kim hanwoong 내용보충함. 74c49ff 25 days ago

0 contributors

151 lines (124 sloc) | 5.71 KB

테스트 코드

코드를 짜고 나서 잘 작동하는 코드인지 테스트를 위해서는 테스트 코드를 작성할 필요가 있습니다. 버그를 미연에 방지하고 버전관리시스템에 PullRequest 시 다른 개발자들이 추가 기능에 대해서 안심도 하는 역할을 합니다.

Python 테스트 코드 작성

아래 두 값을 더하는 함수가 있습니다. 만약 숫자만 더해야 한다고 할 때, 어떻게 테스트 코드를 작성하는 지 알아보겠습니다.

```
# runcode.py
def addNum(a,b):
    return a + b
```

테스트 코드는 아래처럼 작성합니다.

```
# testcode.py
import unittest
from runcode import *
class Test_code(unittest.TestCase):
    def test_addNum(self):
        self.assertEqual(addNum(1,2), 3) # 예측값을 적습니다.
        self.assertEqual(addNum(0.1,2), 2.1)

if __name__ == "__main__":
    unittest.main()
```

Test 코드는 터미널에서 아래처럼 실행합니다.

```
$ python testcode.py
```

이번에는 테스트 코드에 숫자대신 문자를 하나 넣어보죠.

```
# testcode.py
import unittest
from runcode import *
class Test_code(unittest.TestCase):
    def test_addNum(self):
        self.assertEqual(addNum(1,2), 3)
        self.assertEqual(addNum(0.1,2), 2.1)
        self.assertEqual(addNum("테",2), 2)

if __name__ == "__main__":
    unittest.main()
```

```
======================================================================
ERROR: test_addNum (__main__.Test_code)
----------------------------------------------------------------------
```

```
Traceback (most recent call last):
  File "testcode.py", line 9, in test_addNum
    self.assertEqual(addNum("테",2), 2)
  File "/Users/woong/test/runcode.py", line 7, in addNum
    return a+b
TypeError: cannot concatenate 'str' and 'int' objects

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
```

에러가 발생했습니다.

a값에 숫자가 올지, 문자가 올지, 리스트가 올지 알 수 없기 때문이죠. a 또는 b 값중 하나에 문자가 들어가면 아래와 같은 타입에러가 발생할 것 입니다.

테스트 코드를 실행하면 위와 같은 메시지가 출력됩니다. 문자를 넣는것을 예상하지 못하고 함수를 작성했기 때문입니다. 이러한 문제는 혼자 작성하는 코드에서는 자주 일어나지 않습니다. 하지만 여러분이 수많은 개발자를 위해서 만든 API 또는 라이브러리를 작성하는 상황이라면 이야기는 달라집니다. 다른 개발자가 함수에 실제로 어떠한 값이을 넣을지 모른다고 가정하고 작성해야합니다.

실무에서는 addNum 함수 하나를 작성하더라도 모든 에러는 아니지만, 예측할 수 있는 에러를 생각하며 작성할 필요는 있습니다. 이렇게 작성하면 거대한 소프트웨어가 작동중에 갑자기 발생하는 에러를 미연에 방지할 수 있습니다. 파이썬은 다이나믹 렝귀지 입니다. 컴파일러가 코드를 일일히 체크하지 않는다는 이야기 입니다. 코드를 실행을 해야 비로소 그 코드에 에러가 있는지 알 수 있답니다. 강력하게 타입을 체크해야하는 상황이라면 아래같은 코드를 작성해야 할 필요가 있습니다. 아래 코드는 편의상 두 값을 더할 수 없다면 에러와 함께 0을 반환하도록 했습니다.

```
#!/usr/bin/env python
# coding:utf-8
# runcode.py
def addNum(a,b):
    """
    addNum 함수는 두 수를 더하는 함수 입니다.
    """
    if not(isinstance(a,int) or isinstance(a,float)):
        return 0, "두 값을 더할 수 없습니다."
    if not(isinstance(b,int) or isinstance(b,float)):
        return 0, "두 값을 더할 수 없습니다."
    return a+b, None
```

물론 상황에 따라서는 아래 코드처럼 try, except를 사용해도 좋습니다. 하지만 여러분이 서버프로그램처럼 정확하게 어디서 에러가 났는지 알고 싶을때는 위처럼 각 상황별로 에러를 체크하는 것이 더 좋을 수 있어요. 만약 a, b 값 둘다 리스트형으로 함수의 인수로 들어와 버리면 try/except로 구현한 코드는 오작동을 하게될 테니까요.

```
#!/usr/bin/env python
# coding:utf8
# runcode.py
def addNum(a,b):
    """
    addNum 함수는 두 수를 더하는 함수 입니다.
    """
    try:
        return a+b, None
    except:
        return 0, "두 값을 더할 수 없습니다."
```

이 함수에 문제가 있는지 없는지를 판단하기 위하여 a 또는 b 값에 int, float 타입이 아닌 다른 type을 넣어서 테스트해 봅시다. 관련 테스트 코드는 아래와 같은 구조를 띄게 됩니다.

```
#!/usr/bin/env python
# coding:utf8
# testcode.py
import unittest
from runcode import *
class Test_code(unittest.TestCase):
```

```
    def test_addNum(self):
        self.assertEqual(addNum(1,2), (3,None))
        self.assertEqual(addNum(0.1,2), (2.1,None))
        self.assertEqual(addNum("테",2), (0,"두 값을 더할 수 없습니다."))
        self.assertEqual(addNum("테","스"), (0,"두 값을 더할 수 없습니다."))
        self.assertEqual(addNum([1,2],[1,2]), (0,"두 값을 더할 수 없습니다.")) # 리스트를 넣었습니다.
        self.assertEqual(addNum({"a":1},(1)), (0,"두 값을 더할 수 없습니다.")) # 딕셔너리와 튜플을 더해보겠습니다.
if __name__ == "__main__":
    unittest.main()
```

테스트가 잘 진행되면 아래와 같은 메시지를 출력합니다.

```
$ python testcode.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

이상으로 파이썬에서 테스트 코드를 작성하는 과정을 알아보았습니다.