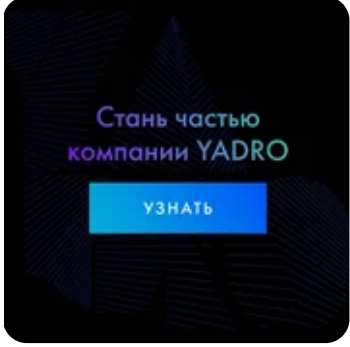


Реализация программ в функциональном стиле



oneweekoffer.yadro.com

РЕКЛАМА

Ищем ведущих программистов в команду YADRO.

Ждем амбициозных и талантливых, которые горят инновационными идеями
Заполни анкету

Узнать больше

[Справочник по языку Python3.](#) / Реализация программ в функциональном стиле

В этом разделе объясняется основная [концепция функционального программирования](#), если просто интересно узнать о возможностях языка Python, то перейдите сразу к "[Итератор как основа функционального стиля Python](#)".

- [Деление задач языками программирования](#);
- [Математическое доказательство правильности программы](#);
- [Модульность](#);
- [Простота отладки и тестирования](#);
- [Сочетаемость](#);
- [Пример кода в функциональном стиле](#);

Деление задач языками программирования.

- Большинство языков программирования являются процедурными: программы представляют собой списки операторов, которые сообщают компьютеру, что делать с вводом программы. C, Pascal, а так же оболочка Unix - это процедурные языки.
- На декларативных языках пишется спецификация, которая описывает проблему, которую необходимо решить, а реализация языка выясняет, как эффективно выполнять вычисления. SQL - это декларативный язык. SQL-запрос описывает набор данных, который необходимо получить, а механизм SQL решает, сканировать ли таблицы или использовать индексы, какие условия следует выполнить в первую очередь и т. д.
- Объектно-ориентированные программы управляют коллекциями объектов. У объектов есть внутреннее состояние и методы, которые тем или иным образом запрашивают или изменяют внутреннее состояние. Smalltalk и Java - объектно-ориентированные языки. C++ и Python - это языки, которые поддерживают объектно-ориентированное программирование, но не заставляют его использовать.
- Функциональное программирование разбивает задачу на набор функций. В идеале функции принимают только какой то ввод и производят вывод и не имеют какого либо внутреннего состояния, которое влияет на вывод, создаваемый для данного ввода. Хорошо известные функциональные языки включают семейство ML (Standard ML, OCaml и другие варианты) и Haskell.

Разработчики некоторых языков программирования предпочитают подчеркивать один конкретный подход к программированию. Это затрудняет написание программ, при использовании другого подхода. Lisp, C++ и Python - языки с несколькими парадигмами, они поддерживают несколько различных подходов к написанию программ. В большой программе, модули могут быть написаны с использованием разных подходов. Например, графический интерфейс пользователя может быть объектно-ориентированным, а логика обработки - процедурной или функциональной.

В функциональной программе ввод данных проходит через цепочки функций. Цепочки функций могут создаваться с целью создать новую функцию. Каждая функция работает со своим вводом и производит некоторый вывод. Функциональный стиль не поощряет функции с эффектами, которые изменяют внутреннее состояние или вносят другие изменения, которые не видны в возвращаемом значении функции. Отказ от изменения внутреннего состояния, означает отказ от использования структур данных, которые обновляются по мере выполнения программы. В функциональном программировании - вывод каждой функции должен зависеть только от её ввода.

Стиль функционального программирования на Python, предоставят функционально выглядящий интерфейс, но будет использовать нефункциональные подходы. Например, реализация функции по-прежнему будет использовать присвоения локальным переменным, но не будет изменять глобальные переменные или иметь другие побочные эффекты.

Функциональное программирование можно рассматривать как противоположность объектно-ориентированного подхода. Объекты - это маленькие капсулы, содержащие некоторое внутреннее состояние вместе с набором методов, которые могут изменять это состояние, а программы, в свою очередь состоят из выполнения правильного набора изменений состояния. Функциональное программирование стремится максимально избегать изменений состояния и работает с данными, передаваемыми между функциями. В Python можно объединить два подхода, написав функции, которые будут принимать и возвращать экземпляры объектов приложения (сообщения электронной почты, транзакции и т. д.).

Функциональный дизайн может показаться странным и ограниченным. У функционального стиля есть теоретические и практические преимущества:

- Математическое доказательство правильности программы.
- Модульность.
- Сочетаемость.
- Лёгкость отладки и тестирования.

Математическое доказательство правильности программы.

В течение долгого времени исследователи были заинтересованы в поиске способов математического доказательства правильности программ. Цель - строгое доказательство того, что программа предоставляет правильный результат для всех возможных входных данных. Это отличается от тестирования программы на множестве входных данных и вывода о том, что её результат обычно правильный, или чтения исходного кода программы и заключения о том, что код выглядит правильно.

Метод, используемый для подтверждения правильности программ, заключается в записи инвариантов, свойств входных данных и переменных программы, которые всегда верны. К сожалению, доказывать правильность программ в значительной степени непрактично и не относится к программному обеспечению Python. Даже тривиальные программы требуют доказательств объемом в несколько страниц, при этом если доказательство создано, то возникнет вопрос о проверке доказательства. Возможно, в доказательстве есть ошибка, и ошибочно полагать, что программа работает правильно.

Модульность.

Более практическое преимущество функционального программирования заключается в том, что оно заставляет разбивать проблему на мелкие части. В результате, программы становятся более модульными. Легче указать и написать небольшую функцию, которая выполняет одно действие, чем большую функцию, выполняющую сложное преобразование. Небольшие функции также легче читать и проверять на наличие ошибок.

Простота отладки и тестирования.

Тестирование и отладка программы в функциональном стиле проще.

Отладка упрощена, поскольку функции обычно небольшие и четко определены. Когда программа не работает, каждая функция является точкой интерфейса, где можно проверить правильность данных. Так же можно просмотреть промежуточные входы и выходы, чтобы определить функцию, которая вызывает ошибку.

Тестировать проще, потому что каждая функция является потенциальным объектом для модульного тестирования. Функции не зависят от состояния системы, которое необходимо воспроизвести перед запуском теста. Нужно только синтезировать правильный ввод, а затем проверить, что результат соответствует ожиданиям.

Сочетаемость.

Работая над программой в функциональном стиле, обычно пишут ряд функций с различными входами и выходами. Некоторые из этих функций неизбежно будут специализированы для конкретного приложения, но другие будут полезны в других программах. Например, функция, которая принимает путь к каталогу и возвращает все файлы, или функция, которая принимает имя файла и возвращает его содержимое, может применяться во многих различных ситуациях.

Со временем, формируется личная библиотека утилит. И в последствии, новые программы будут собираться из существующих функций, при этом, дополнительно нужно будет написать несколько функций, специально предназначенных для текущей задачи.

Пример кода в функциональном стиле.

В примерах используются функции, встроенные в интерпретатор или стандартную библиотеку Python. Функциональный стиль программирования так же предполагает использования функций, написанных пользователем.

Что бы избежать случайного изменения состояния или каких либо побочных эффектов, функциональный стиль предполагает создание цепочек из функций. Другими словами, одни функций передаются в другие функции до получения промежуточного или конечного результата. Цепочки функций могут создаваться с целью породить новые функции.

Ниже представлен яркий пример цепочек функций:

```
>>> rez = [x**2>=50 and x or -x for x in range(10) if x%2==0]
>>> print(rez)
# [0, -2, -4, -6, 8]

# или
>>> import random
>>> n = 10
>>> rez = '\n'.join([' '*(n-i)+'/'+''.join(random.choice(' # *') for _ in range(2*i))+'\\"' for i in range(n)])
>>> print(rez)
#           /\
#          /  \
#         /* *  \
#        /      \
#       / #      \
#      /*  #  #  * \
#     / *  ##**   * \
#    /*##**##**   * \
#   / ##** * * *   * \
#  /#  ## *  #    * * \
```

Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Итератор как основа функционального стиля Python](#)
- [Типы данных Python, поддерживающие итераторы](#)
- [Выражения-генераторы Python в функциональном программировании](#)
- [Функции Python, используемые в функциональном программировании](#)

ХОЧУ ПОМОЧЬ
ПРОЕКТУ

Ещё!
Распродажа
до 80%



12+

