Effective Python 54-59

박진성

목차

54. 배포 환경을 구성하는 데는 모듈 스코프 코드를 고려하자

- 배포 환경
  - 프로그램을 실행하는 구성

1. 개발 환경
   - 프로그램을 개발하는 컴퓨터의 환경

2. 제품 환경
   - 프로그램을 실행하는 컴퓨터의 환경

ConfigParser (configparser in python3)

- 설정 파일을 통해 제품 환경을 셋팅할 수 있다.

config formatting

[Section]
option = value

```
[DEV]
TESTING = True
id = test

[REAL]
TESTING = False
id = real
```

```python
import configparser
config = configparser.ConfigParser()
config.read('config.conf')

section = 'REAL'

if config.getboolean(section, 'TESTING'):
    print('TESTING is True')
else:
    print('TESTING is False')

print(config.get(section, 'id'))
```

```
TESTING is False
real
```

```
[DEV]
TESTING = True
id = test

[REAL]
TESTING = False
id = real
```

```python
import sys
print(sys.platform)
```

```
win32
```

핵심 정리

- 개발 환경에 따라 옵션을 다르게하려면 configparser를 사용하자. (모듈 스코프 내에서 동작이 가능하다)

55. 디버깅 출력용으로는 repr 문자열을 사용하자

```
print(5)
print('5')
```

```
5
5
```

```
print('%s' % 5)
```

```
5
```

```
print('%s' % 'test')
print('%r' % 'test')
```

```
test
'test'
```

```
a = '₩x07'
print(a)
print(repr(a))
```

```
•
'₩x07'
```

```
print(eval(repr(a)))
```

```
•
```

```
class TestClass(object):
    pass

obj = TestClass()
print(obj)
```

```
<__main__.TestClass object at 0x019C8150>
```

```
print(obj.__dict__)
```

```
{}
```

```
class TestClass(object):
    def __repr__(self):
        return 'zzzzzzz'

obj = TestClass()
print(obj)
```

```
zzzzzzz
```

56. unittest로 모든 것을 테스트하자

- 파이썬은 정적 타입 검사 기능이 없다.
- 장단점이 있겠지만 생산성은 좋다.
- 파이썬은 unittest로 안정성을 보장해야한다.

unittest

- 파이썬 내장 모듈
- TestCase를 상속 받아서 사용

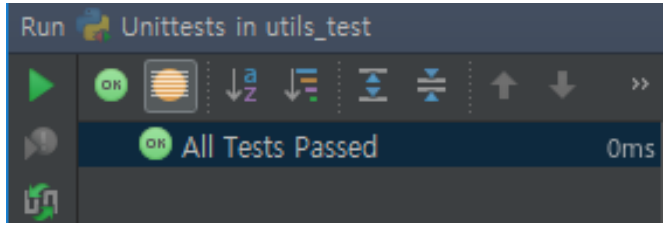
utils.py

```
def to_str(data):
    if isinstance(data, str):
        return data
    elif isinstance(data, bytes):
        return data.decode('utf-8')
    else:
        raise TypeError('Must supply str or bytes, '
                        'found: %r' % data)
```

utils_test.py

```python
from unittest import TestCase, main
from utils import to_str

class UtilsTestCase(TestCase):
    def test_to_str_bytes(self):
        self.assertEqual('hello', to_str(b'hello'))

    def test_to_str_str(self):
        self.assertEqual('hello', to_str('hello'))

    def test_to_str_bad(self):
        self.assertRaises(TypeError, to_str, object())


if __name__ == '__main__':
    main()
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```



pycharm 만세..

setUp & tearDown

- 각 테스트 메소드를 실행하기 전, 후에 실행되는 함수

```
class UtilsTestCase(TestCase):
    def setUp(self):
        print('setUp')

    def tearDown(self):
        print('testDown')

    def test_to_str_bytes(self):
        self.assertEqual('hello', to_str(b'hello'))

    def test_to_str_str(self):
        self.assertEqual('hello', to_str('hello'))

    def test_to_str_bad(self):
        self.assertRaises(TypeError, to_str, object())
```

```
setUp
testDown
setUp
testDown
setUp
testDown
```

고급 활용 (커버리지 보고서, CI, 추가기능)

- nose
  - http://nose.readthedocs.io/en/latest/
- pytest
  - http://docs.pytest.org/en/latest/index.html

핵심 정리

- 파이썬 프로그램을 신뢰할 수 있는 유일한 방법은 테스트 작성
- 내장 모듈 unittest는 좋은 테스트를 작성하는 데 필요한 대부분의 기능을 제공한다
- TestCase를 상속하고 test테서드를 만들 때는 test 라는 단어로 시작해야한다.

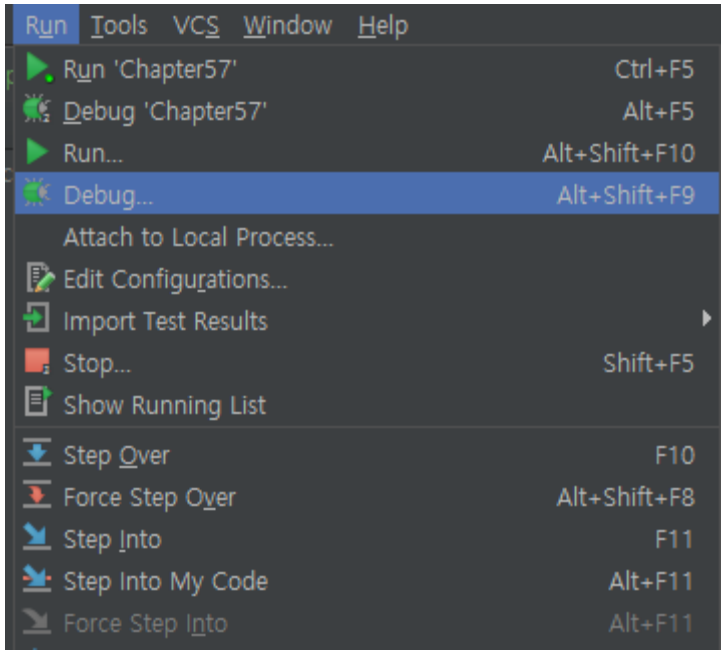
57. pdb를 이용한 대화식 디버깅을 고려하자

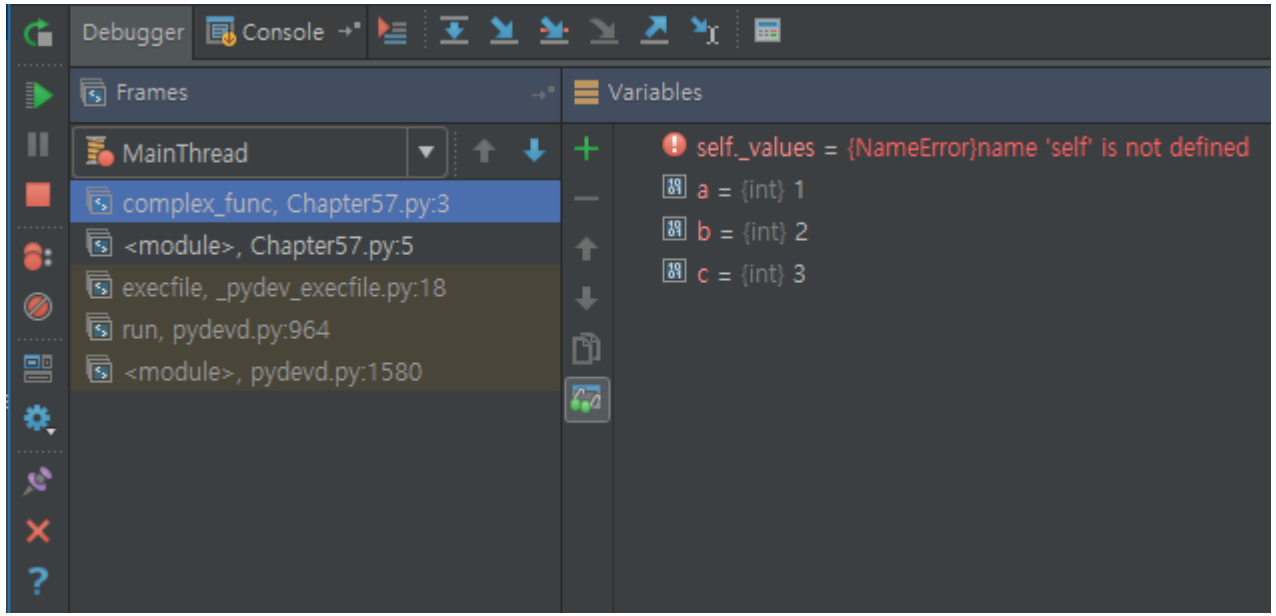
```
def complex_func(a, b, c):
    #...
    import pdb; pdb.set_trace()

complex_func(1, 2, 3)
```

```
-> import pdb; pdb.set_trace()
(Pdb)
```

```
(Pdb) bt
  c:₩users₩jinsung₩pycharmprojects₩effectivepython₩chapter57₩chapter57.py(5)<module>()
-> complex_func(1, 2, 3)
> c:₩users₩jinsung₩pycharmprojects₩effectivepython₩chapter57₩chapter57.py(3)complex_func()->None
-> import pdb; pdb.set_trace()
(Pdb) up
> c:₩users₩jinsung₩pycharmprojects₩effectivepython₩chapter57₩chapter57.py(5)<module>()
-> complex_func(1, 2, 3)
(Pdb) down
> c:₩users₩jinsung₩pycharmprojects₩effectivepython₩chapter57₩chapter57.py(3)complex_func()->None
-> import pdb; pdb.set_trace()
(Pdb) help

Documented commands (type help <topic>):
========================================
EOF    c          d        h         list      q        rv       undisplay
a      cl         debug    help      ll        quit     s        unt
alias  clear      disable  ignore    longlist  r        source   until
args   commands   display  interact  n         restart  step     up
b      condition  down     j         next      return   tbreak   w
break  cont       enable   jump      p         retval   u        whatis
bt     continue   exit     l         pp        run      unalias  where
```

pycharm 만세..


Run Tools VCS Window Help
Run 'Chapter57' Ctrl+F5
Debug 'Chapter57' Alt+F5
Run... Alt+Shift+F10
Debug... Alt+Shift+F9
Attach to Local Process...
Edit Configurations...
Import Test Results
Stop... Shift+F5
Show Running List
Step Over F10
Force Step Over Alt+Shift+F8
Step Into F11
Step Into My Code Alt+F11
Force Step Into Alt+F11


Debugger
Console
Frames
Variables
MainThread
complex_func, Chapter57.py:3
<module>, Chapter57.py:5
execfile, _pydev_execfile.py:18
run, pydevd.py:964
<module>, pydevd.py:1580
self._values = {NameError}name 'self' is not defined
a = {int} 1
b = {int} 2
c = {int} 3

58. 최적화하기 전에 프로파일 하자

- 무턱대고 최적화하지 않고 프로파일을 먼저 하자.
- 파이썬 내장 모듈을 이용하자.
- 순수 파이썬 profile 모듈
- C 확장 모듈 cProfile 모듈
- 순수 파이썬 profile은 성능이 좋지 않아 결과를 왜곡할 수 있을 만큼 부하가 크다.

```
def insertion_sort(data):
    result = []
    for value in data:
        insert_value(result, value)
    return result
```

```
def insert_value(array, value):
    for i, existing in enumerate(array):
        if existing > value:
            array.insert(i, value)
            return
    array.append(value)
```

```
from
max_s
data
test

impor
impor


profi
profiler.runcall(test)

stats = pstats.Stats(profiler)
stats.strip_dirs()
stats.sort_stats('cumulative')
stats.print_stats()
```

```
         20003 function calls in 1.772 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    1.772    1.772 Chapter58.py:18(<lambda>)
        1    0.003    0.003    1.772    1.772 Chapter58.py:1(insertion_sort)
    10000    1.752    0.000    1.769    0.000 Chapter58.py:8(insert_value)
     9989    0.017    0.000    0.017    0.000 {method 'insert' of 'list' objects}
       11    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

ncalls: 프로파일 주기 동안 함수 호출 횟수

tottime: 함수가 실행되는 동안 소비한 초 단위의 시간 (다른 함수 호출을 싱행하는 데 걸린 배제)

curtime: 함수를 싱행하는 데 걸린 초 단위 누적 시간 (다른 함수를 호출하는 데 걸린 시간 포함)

```python
from bisect import bisect_left

def insert_value2(array, value):
    i = bisect_left(array, value)
    array.insert(i, value)
```

```
      30003 function calls in 0.025 seconds

Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    0.025    0.025 Chapter58.py:40(<lambda>)
     1    0.002    0.002    0.025    0.025 Chapter58.py:32(insertion_sort2)
 10000    0.003    0.000    0.023    0.000 Chapter58.py:44(insert_value2)
 10000    0.015    0.000    0.015    0.000 {method 'insert' of 'list' objects}
 10000    0.005    0.000    0.005    0.000 {built-in method _bisect.bisect_left}
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

```
def my_utility(a, b):
    pass
```

```
Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    0.004    0.004 Chapter58.py:69(my_program)
    20    0.003    0.000    0.004    0.000 Chapter58.py:61(first_func)
 20200    0.001    0.000    0.001    0.000 Chapter58.py:58(my_utility)
    20    0.000    0.000    0.000    0.000 Chapter58.py:65(second_func)
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

```
def my_program():
    for _ in range(20):
        first_func()
        second_func()
```

stats.print_callers()

```
   Ordered by: cumulative time

Function                                          was called by...
                                                      ncalls  tottime  cumtime
Chapter58.py:69(my_program)                       <-
Chapter58.py:61(first_func)                       <-      20    0.003    0.004  Chapter58.py:69(my_program)
Chapter58.py:58(my_utility)                       <-   20000    0.001    0.001  Chapter58.py:61(first_func)
                                                         200    0.000    0.000  Chapter58.py:65(second_func)
Chapter58.py:65(second_func)                      <-      20    0.000    0.000  Chapter58.py:69(my_program)
{method 'disable' of '_lsprof.Profiler' objects}  <-
```

핵심 정리

- 최적화 하기 전에 프로파일 하자.
- profile 대신에 cProfile을 사용하자.

59. tracemalloc으로 메모리 사용 현황과 누수를 파악하자

- CPython의 기본 구현은 참조 카운팅으로 메모리를 관리한다.

1. gc

```
import gc
found_objects = gc.get_objects()
print('%d objects before' % len(found_objects))

import waste_memory
x = waste_memory.run()

found_objects = gc.get_objects()
print('%d objects after' % len(found_objects))
for obj in found_objects[:3]:
    print(repr(obj)[:100])
```

```
6051 objects before
17838 objects after
<waste_memory.MyObject object at 0x031B8B70>
<waste_memory.MyObject object at 0x031B8B90>
<waste_memory.MyObject object at 0x031B8BB0>
```

- 문제는 객체가 어떻게 할당되는지 아무런 정보가 없다.
- 문제가 있는 코드를 찾기 어렵다.

2. tracemalloc

```
import tracemalloc
tracemalloc.start(10) # 스택 프레임을 최대 10개까지 저장

time1 = tracemalloc.take_snapshot()
import waste_memory
x = waste_memory.run()
time2 = tracemalloc.take_snapshot()

stats = time2.compare_to(time1, 'lineno')
for stat in stats[:3]:
    print(stat)
```

```
/Users/wwaste_memory.py:29: size=1650 KiB (+1650 KiB), count=29989 (+29989), average=56 B
/Users/wwaste_memory.py:30: size=635 KiB (+635 KiB), count=10000 (+10000), average=65 B
/Users/wwaste_memory.py:35: size=312 KiB (+312 KiB), count=10000 (+10000), average=32 B
```

각 할당 객체의 전체 stack trace

```
stats = time2.compare_to(time1, 'traceback')
top = stats[0]
print('₩n'.join(top.traceback.format()))
```

```
File "C:₩Users₩jinsung₩PycharmProjects₩EffectivePython₩Chapter59₩waste_memory.py", line 29
  self.x = os.urandom(100)
File "C:₩Users₩jinsung₩PycharmProjects₩EffectivePython₩Chapter59₩waste_memory.py", line 35
  obj = MyObject()
File "C:₩Users₩jinsung₩PycharmProjects₩EffectivePython₩Chapter59₩waste_memory.py", line 42
  deep_values.append(get_data())
File "C:/Users/jinsung/PycharmProjects/EffectivePython/Chapter59/using_tracemalloc.py", line 21
  x = waste_memory.run()
```

핵심 정리

- gc 모듈은 어떤 객체가 존재하는지 이해하는 데 도움은 주지만 어떻게 할당 되었는지에 대한 정보는 제공하지 않는다.
- tracemalloc 내장 모듈은 메모리 사용량의 근원을 이해하는 데 필요한 강력한 도구를 제공한다. (3.4 이후 버전만 사용 가능)

끝

- 파이썬 스터디 고생 많으셨습니다!!