# Добро пожаловать в мир функционального мышления

# 1

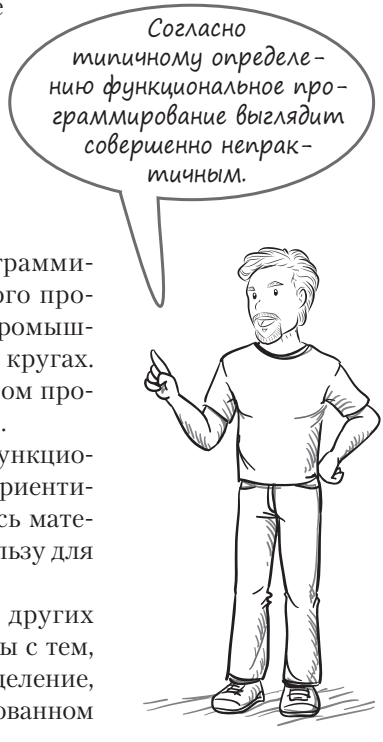
**В этой главе вы**

- ✓ Узнаете определение функционального мышления.
- ✓ Поймете, чем эта книга отличается от других книг по функциональному программированию.
- ✓ Познакомитесь с главной отличительной особенностью кода с точки зрения функциональных программистов.
- ✓ Решите, насколько эта книга подходит лично вам.

Мы начнем с того, что определим функциональное мышление и расскажем, как его главная отличительная особенность помогает программистам-практикам строить более качественные продукты. Кроме того, будет приведен обзор предстоящего путешествия в контексте двух главных идей, осознанных функциональными программистами.

## Что такое функциональное программирование

Программисты постоянно спрашивают меня, что такое функциональное программирование (ФП) и для чего его лучше использовать. Мне трудно объяснить, для чего лучше подойдет ФП, потому что это парадигма общего назначения. Оно хорошо подходит для всего. А о том, в каких областях оно раскрывается по-настоящему, вы узнаете через несколько страниц.




Дать определение функциональному программированию тоже непросто. Мир функционального программирования огромен. Оно применяется в промышленном программировании и в академических кругах. Впрочем, большая часть книг о функциональном программировании написана именно теоретиками.

Эта книга отличается от типичных книг о функциональном программировании. Она безусловно ориентирована на промышленное применение ФП. Весь материал книги должен приносить практическую пользу для профессиональных программистов.

Возможно, вам встретятся определения из других источников, и важно понимать, как они связаны с тем, что мы обсуждаем в этой книге. Типичное определение, приведенное в Википедии, в слегка перефразированном виде выглядит так:

**функциональное программирование** (ФП), *сущ.*

1. Парадигма программирования, для которой характерно использование математических функций и предотвращение побочных эффектов.
2. Стиль программирования, использующий только чистые функции без побочных эффектов.

*Подчеркнутым терминам необходимо дать определение*

Разберем подчеркнутые термины.

К *побочным эффектам* относится все, что делает функция помимо возвращения значения: например, отправка электронной почты или изменение глобального состояния. Побочные эффекты могут создавать проблемы, потому что они происходят при каждом вызове вашей функции. Если вас интересует только возвращаемое значение, а не побочные эффекты, значит, в программе происходит что-то непреднамеренное.

Функциональные программисты обычно стараются избегать побочных эффектов, которые не являются абсолютно необходимыми.

*Чистые функции* — функции, которые зависят только от своих аргументов и не имеют побочных эффектов. С одними и теми же аргументами они всегда выдают одно возвращаемое значение. Можно называть их *математическими функциями*, чтобы отличить от функций как элемента в программировании. Функциональные программисты уделяют особое внимание использованию чистых функций, потому что они более просты для понимания и управления.

Определение подразумевает, что функциональные программисты полностью избегают побочных эффектов и используют только чистые функции. Тем не менее это не так. Функциональные программисты, работающие над реальными программами, используют побочные эффекты и функции с побочными эффектами.

*Обычно программы запускаются как раз ради этого!*

**Типичные побочные эффекты**

1. Отправка электронной почты.
2. Чтение файла.
3. Переключение светового индикатора.
4. Отправка веб-запроса.
5. Включение тормоза в машине.

## Недостатки определения при практическом применении

Такое определение хорошо подойдет для академических кругов, но у него есть ряд недостатков с точки зрения программиста-практика. Еще раз присмотримся к определению:

**функциональное программирование** (ФП), *сущ.*

1. Парадигма программирования, для которой характерно использование математических функций и предотвращение побочных эффектов.
2. Стиль программирования, использующий только чистые функции без побочных эффектов.

Для наших целей такое определение создает три основные проблемы.

#### Проблема 1: ФП необходимы побочные эффекты

В определении сказано, что ФП стремится избегать побочных эффектов, но программы обычно запускаются как раз ради побочных эффектов. Какой прок от почтового клиента, который не отправляет электронную почту? Определение подразумевает, что мы должны полностью избегать их, тогда как на практике побочные эффекты используются там, где это необходимо.

#### Проблема 2: ФП неплохо справляется с побочными эффектами

Функциональные программисты знают, что побочные эффекты необходимы, хоть и проблематичны, поэтому существует значительное количество инструментов для работы с ними. Определение подразумевает, что мы используем только чистые функции. Напротив, мы достаточно часто применяем функции с побочными эффектами. Существует великое множество функциональных методов, которые упрощают их использование.

**Загляни в словарь**

*Побочные эффекты* — это любое поведение функции, кроме возврата значения.

*Чистые функции* зависят только от своих аргументов и не имеют побочных эффектов.

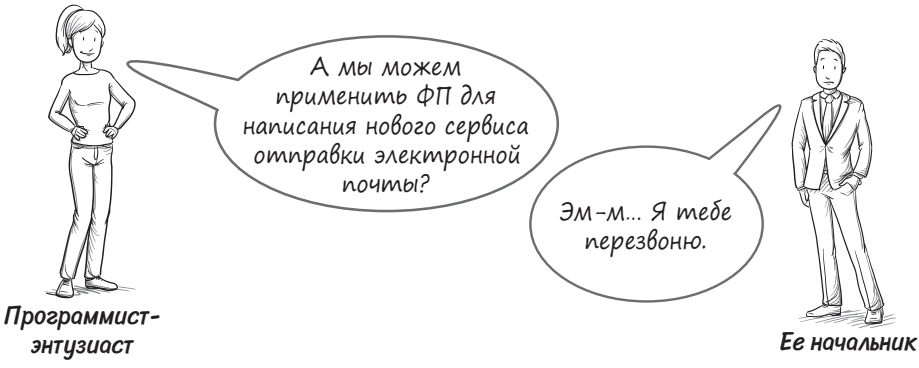
#### Проблема 3: Практичность ФП

Из определения может сложиться впечатление, что ФП имеет в основном математическую природу и для реальных программ оно непрактично. Тем не менее многие важные программные системы написаны с применением функционального программирования.

Определение особенно сильно сбивает с толку людей, которые знакомятся с ФП именно по нему. Представьте руководителя, действующего из лучших побуждений, который прочитал определение в Википедии.

## Определение ФП сбивает с толку руководителей

Представьте, что Дженна, программист-энтузиаст, хочет использовать ФП для сервиса отправки электронной почты. Она знает, что ФП поможет спроектировать систему для улучшения общей надежности. Ее начальник не знает, что такое ФП, поэтому он находит определение в Википедии.



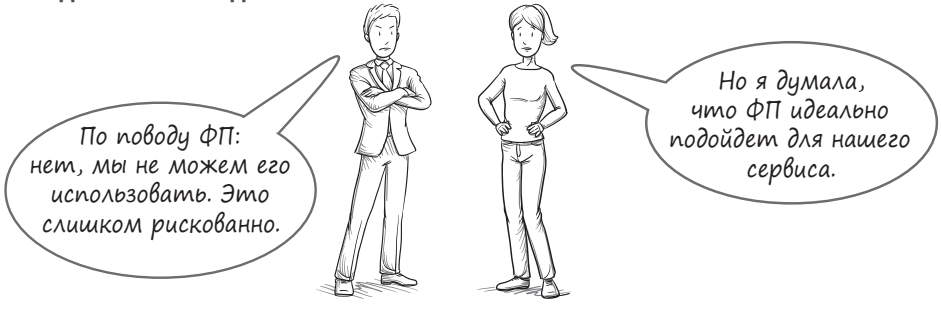

**Начальник ищет «функциональное программирование» в Википедии:**

*. . . предотвращение побочных эффектов . . .*

**Он находит «побочный эффект» в Google. Типичные побочные эффекты:**

- *отправка электронной почты*
- . . .


**Позднее в тот же день...**



## Функциональное программирование рассматривается как совокупность навыков и концепций

Мы не будем использовать типичное определение в этой книге. ФП каждый понимает по-своему, это огромная область для научных исследований и практики.

Я говорил со многими функциональными программистами относительно того, какие свойства ФП кажутся им наиболее полезными. Книга «Грокаем функциональное мышление» стала квинтэссенцией навыков, логических умозаключений и перспектив функциональных программистов-практиков. В книгу были включены только самые практичные и эффективные идеи.

В книге вы не найдете данных последних исследований или эзотерических идей. Мы будем изучать только навыки и концепции, применяемые в наши дни. В ходе исследований я обнаружил, что самые важные концепции ФП применимы даже в объектно-ориентированном и процедурном коде, а также в разных языках программирования. Красота ФП по-настоящему проявляется в том, что ФП имеет дело с полезными, универсальными практиками программирования.

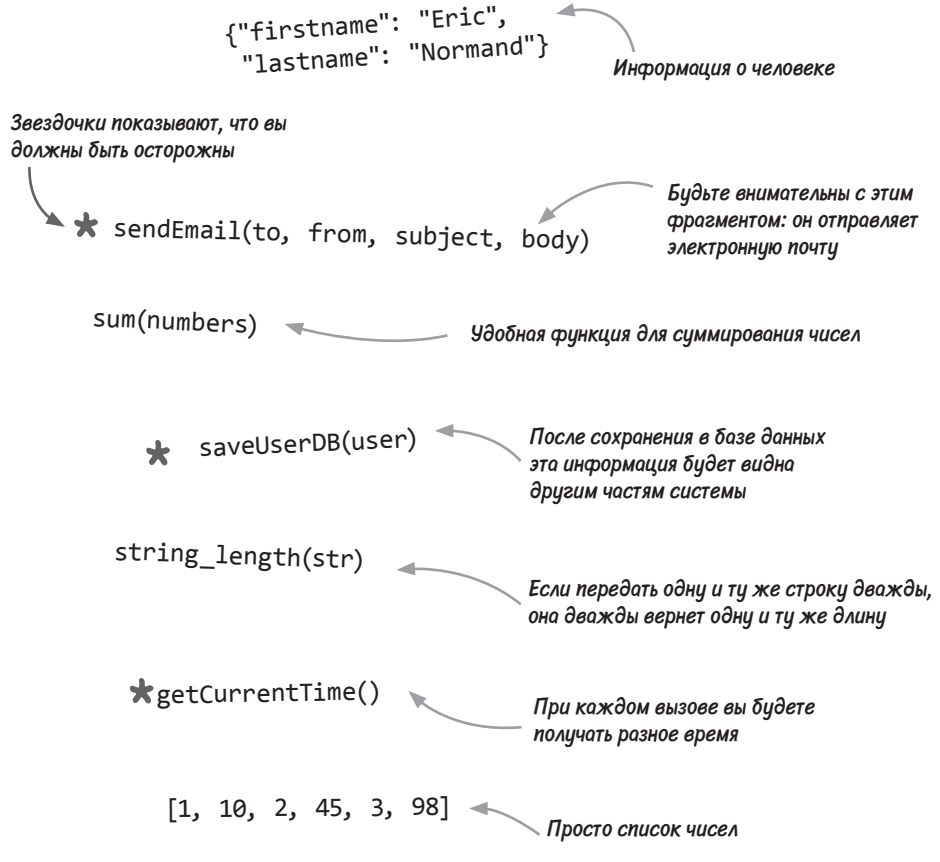
Взглянем на навык, важность которого не будет отрицать ни один функциональный программист: разграничение действий, вычислений и данных.

## Действия, вычисления и данные

Когда функциональный программист смотрит на код, он немедленно классифицирует его на три категории:

1. Действия (actions — A).
2. Вычисления (calculations — C).
3. Данные (data — D).

Рассмотрим несколько примеров кода из существующей базы данных. Будьте особенно внимательны с фрагментами, помеченными звездочкой.

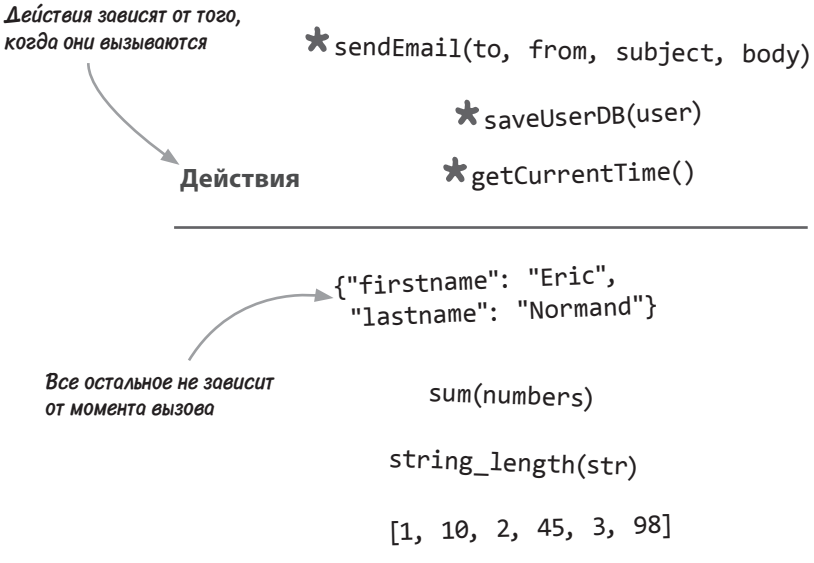



Функции со звездочкой требуют особого внимания, потому что они зависят от того, когда или сколько раз они вызываются. Например, важная электронная почта не должна отправляться дважды или отправляться ноль раз.

Фрагменты, помеченные звездочкой, относятся к *действиям*. Отделим их от остальных.

## Функциональные программисты особо выделяют код, для которого важен момент вызова

Проведем линию и переместим все функции, зависящие от момента вызова, по одну сторону от линии:




Это очень важный момент. Действия (все, что находится над линией) зависят от того, когда или сколько раз они вызываются. Они требуют особого внимания.

Однако с тем, что находится под линией, работать намного проще. Неважно, когда вы вызовете функцию `sum`, — она будет каждый раз вызывать правильный ответ. Неважно и то, сколько раз вы ее вызовете. Она не повлияет на остальные части программы или окружающий мир за пределами программы.

Однако существует еще одно различие: одна часть кода может выполняться, другая остается инертной. Проведем еще одну линию на следующей странице.