один математический оператор: *остаток*. С его помощью можно узнать остаток от деления одного числа на другое.

Например, если вы хотите узнать, какой остаток будет получен от деления 9 на 2, выполните следующее выражение:

```
9 % 2
```

Результат равен `1`. Дело в том, что 9 нацело не делится на 2; при делении мы получаем 4 с остатком 1.

Имена и типы

С данными, как и с домашними питомцами и сезонными напитками, намного удобнее работать по имени. **Константы** и **переменные** используются для хранения данных и обращения к ним по имени. Для присваивания им данных используется **оператор присваивания** =.

Переменная представляет собой именованные данные, *значение которых может изменяться*, тогда как **константа** представляет собой данные, *значение которых изменяться не может*. Несколько примеров:

Ключевое слово let определяет константу, а ключевое слово var определяет переменную.

Знак = также является оператором. Он называется оператором присваивания.

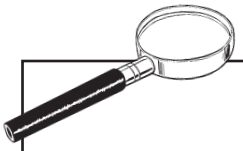
```
let favNumber = 8.7
```

Эта команда объявляет константу с именем favNumber и присваивает ей дробное значение 8.7.

```
var coffeesConsumed = 17
coffeesConsumed = 25
```

Эта команда объявляет переменную с именем coffeesConsumed и присваивает ей целое значение 17.

Переменную можно изменить позднее. Поэтому она и называется переменной. Понимаете?



Переменные и константы под увеличительным стеклом

Объявление переменной

```
var favNumber = 8.7
```

Переменной необходимо назначить имя.

Переменная имеет тип double, потому что ей присвоено число с дробной частью.

Необходимо показать, что это именно переменная, а не что-то другое, поэтому мы используем ключевое слово var.

Это оператор присваивания. Он присваивает значение, указанное в правой части, переменной в левой части.

Объявление константы

```
let goodNumber = 92
```

Константа является целым числом, потому что ей присвоено целое число.

Это константа, потому что в объявлении использовано ключевое слово let.

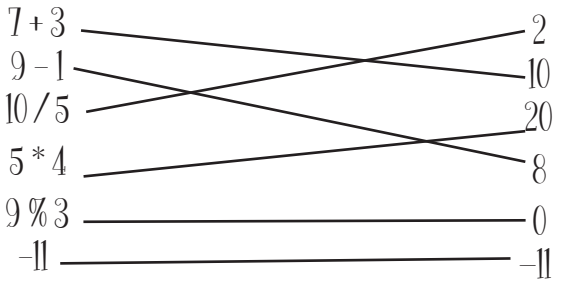
Константе назначено имя.

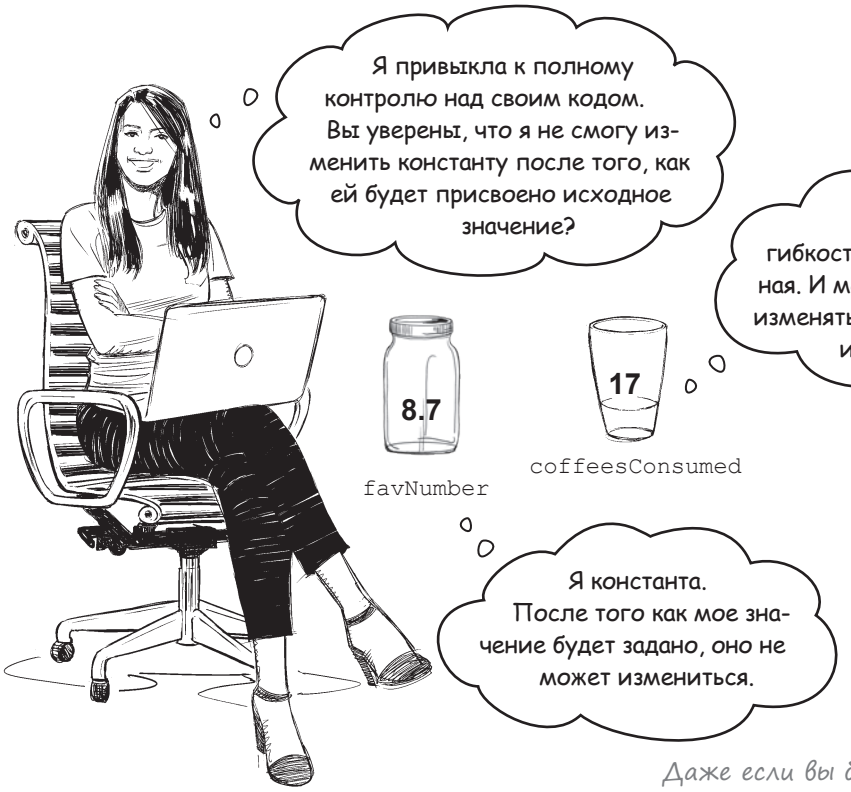
Упражнение Решение

С. 60

Кто и что делает? Решение

С. 62





Значение переменной может измениться, но ее тип остается неизменным.

Даже если вы большой начальник!

Константу невозможно изменить после того, как ей будет присвоено значение. Никогда.

Значение **переменной** можно изменить столько раз, сколько потребуется. Но после того как вы присвоите значение **константе,** изменить его уже не удастся. Давайте повнимательнее присмотримся к тому, как объявляются переменные и константы:

```
let myNumber = 10
```
Определяет константу с именем myNumber и присваивает ей значение 10. Swift понимает, что константа является целочисленной, потому что мы присвоили ей целое число при инициализации.

```
let myDouble = 10.5
```
Выражение создает константу с именем myDouble и сохраняет в ней дробное число 10.5 (double).

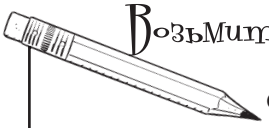
```
var number = 55
```
Выражение объявляет переменную с именем number, которой присваивается целое число 55. Переменная становится целочисленной.

```
number = 9.7
```
Так как переменная number является целочисленной, ей невозможно присвоить double (хотя это и переменная). Попытка присваивания приведет к ошибке.

```
number = 10
```
Впрочем, number можно присвоить другое целое число. Потому что это переменная.

```
let myConstant = 5
```
Создает константу с именем myConstant и присваивает ей целое число 5, поэтому константа становится целочисленной.

```
myConstant = 7
```
После присваивания исходного значения константу нельзя изменить. А значит, эта команда невозможна. Она приведет к ошибке.



Возьмите в руку карандаш

Создайте новую среду Swift Playground и напишите код Swift, который делает следующее:

- ☐ Создает переменную с именем pizzaSlicesRemaining и присваивает ей значение 8.
- ☐ Создает константу с именем totalSlices и присваивает ей 8.
- ☐ Делит totalSlices на pizzaSlicesRemaining.
- ☐ Обновляет pizzaSlicesRemaining значением 4.
- ☐ Снова делит totalSlices на pizzaSlicesRemaining.

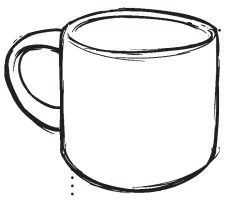
→ Ответы на с. 71.

Упражнение

Создайте новую среду Playground и объявите переменную с типом Integer, присвойте ей некоторые данные (целое число, естественно). Затем в следующей строке попробуйте присвоить строку и посмотрите, что произойдет.

Какую ошибку выдает Swift?

→ Ответ на с. 71.



Расслабьтесь

Сколько всего нового...

Путешествие в страну Swift только начинается, а вам уже приходится запоминать множество элементов программ. Ваши знания будут расширяться, и со временем вы начнете себя чувствовать более уверенно.