36장

프로그램 테스트 및 디버깅하기

36장 프로그램 테스트 및 디버깅하기

36.1 unittest 라이브러리 사용하기

36.1 unittest 라이브러리 사용하기

» 프로그램에 대한 테스트 구조를 만들 수 있게 지원한다.

- 여러 테스트를 한데 묶은 테스트 스위트(suite)를 만들려면 먼저 그 스위트를 표현하는 클래스를 만들어야 한다. 클래스 안에 들어 있는 메서드들은 모두 서로 다른 테스트를 표현한다.
- 테스트 시 실행하고 싶은 모든 테스트 메서드는 test_로 시작해야 한다. test_ 뒤에는 어떤 이름이든 (정상적인 파이썬 메서드 이름이라면) 올 수 있다.

```
import unittest ---- unittest 라이브러리를 불러옴


class TestMyCode(unittest.TestCase): ---- 실행할 테스트 스위트를 나타내는 클래스
    def test_addition_2_2(self): ---- 2 + 2가 4인지 검증하는 메서드
        self.assertEqual(2+2, 4) ---- 2 + 2와 4가 같다는 단언문(assert)을 통해
                                      2 + 2와 4가 같음을 검사하는 테스트
    def test_subtraction_2_2(self): ---- 2 - 2가 4가 아님을 검증하는 메서드
        self.assertNotEqual(2-2, 4) ---- 2 - 2와 4가 같지 않다는 단언문을 통해
                                         2 - 2와 4가 다름을 검사하는 테스트

unittest.main() ---- 여러분이 만든 테스트 스위트 클래스 안에 들어 있는 테스트들을 실행함
```

코드 36-1 간단한 테스트 스위트

36.1 unittest 라이브러리 사용하기

- 첫 번째 테스트에서는 2 + 2와 4가 같은지 테스트하는데 결과가 참이고 두 번째 테스트에서는 2 – 2와 4가 같지 않은지 테스트하는데 역시 결과가 참이기 때문이다.
- 이제 다음과 같이 바꿔서 테스트 중 하나를 실패하게 만들어 보자.

```
def test_addition_2_2(self):
    self.assertEqual(2+2, 5)
```

```
FAIL: test_addition_2_2 (__main__.TestMyCode)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/Ana/.spyder-py3/temp.py", line 5, in test_addition_2_2
    self.assertEqual(2+2, 5)
AssertionError: 4 != 5
----------------------------------------------------------------------
Ran 2 tests in 0.002s
FAILED (failures=1)
```

- 어떤 테스트 스위트가 실패했는가: TestMyCode
- 어떤 테스트가 실패했는가: test_addition_2_2
- 테스트의 어떤 부분(줄)에서 실패했는가: self.assertEqual(2+2, 5)
- 왜 테스트가 실패했는가(테스트에서 원래 예상했던 값과 실제로 테스트가 얻은 값은 무엇인지 비교): 4 != 5

36.2 프로그램과 테스트 분리하기

36.2 프로그램과 테스트 분리하기

여러분이 작성하는 코드(프로그램의 일부)와 프로그램을 테스트하기 위한 코드를 서로 분리해야한다.

- 분리하면 코드를 서로 다른 파일에 넣을 수 있으므로 모듈화를 더 강화할 수 있다.
- 테스트와 프로그램을 분리하면 프로그래머가 프로그램을 읽을 때 테스트 코드 때문에 방해를 받는 것을 방지할 수 있다.

코드 36-2의 두 함수가 한 파일에 들어 있다고 하자.

- 한 함수는 어떤 수가 소수(prime, 1과 자기 자신으로 밖에 나눠지지 않는 정수이며 1은 아님)인지 검사하고 True나 False를 반환한다. 다른 함수는 인자로 받은 수의 절댓값을 반환한다.
- 현재 두 함수 구현에 오류가 있다.

다른 파일에 두 함수가 원하는 대로 동작하는지 검사하는 단위 테스트를 작성할 수 있다.

- 함수 하나를 테스트할 때 다양한 입력에 대해 잘 작동하는지 확인하기 위해 여러 다른 테스트 함수를 많이 만들 수 있기 때문에, 함수별로 테스트 스위트를 따로 만드는 것도 좋은 생각이다.
- 작성한 모든 함수에 대해 테스트를 만드는 것을 단위 테스트(unit test)라고 한다. 각 함수를 하나의 단위(유닛) 그 자체로만 테스트하기 때문이다.

36.2 프로그램과 테스트 분리하기


```
def is_prime(n):
    prime = True
    for i in range(1,n):
        if n%i == 0:
            prime = False
    return prime

def absolute_value(n):
    if n < 0:
        return -n
    elif n > 0:
        return n
```

코드 36-2
테스트 대상 함수가 들어 있는 funcs.py 파일

```
import unittest ---- unittest의 클래스와 함수를 불러옴
import funcs ---- funcs.py의 클래스와 함수를 불러옴

class TestPrime(unittest.TestCase): ---- 테스트 묶음 당 하나씩 클래스를 만듦
    def test_prime_5(self): ---- 테스트, 메서드 이름이 어떤 동작을 테스트하는지 표현함
        isprime = funcs.is_prime(5) ---- 5를 인자로 funcs.py에 있는 is_prime을 호출하고 결과를 isprime에 저장
        self.assertEqual(isprime, True) ---- 함수 호출 결과가 True인지 검사
    def test_prime_4(self):
        isprime = funcs.is_prime(4)
        self.assertEqual(isprime, False)
    def test_prime_10000(self):
        isprime = funcs.is_prime(10000)
        self.assertEqual(isprime, False)

class TestAbs(unittest.TestCase):
    def test_abs_5(self):
        absolute = funcs.absolute_value(5)
        self.assertEqual(absolute, 5)
    def test_abs_neg5(self):
        absolute = funcs.absolute_value(-5)
        self.assertEqual(absolute, 5)
    def test_abs_0(self):
        absolute = funcs.absolute_value(0)
        self.assertEqual(absolute, 0)

unittest.main()
```

코드 36-3
테스트가 들어 있는 test.py

36.2 프로그램과 테스트 분리하기

» 코드를 실행하면 여섯 가지 테스트를 실행했으며 그중 두 테스트에서 오류가 발생했음을 알려준다.

```
FAIL: test_abs_0 (__main__.TestAbs)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/Ana/test.py", line 24, in test_abs_0
    self.assertEqual(absolute, 0)
AssertionError: None != 0
======================================================================
FAIL: test_prime_5 (__main__.TestPrime)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/Ana/test.py", line 7, in test_prime_5
    self.assertEqual(isprime, True)
AssertionError: False != True
----------------------------------------------------------------------
Ran 6 tests in 0.000s
FAILED (failures=2)
```

- 이 정보를 가지고 funcs.py에 있는 함수의 오류를 수정할 수 있다. test_abs_0과 test_prime_5라는 테스트가 실패했으므로,
- 함수 소스 코드를 살펴보면서 문제가 되는 부분을 수정할 수 있다. 이런 과정을 디버깅이라 부른다.

36.2 프로그램과 테스트 분리하기

» 셀프 체크 36.2

36.2.1 검사 유형

- unittest 라이브러리에는 어떤 값이 다른 값과 같은지 비교하는 assertEqual 외에 수많은 검사가 있다. 전체 목록은 라이브러리를 참조하라.

- 셀프 체크 36.3

36.3 작성한 코드 디버깅하기

36.3 작성한 코드 디버깅하기

테스트 중에 실패한 것이 있으면 디버깅을 시작해야 한다.

- 실패한 테스트를 통해 코드의 어떤 부분을 살펴봐야 할지 알 수 있고, 어떤 조건에서 오류가 발생하는지도 알 수 있다.

디버깅에서는 종종 막무가내식의 접근이 가장 효율적일 수 있다.

- 막무가내식 접근이란 코드를 한줄한줄 빠짐없이 살펴보면서 종이와 연필을 사용해 값을 적어 내려가는 것을 말한다.
- 프로그램을 한줄한줄 따라 갈 때(이를 트레이싱(tracing)이라 한다) 흔히 저지르는 실수는 코드가 단순하면 옳다고 가정하는 것이다.
- 특히 직접 작성한 코드를 트레이싱할 때 이런 실수를 저지르기 쉽다.

모든 줄을 회의주의자적 관점에서 살펴봐야 한다. 프로그래밍을 전혀 모르는 사람에게 코드를 설명하는 척 하라.

- 이런 디버깅 과정을 고무 오리 디버깅(rubber ducky debugging)이라고 한다.
- 이 과정은 프로그래밍 관련 용어가 아닌 일상적인 우리 말로 코드를 설명하고, 상대방(고무 오리 등)에게 코드의 각 줄이 어떤 역할을 하는지 구체적으로 설명하게 만든다.

36.3.1 코드를 한 단계씩 쫓아갈 때 도움이 되는 도구 사용하기

스파이더에는 디버거(debugger)가 들어 있다.

- 디버깅을 위해 프로그램을 한 줄씩 실행할 때 디버거가 변수의 값을 표시한다. 그렇지만 그렇게 출력된 변수의 값이 왜 예상과 다른지 알아내는 것은 여전히 여러분의 몫이다.

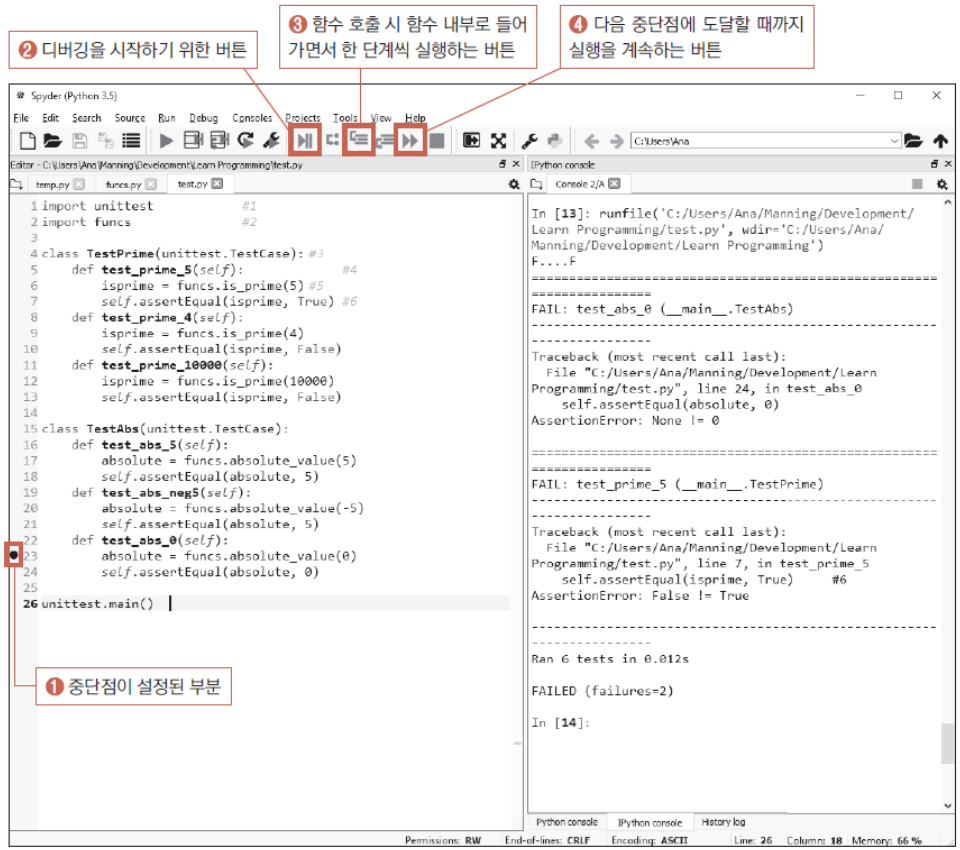

그림 36-1 디버깅 윈도우

36.3.1 코드를 한 단계씩 쫓아갈 때 도움이 되는 도구 사용하기

» 셀프 체크 36.4

36.4 요약

36.4 요약

» (Q36.1) 다음은 버그가 있는 프로그램이다. 이 프로그램을 위한 단위 테스트를 작성하고, 프로그램을 디버깅하라.

```
def remove_buggy(L, e):
    """
    L, 리스트
    e, 임의의 객체
    L에 있는 모든 e를 제거함.
    """
    for i in L:
        if e == i:
            L.remove(i)
```