БИБЛИОТЕКА ПРОГРАММИСТА

Дэн Бейдер

ЧИСТЫЙ PYTHON

ТОНКОСТИ ПРОГРАММИРОВАНИЯ ДЛЯ ПРОФИ

ПИТЕР®

Python Tricks: The Book

Dan Bader

ББК 32.973.2-018.1
УДК 004.43
Б41

Бейдер Д.

Б41 Чистый Python. Тонкости программирования для профи. — СПб.: Питер, 2024. — 288 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-0803-9

Изучение всех возможностей Python — сложная задача, а с этой книгой вы сможете сосредоточиться на практических навыках, которые действительно важны. Раскопайте «скрытое золото» в стандартной библиотеке Python и начните писать чистый код уже сегодня.

Если у вас есть опыт работы со старыми версиями Python, вы сможете ускорить работу с современными шаблонами и функциями, представленными на Python 3.

Если вы работали с другими языками программирования и хотите перейти на Python, то найдете практические советы, необходимые для того, чтобы стать эффективным питонистом.

Если вы хотите научиться писать чистый код, то найдете здесь самые интересные примеры и малоизвестные трюки.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.1
УДК 004.43

ISBN 978-1775093305 англ.
ISBN 978-5-4461-0803-9

Оглавление

Глава 5. Общие структуры данных Python 153

Что питонисты говорят о книге «Чистый Python. Тонкости программирования для профи»

«Мне эта книга безумно нравится. Она похожа на репетитора, который разъясняет... ну, типа, всякие трюки или идиомы. Я изучаю Python на работе и перешел на него с оболочки командной строки Powershell, с которой я познакомился там же, — масса нового и фантастического материала. Всякий раз, когда я попадаю в тупик с Python (обычно с шаблонным кодом Flask) или когда чувствую, что мой исходный код мог бы выглядеть как-то более по-питоновски, я направляю вопросы в нашу внутреннюю дискуссионную группу Python.

Я часто восхищаюсь некоторыми ответами, которые дают мне коллеги. Их отзывы пестрят терминами типа «включения в словари», «лямбды» и «генераторы». Я всегда впечатлен и даже поражен тем, насколько же Python мощный, когда вы владеете этими приемами и можете их правильно реализовать.

Ваша книга стала именно той, которая нужна, чтобы превратиться из запутавшегося скриптера Powershell в того, кто знает, как и когда применять эти питоновские «штучки», о которых все говорят.

Как человеку, у которого нет ученой степени в области Computer Science, мне приятно иметь учебное пособие, объясняющее вещи, которые другие, возможно, узнали, получая академическое образование. Для меня огромное удовольствие читать эту книгу. И кроме того, я также подписан на электронную рассылку, что и помогло мне выйти на это издание».

— **Даниэль Мейер** (Daniel Meyer),
старший администратор локальных систем в Tesla Inc.

«Впервые о вашей книге я услышал от коллеги, который хотел запутать меня примером вашего кода с построением словаря. Я был почти на сто процентов уверен, почему словарь в итоге получился и более простой, и меньшего размера, но, должен признаться, что такого результата я не ожидал. :)

Он показал мне книгу по видеосвязи — и я как бы просмотрел ее, когда он пролистывал мне страницы. Я сразу же загорелся узнать больше.

Уже к полудню я купил собственный экземпляр книги и продолжил читать о том, как в Python создаются словари. Спустя несколько часов, когда я встретил другого коллегу за чашкой кофе, то использовал похожий трюк уже с ним. :)

Затем коллега поднял вопрос по той же самой теме, и благодаря тому, как вы все объяснили в книге, я мог не теряться в догадках, а верно ответить, каким будет результат. Это значит, что вы прекрасно объяснили материал. :)

В Python я не новичок, и некоторые из концепций в отдельных главах для меня тоже не новы, но, признаюсь, читая книгу, я все время узнаю что-нибудь новое из каждой главы, поэтому респект за написание очень приличной книги и за фантастическую работу по объяснению принципов, лежащих в основе всех тонкостей! С большим нетерпением жду обновлений и, безусловно, познакомлю своих друзей и коллег с вашей книгой».

— **Ог Масьел** (Og Maciel),
разработчик на Python в Red Hat

«С великим удовольствием читал книгу Дэна. Он раскрывает важные аспекты Python на ясных примерах (в одном из них используя кошек-близнецов, чтобы объяснить операторы `is` *и* `==`*).*

Речь не просто о примерах исходного кода. В издании всесторонне обсуждаются соответствующие детали реализации. По-настоящему важно то, что эта книга реально позволяет вам писать программный код на Python лучше!

Благодаря этой книге я вдохновился новейшими практическими приемами программирования на Python: например, стал использовать собственные исключения и абстрактные классы (когда искал их, нашел и блог Дэна). Одни только эти новые знания оправдывают цену книги».

— **Боб Бельдербос** (Bob Belderbos),
инженер Oracle и соучредитель PyBites

Предисловие

Прошло почти десять лет с тех пор, как я впервые познакомилась с языком программирования Python. Когда много лет назад я впервые попробовала заняться им, то, признаюсь, начала с неохотой. До того я программировала на другом языке, и совсем неожиданно на работе меня определили в ту команду, где все использовали Python. Это стало началом моего собственного путешествия по миру Python.

Когда меня впервые познакомили с языком Python, то сказали, что все будет легко и я освою его очень быстро. Когда же я спросила коллег о ресурсах по изучению Python, мне дали всего одну-единственную ссылку на официальную документацию. Чтение ее поначалу сбивало с толку, и ушло достаточно много времени, прежде чем я научилась уверенно в ней ориентироваться, не говоря уже о том, чтобы разбираться. Нередко мне приходилось искать решения на веб-сайте StackOverflow.

Придя из другого языка программирования, я не просто нуждалась в каком-нибудь источнике, посвященном обучению программированию или дающем пояснения по поводу классов и объектов. Я искала конкретные ресурсы, которые научили бы меня функциональным средствам языка Python, объяснили разницу между ним и другими языками и то, как написание исходного кода на Python отличается от написания его на другом языке.

Я потратила немало лет, чтобы полностью осознать ценность этого языка. Читая книгу Дэна, я досадовала, что у меня не было ее тогда, когда много лет назад я только начала изучать Python.

Например, одним из многих уникальных функциональных средств языка Python, которое поначалу меня удивило больше всего, была конструкция включения в список. Как Дэн отмечает в своей книге, обычной реакцией тех, кто только перешел на Python с другого языка, становятся слова «Так вот как они используют циклы `for`!». Помню один из первых комментариев с обзором исходного кода, который я получила, когда начинала

программировать на Python: «Почему бы здесь не применить включение в список?» Дэн четко разъясняет это понятие в главе 6, начиная с показа организации цикла в чисто питоновском стиле и постепенно достраивая его до итераторов и генераторов.

В разделе 2.5 Дэн рассматривает различные способы форматирования строковых значений в Python. Форматирование строковых значений — это одна из тех вещей, которые бросают вызов Дзену языка Python, гласящему, что должен существовать один и желательно только один очевидный способ сделать это. Дэн показывает разные способы, в том числе мое любимое новое дополнение к языку, `f`-строки, а также объясняет плюсы и минусы каждого метода.

Глава «*Питоновские методы повышения производительности*» представляет собой еще один великолепный ресурс. Она охватывает аспекты, лежащие за пределами языка программирования Python, а также содержит советы о том, как отлаживать свои программы, как управлять библиотеками, от которых они зависят, и дает вам возможность заглянуть внутрь байткода Python.

Для меня большая честь и удовольствие представить книгу «*Чистый Python. Тонкости программирования для профи*» моего друга Дэна Бейдера.

Участвуя в развитии языка Python в качестве разработчика ядра CPython, я общаюсь со многими членам сообщества. На своем пути я встретила много наставников, помощников и завела много новых друзей. Они напоминают мне о том, что Python — это не только исходный код, но прежде всего — сообщество.

Чтобы освоить программирование на Python, нужно не только понимать теоретические аспекты языка. Для достижения этой цели придется понять и принять общие правила и самые лучшие практические приемы, используемые сообществом.

Книга Дэна поможет вам в этом путешествии. Я убеждена, что, прочитав ее, вы почувствуете себя увереннее в написании программ на Python.

— **Мариатта Виджайя** (Mariatta Wijaya),
разработчик ядра Python (mariatta.ca)

Комментарии переводчика

Весь материал настоящей книги протестирован в среде Windows 10. При тестировании исходного кода за основу взят Python версии 3.6.4 (время перевода — апрель 2018 года).

Хотя в настоящей книге установка и применение сторонних библиотек практически не рассматривается, тем не менее в комментарии переводчика включена информация о базовом наборе инструментов, необходимых для дальнейшей работы. Эта информация ни к чему не обязывает, но служит прекрасной отправной точкой для всех, кто интересуется программированием на Python.

Базовый набор библиотек для разработчика

В обычных условиях библиотеки Python можно скачать и установить из каталога библиотек Python `PyPi` (https://pypi.python.org/) с помощью менеджера пакетов `pip`. Однако следует учесть, что в ОС Windows для работы некоторых библиотек, в частности SciPy, Scikit-learn и Scikit-image, требуется, чтобы в системе была установлена библиотека Numpy+MKL. Библиотека **Numpy+MKL** привязана к библиотеке Intel® Math Kernel Library и включает в свой состав необходимые динамические библиотеки (DLL) в каталоге numpy.core. Библиотеку Numpy+MKL следует скачать из хранилища whl-файлов на веб-странице Кристофа Голька из лаборатории динамики флуоресценции Калифорнийского университета в г. Ирвайн (http://www.lfd.uci.edu/~gohlke/pythonlibs/) и установить при помощи менеджера пакетов `pip` как `whl` (соответствующая процедура установки пакетов в формате `whl` описана ниже). Например, для 64-разрядной операционной системы Windows и среды Python 3.6 команда будет такой:

```
pip install numpy-1.14.2+mkl-cp36-cp36m-win_amd64.whl
```

Такой режим установки также касается библиотек `scipy`, `scikit-image` и `scikit-learn`. Стоит также отметить, что эти особенности установки не относятся к ОС Linux и Mac. Далее приводятся сведения об основополагающих библиотеках.

- **NumPy** — основополагающая библиотека, необходимая для научных вычислений на Python.
- **SciPy** — библиотека, используемая в математике, естественных науках и инженерном деле. Требует наличия `numpy+mkl`.
- **Matplotlib** — библиотека для работы с двумерными графиками.
- **Pandas** — инструмент для анализа структурных данных и временных рядов. Требует наличия `numpy` и некоторых других. Для чтения файлов Excel требует установки библиотеки `xlrd`.
- **Scikit-learn** — интегратор классических алгоритмов машинного обучения. Требует наличия `numpy+mkl`.
- **Scikit-image** — коллекция алгоритмов для обработки изображений. Требует наличия `numpy+mkl`.
- **Jupyter** — интерактивная онлайновая вычислительная среда.
- **PyQt5** — библиотека инструментов для программирования графического интерфейса пользователя, требуется для работы инструментальной среды программирования `Spyder`.
- **Spyder** — инструментальная среда программирования на Python.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

1 Введение

1.1. Что такое идиома Python

> **Идиома Python** (Python Trick) — короткий фрагмент исходного кода на Python, используемый как инструмент обучения. Идиома Python обучает отдельному свойству языка Python путем простой иллюстрации либо служит в качестве мотивирующего примера, который дает возможность копнуть глубже и развить интуитивное понимание.

Книга «Чистый Python. Тонкости программирования для профи» началась как серия скриншотов с фрагментами кода, которыми я делился в Твиттере в течение одной недели. К моему удивлению, они получили отклики, а потом еще несколько дней подряд их продолжали распространять и ретвитить.

Разработчики все чаще и чаще стали обращаться ко мне с вопросом, как «получить всю серию». На самом деле у меня на очереди было еще лишь несколько таких идиом, которые охватывали целый ряд тем, связанных с Python. И за ними не было никакого плана. Они были просто забавным экспериментом в Твиттере.

Но из этих запросов я понял то, что мои краткие и четкие примеры кода стоит рассматривать и как инструмент для обучения. В конце концов я занялся созданием еще ряда идиом Python и начал ими делиться в серии почтовых рассылок. В течение нескольких дней на мою рассылку

подписались несколько сотен разработчиков на Python, и я был просто в восторге от этого.

В следующие дни и недели ко мне нескончаемым потоком стали обращаться разработчики на Python. Они благодарили за то, что я довел до них ту часть языка, которая оставалась для них камнем преткновения. Ощущение от этих отзывов было потрясающим. Я-то считал, что эти идиомы Python являлись простыми снимками экрана с примерами кода, но оказалось, что для многих людей они стали неоценимой помощью.

Именно тогда я решил удвоить ставку на моем эксперименте с идиомами Python и довел его до серии из порядка 30 электронных сообщений. Каждое из них представляло собой заголовок и снимок экрана с примером, и вскоре я осознал пределы этого формата. Примерно в ту же пору на мой электронной ящик пришло письмо от незрячего разработчика на Python, разочарованного тем, что эти идиомы поставлялись как изображения, которые он не мог прочитать с помощью экранного диктора.

Стала очевидной необходимость уделить этому проекту больше времени, чтобы сделать его привлекательнее и доступнее для более широкой аудитории. Так что я засел за воссоздание всей серии электронных сообщений с идиомами Python в текстовом формате и с надлежащей подсветкой синтаксиса на основе HTML-разметки. Переиздание моей книги о Python было встречено одобрительно. По откликам я понял, что разработчики обрадовались тому, что наконец смогли копипастить примеры кода и экспериментировать с ними.

По мере того как все больше разработчиков подписывалось на электронную рассылку этой серии, я начал замечать закономерности в откликах и вопросах, которые я получал. Некоторые идиомы хорошо работали именно в качестве мотивационных примеров. Однако что касается более сложных из них, то не хватало рассказчика, который направлял бы читателей или подсказывал им дополнительные ресурсы, где можно было бы расширить свое понимание.

Скажем так: этот аспект был еще одной большой областью для улучшения. Основная задача моего веб-сайта dbader.org состоит в том, чтобы помогать разработчикам на Python становиться еще более потрясающими, —

и очевидно, что этот аспект предоставлял возможность приблизиться к этой цели.

Я решил взять из своего почтового курса самые лучшие и самые ценные трюки и идиомы и на их основе приступил к написанию книги нового типа по Python:

- книги, которая обучает самым крутым аспектам языка с помощью коротких и легких для усвоения примеров;
- книги, в которой хранятся потрясающие функциональные средства языка Python (класс!) и которая поддерживает мотивацию на высоком уровне;
- книги, которая берет вас за руку, ведет по правильному пути и помогает углубить свое понимание языка Python.

> Для меня эта книга — результат моих любимых занятий и большой эксперимент. Надеюсь, что вы получите удовольствие от ее прочтения и по ходу узнаете еще что-то о Python!
>
> — Дэн Бейдер

1.2. Чем эта книга будет полезна

Цель этой книги в том, чтобы сделать вас лучшим — более эффективным, более осведомленным, более практичным — разработчиком на языке Python. Вы, вероятно, задаетесь вопросом: *а как чтение этой книги поможет мне всего этого достигнуть?*

«Чистый Python» — это не пошаговое учебное пособие по Python. И это не курс языка Python начального уровня. Если вы находитесь на начальных стадиях изучения этого языка, то в одиночку эта книга не превратит вас в профессионального разработчика на Python. Ее чтение, безусловно, окажет на вас благотворное влияние, но при этом вам обязательно нужно поработать с другими ресурсами, которые сформируют ваши основополагающие навыки программирования на Python.

Вы извлечете из этой книги максимальную пользу, если в той или иной степени владеете языком Python и хотите перейти на следующий уровень. Она прекрасно поможет, если вы уже некоторое время программируете на Python и готовы пойти дальше, чтобы придать своим познаниям законченный вид и сделать свой программный код более питоновским.

Чтение книги «*Чистый Python. Тонкости программирования для профи*» также окажет бесценную помощь, если у вас уже имеется опыт работы с другими языками программирования и вы надеетесь поскорее разобраться в тонкостях языка Python. Вы обнаружите массу практических советов и шаблонов проектирования, которые сделают вас более эффективным и квалифицированным программистом на Python.

1.3. Как читать эту книгу

Оптимальный способ чтения книги «*Чистый Python. Тонкости программирования для профи*» — рассматривать ее как копилку потрясающих функциональных средств языка Python. Каждая приводимая в книге идиома Python — самодостаточна, и поэтому ничего страшного, если вы будете обращаться к тем из них, которые вызывают у вас наибольший интерес. На самом деле именно это я вам и рекомендую делать.

Разумеется, вы также можете прочитать всю книгу «*Чистый Python. Тонкости программирования для профи*» от начала до конца. И когда вы дойдете до заключительной страницы, вы не пропустите ни одной идиомы и шаблона и будете знать, что ознакомились со всем.

Некоторые из этих идиом легко понять сразу, и вы не испытаете никаких затруднений при их внедрении в повседневную работу, просто прочитав главу. Чтобы разобраться в других идиомах, может потребоваться немного больше времени.

Если вы испытываете затруднения в том, чтобы та или иная идиома заработала в ваших программах, то, как правило, помогает возможность поэкспериментировать с каждым примером кода в сеансе интерпретатора Python.

Если и это не расставит все на свои места, то, пожалуйста, не стесняйтесь обращаться ко мне, чтобы я смог выручить вас и дать более подробное объяснение. В конечном счете это принесет пользу не только вам, но и всем питонистам, которые читают эту книгу.

1.4. Тонкости Python: цифровой комплект инструментов в качестве бонуса

Эта книга сопровождается коллекцией бонусных ресурсов, которые я называю «*Тонкости Python: цифровой комплект инструментов*»[1].

Среди всего прочего этот комплект инструментов включает 12 видеоуроков HD-качества общей продолжительностью более двух часов. Эти видеопособия тесно связаны с отдельными главами книги и помогут вам быстрее усвоить и закрепить знания, акцентировав внимание на ключевых моментах.

Включенные в этот комплект инструментов ресурсы стоят 100 $, но при покупке этой книги вы получаете их без всякой дополнительной оплаты.

Доступ к копии цифрового комплекта инструментов можно получить онлайн на моем веб-сайте по адресу dbader.org/python-tricks-toolkit.

[1] См. https://dbader.org/python-tricks-toolkit

2 Шаблоны для чистого Python

2.1. Прикрой свой з** инструкциями assert

Иногда по-настоящему полезное функциональное средство языка привлекает меньше внимания, чем оно того заслуживает. По некоторым причинам это именно то, что произошло со встроенной в Python инструкцией `assert`.

В этой главе я собираюсь дать вам представление об использовании `assert` в Python. Вы научитесь ее применять для автоматического обнаружения ошибок в программах Python. Эта инструкция сделает ваши программы надежнее и проще в отладке.

В этом месте вы, вероятно, заинтересуетесь: «Что такое `assert` и в чем ее прелесть?» Позвольте дать вам несколько ответов.

По своей сути инструкция Python `assert` представляет собой средство отладки, которое проверяет условие. Если условие утверждения `assert` *истинно*, то ничего не происходит и ваша программа продолжает выполняться как обычно. Но если же вычисление условия дает результат *ложно,* то вызывается исключение `AssertionError` с необязательным сообщением об ошибке.

Инструкция assert в Python — пример

Вот простой пример, чтобы дать вам понять, где утверждения `assert` могут пригодиться. Я попытался предоставить вам некоторое подобие реальной задачи, с которой вы можете столкнуться на практике в одной из своих программ.

Предположим, вы создаете интернет-магазин с помощью Python. Вы работаете над добавлением в систему функциональности скидочного купона, и в итоге вы пишете следующую функцию `apply_discount`:

```
def apply_discount(product, discount):
     price = int(product['цена'] * (1.0 – discount))
     assert 0 <= price <= product['цена']
     return price
```

Вы заметили, что здесь есть инструкция `assert`? Она будет гарантировать, что, независимо от обстоятельств, вычисляемые этой функцией сниженные цены не могут быть ниже 0 $ и они не могут быть выше первоначальной цены товара.

Давайте убедимся, что эта функция действительно работает как задумано, если вызвать ее, применив допустимую скидку. В этом примере товары в нашем магазине будут представлены в виде простых словарей. И скорее всего, в реальном приложении вы примените другую структуру данных, но эта безупречна для демонстрации утверждений `assert`. Давайте создадим пример товара — пару симпатичных туфель по цене 149,00 $:

```
>>> shoes = {'имя': 'Модные туфли', 'цена': 14900}
```

Кстати, заметили, как я избежал проблем с округлением денежной цены, использовав целое число для представления цены в центах? В целом неплохое решение... Но я отвлекся. Итак, если к этим туфлям мы применим 25 %-ную скидку, то ожидаемо придем к отпускной цене 111,75 $:

```
>>> apply_discount(shoes, 0.25)
11175
```

Отлично, функция сработала безупречно. Теперь давайте попробуем применить несколько недопустимых скидок. Например, 200 %-ную «скидку», которая вынудит нас отдать деньги покупателю:

```
>>> apply_discount(shoes, 2.0)
Traceback (most recent call last):
   File "<input>", line 1, in <module>
     apply_discount(prod, 2.0)
   File "<input>", line 4, in apply_discount
     assert 0 <= price <= product['price']
AssertionError
```

Как вы видите, когда мы пытаемся применить эту недопустимую скидку, наша программа останавливается с исключением `AssertionError`. Это происходит потому, что 200 %-ная скидка нарушила условие утверждения `assert`, которое мы поместили в функцию `apply_discount`.

Вы также можете видеть отчет об обратной трассировке этого исключения и то, как он указывает на точную строку исходного кода, содержащую вызвавшее сбой утверждение. Если во время проверки интернет-магазина вы (или другой разработчик в вашей команде) когда-нибудь столкнетесь с одной из таких ошибок, вы легко узнаете, что произошло, просто посмотрев на отчет об обратной трассировке исключения.

Это значительно ускорит отладку и в дальнейшем сделает ваши программы удобнее в поддержке. А в этом, дружище, как раз и заключается сила `assert`!

Почему просто не применить обычное исключение?

Теперь вы, вероятно, озадачитесь, почему в предыдущем примере я просто не применил инструкцию `if` и исключение.

Дело в том, что инструкция `assert` предназначена для того, чтобы сообщать разработчикам о *неустранимых* ошибках в программе. Инструкция `assert` *не* предназначена для того, чтобы сигнализировать об ожидаемых ошибочных условиях, таких как ошибка «Файл не найден», где пользо-

ватель может предпринять корректирующие действия или просто попробовать еще раз.

Инструкции призваны быть *внутренними самопроверками* (internal self-checks) вашей программы. Они работают путем объявления неких условий, возникновение которых в вашем исходном коде *невозможно*. Если одно из таких условий не сохраняется, то это означает, что в программе есть ошибка.

Если ваша программа бездефектна, то эти условия никогда не возникнут. Но если же они возникают, то программа завершится аварийно с исключением `AssertionError`, говорящим, какое именно «невозможное» условие было вызвано. Это намного упрощает отслеживание и исправление ошибок в ваших программах. А мне нравится все, что делает жизнь легче. Надеюсь, вам тоже.

А пока имейте в виду, что инструкция `assert` — это средство отладки, а не механизм обработки ошибок времени исполнения программы. Цель использования инструкции `assert` состоит в том, чтобы позволить разработчикам как можно скорее найти вероятную первопричину ошибки. Если в вашей программе ошибки нет, то исключение `AssertionError` никогда не должно возникнуть.

Давайте взглянем поближе на другие вещи, которые мы можем делать с инструкцией `assert`, а затем я покажу две распространенные ловушки, которые встречаются во время ее использования в реальных сценариях.

Синтаксис инструкции Python assert

Прежде чем вы начнете применять какое-то функциональное средство языка, всегда неплохо подробнее познакомиться с тем, как оно практически реализуется в Python. Поэтому давайте бегло взглянем на синтаксис инструкции `assert` в соответствии с документацией Python[1]:

```
инструкция_assert ::= "assert" выражение1 ["," выражение2]
```

[1] См. документацию Python «Инструкция `assert`»: https://docs.python.org/3/reference/simple_stmts.html%23the-assert-statement

В данном случае выражение1 — это условие, которое мы проверяем, а необязательное выражение2 — это сообщение об ошибке, которое выводится на экран, если утверждение дает сбой. Во время исполнения программы интерпретатор Python преобразовывает каждую инструкцию `assert` примерно в следующую ниже последовательность инструкций:

```
if __debug__:
     if not выражение1:
         raise AssertionError(выражение2)
```

В этом фрагменте кода есть две интересные детали.

Перед тем как данное условие инструкции `assert` будет проверено, проводится дополнительная проверка глобальной переменной `__debug__`. Это встроенный булев флажок, который при нормальных обстоятельствах имеет значение `True`, — и значение `False`, если запрашивается оптимизация. Мы поговорим об этом подробнее чуть позже в разделе, посвященном «распространенным ловушкам».

Кроме того, вы можете применить выражение2, чтобы передать необязательное сообщение об ошибке, которое будет показано в отчете об обратной трассировке вместе с исключением `AssertionError`. Это может еще больше упростить отладку. Например, я встречал исходный код такого плана:

```
>>> if cond == 'x':
 ...     do_x()
 ... elif cond == 'y':
 ...     do_y()
 ... else:
 ...     assert False, (
 ...         'Это никогда не должно произойти, и тем не менее это '
 ...         'временами происходит. Сейчас мы пытаемся выяснить'
 ...         'причину. Если вы столкнетесь с этим на практике, то '
 ...         'просим связаться по электронной почте с dbader. Спасибо!')
```

Разве это не ужасно? Конечно, да. Но этот прием определенно допустим и полезен, если в одном из своих приложений вы сталкиваетесь с плавающей ошибкой Гейзенбаг[1].

[1] См. Википедию: https://en.wikipedia.org/wiki/Heisenbug и https://ru.wikipedia.org/wiki/Гейзенбаг

Распространенные ловушки, связанные с использованием инструкции assert в Python

Прежде чем вы пойдете дальше, есть два важных предостережения, на которые я хочу обратить ваше внимание. Они касаются использования инструкций `assert` в Python.

Первое из них связано с внесением в приложения ошибок и рисков, связанных с нарушением безопасности, а второе касается синтаксической причуды, которая облегчает написание *бесполезных* инструкций `assert`.

Звучит довольно ужасно (и потенциально таковым и является), поэтому вам, вероятно, следует как минимум просмотреть эти два предостережения хотя бы бегло.

Предостережение № 1: не используйте инструкции assert для проверки данных

Самое большое предостережение по поводу использования утверждений в Python состоит в том, что утверждения могут быть глобально отключены[1] переключателями командной строки `-O` и `-OO`, а также переменной окружения `PYTHONOPTIMIZE` в CPython.

Это превращает любую инструкцию `assert` в нулевую операцию: утверждения `assert` просто компилируются и вычисляться не будут, это означает, что ни одно из условных выражений не будет выполнено[2].

Это преднамеренное проектное решение, которое используется схожим образом во многих других языках программирования. В качестве побочного эффекта оно приводит к тому, что становится чрезвычайно опасно использовать инструкции `assert` в виде быстрого и легкого способа проверки входных данных.

---

[1] См. документацию Python «Константы (`__debug__`)»: https://docs.python.org/3/library/constants.html%23__debug__

[2] Нулевая операция (null-operation) — это операция, которая не возвращает данные и оставляет состояние программы без изменений. См. https://en.wikipedia.org/wiki/Null_function — *Примеч. пер.*

Поясню: если в вашей программе утверждения `assert` используются для проверки того, содержит ли аргумент функции «неправильное» или неожиданное значение, то это решение может быстро обернуться против вас и привести к ошибкам или дырам с точки зрения безопасности.

Давайте взглянем на простой пример, который демонстрирует эту проблему. И снова представьте, что вы создаете приложение Python с интернет-магазином. Где-то среди программного кода вашего приложения есть функция, которая удаляет товар по запросу пользователя.

Поскольку вы только что узнали об `assert`, вам не терпится применить их в своем коде (я бы точно так поступил!), и вы пишете следующую реализацию:

```
def delete_product(prod_id, user):
     assert user.is_admin(), 'здесь должен быть администратор'
     assert store.has_product(prod_id), 'Неизвестный товар'
     store. get_product(prod_id).delete()
```

Приглядитесь поближе к функции `delete_product`. Итак, что же произойдет, если инструкции `assert` будут отключены?

В этом примере трехстрочной функции есть две серьезные проблемы, и они вызваны неправильным использованием инструкций `assert`:

1. **Проверка полномочий администратора инструкциями `assert` несет в себе опасность.** Если утверждения `assert` отключены в интерпретаторе Python, то проверка полномочий превращается в нулевую операцию. И поэтому *теперь любой пользователь может удалять товары*. Проверка полномочий вообще не выполняется. В результате повышается вероятность того, что может возникнуть проблема, связанная с обеспечением безопасности, и откроется дверь для атак, способных разрушить или серьезно повредить данные в нашем интернет-магазине. Очень плохо.

2. **Проверка `has_product()` пропускается, когда `assert` отключена.** Это означает, что метод `get_product()` теперь можно вызывать с недопустимыми идентификаторами товаров, что может привести к более серьезным ошибкам, — в зависимости от того, как написана наша программа. В худшем случае она может стать началом запуска DoS-атак.