Python Tricks The Book

슬기로운 파이썬 트릭 짧고 흥미로운 코드로 배우는 파이썬 실용 코딩

댄 베이더 지음 | 전석환 옮김 프로그래밍인사이트

슬기로운 파이썬 트릭

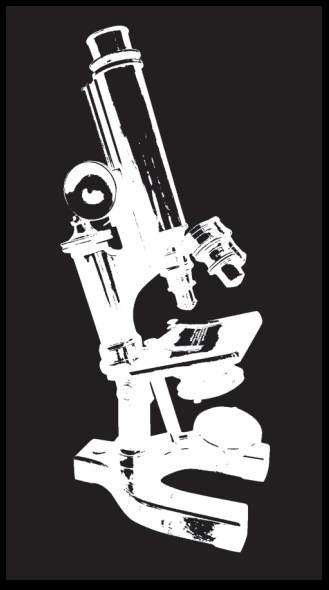
Python Tricks The Book

Python Tricks: ISBN 978-1775093305

슬기로운 파이썬 트릭

초판 PDF 1.0 2020년 6월 1일 **지은이** 댄 베이더 **옮긴이** 전석환 **펴낸이** 한기성 **펴낸곳** 인사이트 **편집** 송우일 **등록번호** 제2002-000049호 **등록일자** 2002년 2월 19일 **주소** 서울시 마포구 연남로5길 19-5 **전화** 02-322-5143 **팩스** 02-3143-5579 **블로그** http://blog.insightbook.co.kr **이메일** insight@insightbook.co.kr **ISBN** 978-89-6626-266-3

프로그래밍 인사이트



슬기로운 파이썬 트릭

짧고 흥미로운 코드로 배우는 파이썬 실용 코딩

댄 베이더 지음 | 전상현 옮김

인사이트 insight

차례

옮긴이의 글

출판사의 제안을 받기도 했지만 이 책을 번역한 계기는 평소 파이썬에 대한 애정 때문이었다. 회사에서 그리고 개인적으로 파이썬을 개발 언어로 사용한 지 벌써 10년이 넘었다. 대학원 유학 시절 처음 파이썬 2.4를 접한 후 교내 가상현실(virtual reality, VR)을 다루는 스튜디오에서 예술 프로젝트의 연구 조교로 일할 때도 파이썬을 사용했다. 한국에 돌아와서는 파이썬 기반 웹 프레임워크인 플라스크(Flask)를 활용해 다수의 상업용 웹 사이트를 만들었고 각종 백엔드 프로젝트에도 파이썬을 두루 적용했다. 물론 취미로 하는 코딩에서도 파이썬은 좋은 도구가 되고 있다.

이 책은 파이썬 프로그래밍에 도움이 될 만한 유용한 비법을 안내해 준다. 파이썬이 쉽게 익히고 사용하기 좋은 프로그래밍 언어이기는 하지만 파이썬다운 코드를 작성하는 데는 많은 수고가 필요하다. 그리고 이 책이 그 수고를 더는 데 도움이 된다고 생각한다.

지은이는 시행착오를 겪으며 익힌 경험을 바탕으로 이 책을 통해 파이썬 프로그래머가 좀 더 파이썬다운 코드를 작성하도록 돕는다. 그리고 실용적인 토막 코드 예제는 업무에 바로 사용할 수 있을 정도다.

2장의 문자열 형식 처리에 관한 간단한 방법부터 3장의 람다와 데코레이터 같은 파이썬의 고급 기능까지 그동안 몰랐거나 알았지만 쉽게 써 볼 수 없었던 파이썬의 좋은 기능을 이 책에 전부 담아 놓았다. 나 역시 평소 제대로 이해하지 못했던 것을 발견하고 번역을 하는 동안 예제를 실행해 보며 배우고 익히기도 했다.

끝으로 번역에 신경 쓴 부분은 구어체와 농담 같은 표현이었고 용어는 최대한 쉽게 이해할 수 있도록 옮기려고 했다. 작업하는 동안 꼼꼼하게 읽어 주신 인사이트 출판사 편집 팀에 감사드린다. 파이썬 프로그래머들이 이 책을 통해 많은 것을 얻어 가길 바란다.

추천사

파이썬이라는 프로그래밍 언어를 알게 된 지 거의 10년이 지났다. 파이썬을 처음 배울 당시에는 약간 내키지 않았다. 다른 언어로 프로그래밍을 하다가 갑작스런 개발 업무가 생겨 팀 전체가 파이썬을 사용하는 곳에 배치됐기 때문이다. 그것이 내 파이썬 여행의 시작이었다.

파이썬을 처음 소개받았을 때 파이썬은 쉬워서 금방 배울 수 있을 거라는 이야기를 들었다. 동료에게 파이썬을 공부하기 위한 자료를 요청했을 때 내가 받은 건 파이썬 공식 문서 링크뿐이었다. 문서를 읽는데 처음에는 혼란스러웠다. 그리고 익숙하게 찾아보기까지 오래 걸렸다. 스택오버플로에서 답을 구해야 하는 경우도 종종 있었다.

다른 프로그래밍 언어를 써 왔기 때문에 프로그래밍을 하는 방법이나 클래스와 객체의 개념을 배우는 데 필요한 자료는 찾지 않았다. 대신 여타 언어로 코드를 작성하는 것과 다른, 파이썬 설정과 작성 방법 같은 파이썬의 특징을 알 수 있는 특별한 자료를 찾았다.

이 언어를 완전히 이해하기까지 몇 년이 걸렸다. 나는 댄의 책을 읽으면서 파이썬을 배울 때 이런 책을 읽었으면 좋았겠다고 생각했다.

예를 들어 파이썬의 독특한 특징 중 하나인 리스트 내포식이 나를 놀라게 했다. 댄이 이 책에서 언급했듯이 다른 언어에서 파이썬으로 막 들어온 사람은 `for` 반복문 사용 방식에 대해 이야기를 듣는다. 파이썬으로 프로그래밍을 시작했을 때 받은 코드 리뷰 코멘트 중 하나가 "여기에 리스트 내포식을 사용하는 게 낫지 않나?"였다. 댄은 6장에서 이 개념을 분명하게 설명한다. 파이썬다운 방식으로 반복하는 법과 이터레이터와 제너레이터를 사용하는 방식까지 보여준다.

'2.5 문자열 형식에 관한 충격적인 진실'에서 댄은 파이썬 문자열 형식을 다루는 여러 가지 방식을 설명한다. "어떤 일을 하는 분명한 방법은 하나뿐이어야 한다"는 파이썬의 선(禪, Zen)을 무시하는 예 중 하나다. 댄은 우리에게 다

양한 방식을 보여 주는데 파이썬에 새롭게 추가된 f-문자열을 포함해서 각 방법의 장단점을 설명한다.

'8장 파이썬다운 생산성 향상 기법'은 또 다른 좋은 자료다. 여기에는 파이썬 프로그래밍 언어 외의 측면이 포함되어 있고 프로그램을 디버그하는 방법과 의존성을 관리하는 방법, 파이썬 바이트코드의 내부를 들여다볼 수 있는 팁도 있다.

내 친구인 댄 베이더가 쓴 이 책을 소개할 수 있어서 영광이다.

나는 CPython 코어 개발자로 파이썬에 공헌함으로써 많은 커뮤니티 구성원과 만났다. 이 여정을 통해 멘토와 동지를 찾았고 새로운 친구를 사귈 수 있었다. 그들은 파이썬이 단지 코드가 아니라 커뮤니티라는 것을 상기시켜 줬다.

파이썬 프로그래밍에 통달하는 것은 단지 언어의 이론적 측면을 파악하는 일이 아니다. 커뮤니티에서 사용되는 모범 사례와 규칙을 이해하고 채택하는 것 역시 중요하다.

이 여행에서 댄의 책이 독자들을 도울 것이다. 이 책을 읽은 후 파이썬 프로그램을 작성하면 자신감이 더 생기리라고 확신한다.

– 마리아타 위자야(Mariatta Wijaya), 파이썬 코어 개발자(mariatta.ca)

1장

Python Tricks

소개

1.1 파이썬 트릭이란?

> **파이썬 트릭**: 짧은 파이썬 토막 코드(code snippet)를 교육 도구로 사용한다. 파이썬 트릭(trick)은 간단한 설명으로 파이썬의 한 부분을 가르치거나 동기 부여를 위한 예제로 사용되어 직관적인 이해를 높이거나 파이썬에 더 깊이 파고들 수 있게 한다.

파이썬 트릭은 짧은 코드 스크린샷 연재로 시작됐고 일주일 동안 트위터에 공유됐다. 놀랍게도 극찬을 얻었고 며칠 동안 리트윗(retweet)되면서 공유됐다.

점점 더 많은 개발자가 전체 연재를 구할 수 있는 방법을 묻기 시작했다. 사실 처음에는 파이썬 관련 주제를 다양하게 다루는 몇 가지 트릭만 있었고 제대로 된 연재 계획은 없었다. 단지 재미난 트위터 실험이었다.

그러나 이러한 요청들을 보고 나는 짧고 유용한 코드 예제가 교육 도구로 탐구할 가치가 있다고 생각했다. 결국 파이썬 트릭을 몇 개 더 만들어 이메일 연재로 공유했다. 며칠 만에 수백 명의 파이썬 개발자가 구독했고 그 반응은 폭발적이었다.

다음 몇 주 동안 꾸준하게 파이썬 개발자들의 구독이 늘어났다. 그들은 완전히 이해하는 데 어려움을 겪고 있는 언어의 일부를 분명하게 설명해 준 것에 감사를 표시했다. 이런 반응을 듣는 건 굉장한 경험이었다. 나는 이 파이썬 트

릭들이 그저 코드 스크린샷일 뿐이라고 생각했지만 많은 개발자에게 큰 도움이 되었다.

파이썬 트릭 실험을 더 열심히 하기로 하고 이메일을 30회가량 보냈을 때였다. 여전히 그냥 제목과 코드 스크린샷 구성이었는데 곧 형식의 제한을 깨닫게 됐다. 이 무렵 시각 장애인 개발자가 내게 이메일을 보내 이 파이썬 트릭이 스크린 리더로 읽을 수 없는 이미지여서 실망이라고 했다.

분명히 더 많은 독자들이 더 쉽게 접근하고 더 매력적으로 느낄 수 있도록 하기 위해 이 프로젝트에 많은 시간을 투자해야 했다. 그래서 나는 일반 텍스트와 적절한 HTML 기반의 문법 강조를 섞어서 파이썬 트릭 이메일 전체 연재를 다시 만들었다. 새롭게 작업한 파이썬 트릭은 한동안 멋지게 진행됐다. 반응을 보니 코드 샘플을 복사하고 붙여 넣기를 하면서 재미있게 노는 개발자들이 많아 보였다.

이메일 연재에 가입하는 개발자가 점점 늘어남에 따라 내가 받는 질문과 응답의 패턴을 발견하기 시작했다. 일부 트릭은 동기를 부여하는 예제로 효과가 있었다. 그러나 좀 더 복잡한 트릭에는 개발자가 더 깊이 이해할 수 있도록 돕는 추가 자료나 독자를 안내해 주는 설명이 부족했다.

이 부분은 크게 개선해야 할 부분이라고 생각했다. 파이썬 개발자가 더 훌륭해지도록 돕는 것이 dbader.org에 쓴 내 임무 선언문이다. 그리고 파이썬 트릭은 분명 그 목표에 다가설 수 있는 기회였다.

나는 그동안 보낸 이메일에 실린 파이썬 트릭 중에서 가장 좋은 내용을 골라 새로운 파이썬 책을 쓰기로 했다.

- 짧고 소화하기 쉬운 예제로 언어의 가장 멋진 면을 가르치는 책
- 멋진 파이썬 특징들이 있는 (맛있는!) 뷔페처럼 사용되고 동기 유발을 고취하는 책
- 저절로 손이 가고 파이썬을 깊이 이해하는 데 도움이 되는 책

이 책은 내가 기꺼이 하는 일이자 거대한 실험이다. 나는 여러분이 이 책을 즐겁게 읽고 파이썬에 대해 뭔가 배우기를 바란다.

1.2 이 책이 독자에게 알려 주는 것

이 책의 목표는 독자들이 더 나은 파이썬 개발자가 되도록 돕는 것이다. 어떻게 이 책을 읽어야 그 목표를 이룰 수 있을지 궁금할 것이다.

이 책은 단계별 파이썬 학습서가 아니다. 그리고 초급 파이썬 과정도 아니다. 파이썬을 배우는 초기 단계에 있다면 이 책만으로는 전문적인 파이썬 개발자가 될 수 없다. 이 책을 읽는 것이 유익할 수는 있지만 기초적인 파이썬 실력을 쌓으려면 다른 자료와 함께 공부해야 한다.

이미 파이썬에 대한 지식을 가지고 있고 다음 단계로 넘어가고 싶다면 이 책을 최대한 활용할 수 있다. 파이썬 코딩을 조금 해 봤고 더 깊게 파고들어 완전히 이해해서 파이썬다운 코드를 만들어 보고 싶다면 이 책이 큰 도움이 될 것이다.

이미 다른 프로그래밍 언어에 대한 경험이 있고 파이썬에 익숙해지고자 한다면 이 책을 읽는 게 큰 도움이 될 것이다. 좀 더 숙련된 파이썬 개발자가 될 수 있는 실용적인 팁과 디자인 패턴을 발견할 것이다.

1.3 이 책을 읽는 방법

이 책을 읽는 가장 좋은 방법은 뷔페처럼 사용하는 것이다. 이 책에 있는 파이썬 트릭은 각각 구분돼 있어서 흥미로운 곳으로 바로 넘어갈 수 있다. 사실 나는 그렇게 하도록 권장한다. 물론 모든 파이썬 트릭을 책에 배치된 순서대로 읽을 수도 있다.

트릭들 중 일부는 즉시 이해할 수 있을 만큼 쉬우며 각 장을 읽기만 해도 일상 업무에 반영하는 데 어려움이 없다. 반면 사용하려면 약간의 시간이 필요한 트릭도 있다.

특정 트릭을 자신의 프로그램에서 돌아가게 하는 데 어려움을 겪을 경우에는 파이썬 인터프리터(interpreter)에서 각 코드 예제를 가지고 놀아 보면 도움이 된다.

내용을 완전히 이해하기 어려우면 나에게 연락해도 된다. 그러면 여러분을

도울 수 있고 이 책의 설명을 개선하는 데도 도움이 된다. 장기적으로 이 책은 여러분뿐 아니라 모든 파이썬 사용자에게 도움이 될 것이다.

2장

Python Tricks

파이썬 코드를 정돈하기 위한 패턴

2.1 assert 문으로 방어하기

간혹 정말 도움이 되는 언어의 특징이 그 가치에 비해 주목을 적게 받기도 한다. 파이썬에 내장된 assert 문도 그렇다.

지금부터 단언(assertion)문을 살펴보자. 단언문을 사용하여 파이썬 프로그램의 에러를 자동으로 감지하는 방법을 배울 것이다. 단언문으로 프로그램 안정성을 높이고 프로그램을 쉽게 디버그할 수 있다.

이 시점에서 "단언문이 무엇이고 어디에 유용한가?" 하고 궁금해할 것이다.

파이썬의 단언문은 어떤 조건을 테스트하는 디버깅 보조 도구라는 것이 핵심이다. 단언 조건이 참이면 아무 일도 일어나지 않고 프로그램이 정상으로 계속 실행된다. 그러나 조건이 거짓으로 판명되면 `AssertionError` 예외가 발생한다.

파이썬의 단언문 예제

단언문이 어디에 유용한지 알려 주는 간단한 예가 있다. 독자들이 프로그램을 개발하다가 마주할 수 있는 실제 문제와 조금은 비슷하도록 예제를 만들었다.

온라인 쇼핑몰을 만들고 있다고 가정해 보자. 할인 쿠폰 기능을 시스템에 추가하려고 다음과 같은 apply_discount 함수를 작성했다.

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

assert 문이 보일 것이다. 이 함수로 계산된 할인 가격은 0달러보다 낮을 수 없으며 제품의 원래 가격보다 높을 수 없다.

유효한 할인을 적용해 이 함수를 호출하면 실제로 의도한 대로 작동하는지 확인해 보자. 이 예제에서 우리 쇼핑몰의 제품은 평범한 딕셔너리(dictionary) 형으로 표시된다. 아마도 실제 애플리케이션에서 이렇게 사용되지는 않겠지만 단언문을 설명하기에는 적절하다. 149달러짜리 멋진 신발 한 켤레를 예로 들어 보자.

```
>>> shoes = {'name': 'Fancy Shoes', 'price': 14900}
```

내가 금액을 센트로 표시해서 통화 반올림 문제를 해결하려고 한 게 눈에 띌 것이다. 대체로 좋은 생각이다. 잠시 딴 길로 샜다. 이제 이 신발에 25% 할인을 적용하면 판매 가격은 111.75달러다.

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

좋다. 훌륭하게 동작했다. 자, 잘못된 할인율을 적용해 보자. 예를 들어 200% '할인'이라면 우리가 고객에게 돈을 줘야 한다.

```
>>> 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
```

보다시피 이 잘못된 할인을 적용하려고 할 때 프로그램은 AssertionError로 중

단된다. 200% 할인이 `apply_discount` 함수에 포함된 단언 조건을 위반했기 때문이다.

또한 예외 스택트레이스(stacktrace)가 실패한 단언문을 포함하는 코드 라인을 정확히 가리키는지 확인할 수 있다. 온라인 쇼핑몰을 테스트하는 동안 이러한 에러 중 하나가 발생하면 예외 트레이스백(traceback)을 살펴봄으로써 발생한 문제를 쉽게 찾을 수 있다.

이렇게 하면 디버깅 작업이 상당히 빨라지고 장기적으로 프로그램을 유지보수하기 좋아진다. 이것이 단언문의 힘이다.

그냥 일반적인 예외 처리를 사용하면 안 되나?

아마도 이전 예제에서 `if` 문과 예외 처리를 사용하지 않은 이유가 궁금할 것이다.

단언문을 적절하게 사용하면 개발자에게 프로그램에서 복구할 수 없는 에러를 알릴 수 있다. 단언문은 사용자가 시정 조치를 취하거나 다시 시도할 수 있는 'File-Not-Found' 에러와 같은 예상되는 에러 조건을 알리기 위한 것이 아니다.

단언문은 내부 자체 검사(self-check)[1]를 하게 되어 있는데 코드에서 일부 조건을 불가능하다고 선언하는 방식으로 작동한다. 이러한 조건 중 하나가 만족스럽지 않으면 프로그램에 버그가 있음을 의미한다.

프로그램에 버그가 없으면 이런 조건은 발생하지 않는다. 그러나 그것이 발생하면 프로그램은 '불가능한' 조건이 발동되었음을 알려 주는 단언문 에러를 내고 죽을 것이다. 이렇게 하면 프로그램에서 버그를 추적하고 수정하기가 훨씬 쉬워진다. 나는 이렇게 개발이 편해지는 기술을 좋아한다.

파이썬의 단언문은 런타임 에러를 처리하기 위한 메커니즘이 아니라 디버깅을 돕는 것임을 명심하자. 단언문을 사용하는 목적은 개발자가 버그의 근본 원인을 더 빨리 발견하도록 하는 것이다. 프로그램에 버그가 없다면 단언문 에러는 절대 발생하지 않는다.

1 (옮긴이) 신뢰성을 높이기 위해 시스템이 스스로 처리 과정이나 결과를 점검하는 기능을 말한다.

계속해서 단언문으로 할 수 있는 다른 일들에 대해 자세히 살펴보고 실제 시나리오에서 두 가지 일반적인 함정을 다룰 것이다.

파이썬 단언문 문법

파이썬 언어의 기능을 활용하기 전에 그 기능이 실제로 어떻게 구현됐는지 공부하는 것이 좋다. 이제 파이썬 공식 문서에 나온 단언문 문법을 간략하게 살펴보자.[2]

```
assert_stmt ::= "assert" expression1 ["," expression2]
```

이 경우 expression1은 테스트할 조건이고 추가적인 expression2는 단언문이 실패할 경우 표시되는 에러 메시지다. 실행 시에 파이썬 인터프리터는 각 단언문을 대략 다음과 같은 문장으로 변환한다.

```
if __debug__:
    if not expression1:
        raise AssertionError(expression2)
```

이 토막 코드에는 두 가지 흥미로운 점이 있다.

단언 조건을 검사하기 전에 __debug__ 전역 변수에 대한 추가 검사가 있다. 이 조건은 일반적인 상황에서 참이고 최적화가 필요한 경우에는 거짓이다. 이에 대해서는 이어지는 '파이썬 단언문의 일반적인 함정' 절에서 더 이야기하겠다.

또한 expression2를 사용하여 트레이스백 메시지의 AssertionError와 함께 표시될 추가적인 에러 메시지를 전달할 수 있다. 이렇게 하면 디버깅을 더욱 단순하게 할 수 있다. 예를 들어 다음 코드를 보자.

```
>>> if cond == 'x':
...     do_x()
... elif cond == 'y':
...     do_y()
... else:
...     assert False, (
...         'This should never happen, but it does '
```

2 파이썬 공식 문서: 'The assert statement'(https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement)

```
...         'occasionally. We are currently trying to '
...         'figure out why. Email dbader if you '
...         'encounter this in the wild. Thanks!')
```

코드가 보기 흉하지만, 애플리케이션 개발 과정에서 고치려고 하면 사라지는 하이젠버그(Heisenbug)로 곤란을 겪을 때 확실히 유용하고 도움이 된다.

파이썬 단언문의 일반적인 함정

계속 진행하기 전에 파이썬에서 단언문을 사용할 때 주의해야 할 중요한 사항이 두 가지 있다.

첫 번째는 애플리케이션에 생기는 보안 위험과 버그에 대한 것이고 두 번째는 쓸모없는 단언문을 작성하는 버릇에 대한 것이다.

잠재적으로 끔찍한 상황을 불러올 수 있으므로 적어도 다음 두 가지 주의 사항은 꼭 훑어보기 바란다.

주의 사항 1. 데이터 유효성 검증에 단언문을 사용하지 말자

파이썬에서 단언문을 사용하는 데 있어 가장 큰 주의 사항은 `-O` 및 `-OO` 커맨드라인 스위치와 CPython의 `PYTHONOPTIMIZE` 환경 변수를 사용하여 단언문을 전역으로 비활성할 수 있다는 사실이다.[3]

이는 단언문을 널(null) 연산으로 만든다. 단언문은 그냥 컴파일만 되고 평가되지 않으므로 조건식은 실행되지 않는다.

이는 다른 많은 프로그래밍 언어에서도 비슷하게 사용되는 의도적인 설계 방식이다. 하지만 그 부작용으로, 개발자들이 입력 데이터의 유효성을 검사하는 빠르고 쉬운 방법으로 단언문을 오용하기도 하는데, 이렇게 하면 극도로 위험해진다.

설명해 보겠다. 프로그램에서 함수 인자에 '잘못되거나' 예기치 않은 값이 포함되어 있는지 확인하는 데 단언문을 사용하면 역효과를 일으켜 버그나 보안 허점을 초래할 수 있다.

이 문제를 보여 주는 간단한 예를 살펴보자. 파이썬을 사용하여 온라인 쇼핑

3 파이썬 공식 문서: 'Constants (`__debug__`)'(https://docs.python.org/3/library/constants.html#__debug__)

몰을 작성한다고 가정해 보자. 쇼핑몰 코드의 어딘가에 사용자의 요청에 따라 제품을 삭제하는 함수가 있다.

단언문에 대해 배웠으므로 여러분은 프로그램 코드에서 이를 사용하고 싶을 것이다. 그리고 다음과 같이 구현한다.

```
def delete_product(prod_id, user):
    assert user.is_admin(), 'Must be admin'
    assert store.has_product(prod_id), 'Unknown product'
    store.get_product(prod_id).delete()
```

이 `delete_product` 함수를 살펴보자. 단언문이 비활성화되면 어떻게 될까?

이 함수 예제에는 두 가지 심각한 문제가 있으며 이 문제는 단언문을 잘못 사용하여 발생한 것이다.

1. **단언문에서 관리자 권한을 확인하는 것은 위험하다.** 파이썬 인터프리터에서 단언문을 비활성화하면 이 확인 코드는 널 연산으로 변한다. 따라서 어떤 사용자도 이제 상품을 삭제할 수 있다. 권한 검사가 실행되지 않는다. 이로 인해 보안 문제가 발생하고 공격자가 온라인 쇼핑몰에서 데이터를 파괴하거나 심각하게 손상시킬 수 있는 문이 열리게 된다. 좋지 않다.
2. **단언문이 비활성화되면 `has_product()` 검사를 건너뛴다.** 이는 `get_product()`가 잘못된 제품 아이디(ID)로 호출될 수 있음을 의미한다. 이렇게 되면 프로그램을 어떻게 작성했느냐에 따라 더 심각한 버그를 초래할 수 있다. 최악의 경우 쇼핑몰이 서비스 거부 공격을 받을 수도 있다. 예를 들어 사용자가 알 수 없는 제품을 삭제하려고 시도할 때 쇼핑몰 서비스가 크래시를 일으킨다면 공격자가 유효하지 않은 삭제 요청을 다량으로 보내 서비스를 중지시킬 수 있다.

이러한 문제를 어떻게 피할 수 있을까? 답은 데이터 유효성 검사를 수행하기 위한 평가 용도로는 절대로 사용하지 않는 것이다. 유효성 검사에는 단언문 대신 일반적인 `if` 문을 사용하고, 필요에 따라 유효성 검사 에러를 발생시킨다. 다음은 수정한 코드다.

```
def delete_product(product_id, user):
    if not user.is_admin():
        raise AuthError('Must be admin to delete')
    if not store.has_product(product_id):
        raise ValueError('Unknown product id')
    store.get_product(product_id).delete()
```

또한 이 업데이트된 예제는 뜻이 구체적이지 않은 AssertionError 에러를 발생시키는 대신 (우리가 직접 정의한) ValueError나 AuthError와 같이 맥락에 어울리는 에러를 내는 이점이 있다.

주의 사항 2. 절대 실패하지 않는 단언문

놀랍게도 항상 참이 되는 파이썬 단언문을 실수로 작성하기 쉽다. 과거에 나도 내 발등을 찍은 적이 있다. 여기 그 문제를 간단하게 보여 주겠다.

assert 문에서 첫 번째 인자로 튜플을 전달하면 그 단언문은 항상 참이 되므로 결코 실패하지 않는다.

예를 들어 이 단언문은 절대로 실패하지 않는다.

```
assert(1 == 2, 'This should fail')
```

이는 비어 있지 않은 튜플이 파이썬에서 항상 참인 것과 관련이 있다. 튜플을 단언문에 넘기면 항상 단언 조건이 참이 되며 앞의 단언문은 실패하지 않고 예외를 감지할 수 없으므로 쓸모없어진다.

이런 직관적이지 않은 동작 때문에 실수로 잘못된 멀티라인 단언문을 작성하기가 상대적으로 쉽다. 실제로 나도 (테스트가 성공했다는) 잘못된 신호를 주는 테스트 케이스들을 테스트 스위트에 열심히 추가한 적이 있다. 이제 단위 테스트 중 하나에 다음 단언문이 있다고 상상해 보자.

```
assert (
    counter == 10,
    'It should have counted all the items'
)
```

얼핏 보면 이 테스트 케이스는 완벽해 보인다. 그러나 결코 잘못된 결과를 잡을 수 없다. 단언문은 카운터 변수의 상태에 관계없이 항상 True로 평가된다.

왜 그럴까? 이 단언문은 항상 참인 튜플 객체를 검사하기 때문이다.

이처럼 자기 발등을 찍기란 그다지 어렵지 않다. 코드 린터(linter)[4]를 사용하면 이러한 문법적 괴이함에 속는 실수를 예방할 수 있다. 파이썬 3 최신판에서는 이런 모호한 단언문에 대한 문법 경고도 표시된다.

한편, 이는 단위 테스트 케이스를 작성할 때면 항상 스모크 테스트를 해야 하는 이유이기도 하다. 다음 테스트를 작성하기 전에 방금 작성한 테스트가 실제로 실패하는지 확인하자.

요약: 파이썬 단언문

이런 위험한 면이 있지만 파이썬의 단언문은 파이썬 개발자가 자주 사용하지 않는 강력한 디버깅 도구라고 생각한다.

단언문 작동 방식과 적용 시기를 이해하면 디버깅과 유지 보수가 쉬운 파이썬 프로그램을 작성할 수 있다.

단언문은 파이썬 지식을 한 단계 높여 주고 좀 더 다재다능한 파이썬 사용자가 되는 데 도움이 되는 훌륭한 기술이다. 실제로 단언문 덕분에 나는 디버깅에 허비하는 시간을 놀랍도록 줄일 수 있었다.

요점 정리

- 파이썬의 단언문은 프로그램 내부 자체 검사로 조건을 테스트하는 디버깅 도구다.
- 단언문은 개발자가 버그를 식별하는 데 도움이 되지만 런타임 에러를 처리하기 위한 메커니즘은 아니다.
- 인터프리터 설정으로 단언문을 전역적으로 비활성화할 수 있다.

2.2 보기 좋은 쉼표 배치

다음은 파이썬의 리스트, 딕셔너리, 세트 상수에서 항목을 추가, 제거할 때 유

4 파이썬 테스트에서 무의미한 단언문을 피하는 방법에 대해 내가 쓴 글을 참고하자: http://dbader.org/blog/catching-bogus-python-asserts