Automatické testování softwaru

Petr Pošík, Tomáš Svoboda

Katedra kybernetiky (http://cyber.felk.cvut.cz), FEL ČVUT v Praze

OI (http://oi.fel.cvut.cz), **B4B33RPH (https://cw.fel.cvut.cz/wiki/courses/b4b33rph/start)**: Řešení problémů a hry, 2018-11-13

Předpoklady:

- funkce
- moduly

Testujte svůj kód!

- Nebudete vědět, zda váš kód funguje, dokud jej neotestujete, tj. **dokud se ho nepokusíte použít!**

Obsah

Ukážeme si několik možností, jak testovat vlastní kód. Dostaneme se až do stavu, kdy budeme mít 3 moduly:

- `tools.py` - modul s funkcí `sum_digits()`, kterou budeme chtít testovat,
- `test_tools.py` - modul s testy pro funkci `sum_digits()` a
- `testing.py` - modul obsahující náš vlastní testovací "framework", tj. funkce, které nám usnadní tvorbu a spouštění testů.

Příklad: `sum_digits()`

Specifikace: V modulu `tools.py`, vytvořte funkci `sum_digits(string)`, která vrátí součet všech číslic nalezených v řetězci `string`.

Řešení: Vytvoříme požadovaný modul s níže uvedeným obsahem:

```
%%writefile tools.py
def sum_digits(string):
    """Return the sum of all digits in the string"""
    sum = 0
    for ch in string:
        if ch in '012346789':
            sum += int(ch)
    return sum
```

```
Overwriting tools.py
```

A máme hotovo? Jak víme, že tento kód funguje?

Možnost 1: Zkusíme funkci použít v konzoli Pythonu

```
>>> from tools import sum_digits
>>> sum_digits('1, 2, 3, dee, dah, dee')
```

```
6
```

Výsledek je správně, ale:

- Vyzkoušeli jsme jediný testovací případ.
- Sami musíme vyhodnotit, zda je výsledek správný.
- Co když chceme test spustit znovu (např. poté, co jsme v testované funkci provedli nějakou změnu)?

Možnost 2: Testovací kód vložíme přímo do modulu

Kód, který jsme před chvilkou psali do konzole Pythonu, můžeme napsat přímo do modulu, který chceme testovat (nebo do odděleného modulu).

```
%%writefile tools2.py
def sum_digits(string):
    """Return the sum of all digits in the string"""
    sum = 0
    for ch in string:
        if ch in '012346789':
            sum += int(ch)
    return sum

if __name__ == "__main__":
    # All the code below is executed only when the file is run as a script.
    print(sum_digits('1, 2, 3, dee, dah, dee'))
```

```
Overwriting tools2.py
```

```
import tools2   # "Nothing" happens when we import the module (desired), ...
```

```
%run tools2.py     # ... but the testing code is executed when we run the module!
```

```
6
```

- Stále testujeme jediný případ.
- Stále musíme sami vyhodnotit, zda je vrácený výsledek správný.
- **Ale můžeme test snadno spouštět opakovaně, a to kolikrát chceme!**

Možnost 3: Automatická kontrola správnosti výsledku

Proč jen tisknout výsledek, když můžeme přímo otestovat, zda je správný!?

```
%%writefile tools3.py
def sum_digits(string):
    """Return the sum of all digits in the string"""
    sum = 0
    for ch in string:
        if ch in '012346789':
            sum += int(ch)
    return sum

if __name__ == "__main__":
    observed = sum_digits('1, 2, 3, dee, dah, dee')
    expected = 6
    if observed == expected:
        print('.')
    else:
        print('Test failed.')
        print('- Expected:', str(expected))
        print('- But got: ', str(observed))
```

```
Overwriting tools3.py
```

```
%run tools3.py
```

.

- Stále testujeme jediný případ.
- **Ale nemusíme složitě kontrolovat výsledek testu. Okamžitě vidíme, zda test prošel nebo selhal!**
- **A můžeme test snadno spouštět opakovaně, a to kolikrát chceme!**

Náš vlastní testovací modul!

Kontrola správnosti výsledku se dá extrahovat do funkce, která

- nám umožní psát testy jen s malým množstvím kódu navíc, a
- bude částí modulu, který lze použít opakovaně v mnoha projektech!

Vytvořme modul `testing` s funkcí `assert_equal()`, která bude mít 2 parametry:

- pozorovanou (`observed`) a
- očekávanou (`expected`) hodnotu.

Funkce buď

- vytiskne `"."` a vrátí `True`, když test projde v pořádku, nebo
- vytiskne informativní zprávu a vrátí `False`, pokud test selže.

```
%%writefile testing.py
import sys

def assert_equal(observed, expected):
    """Compare the observed and expected results"""
    if observed == expected:
        print('.', end='')
        return True
    else:
        linenum = sys._getframe(1).f_lineno   # Get the caller's line number.
        print("\nTest at line {} FAILED:".format(linenum))
        print("- Expected:", str(expected))
        print("- But got: ", str(observed))
        return False
```

```
Overwriting testing.py
```

Pomocí našeho modulu `testing`, můžeme upravit modul `tools` následovně:

```
%%writefile tools4.py
from testing import assert_equal

def sum_digits(string):
    """Return the sum of all digits in the string"""
    sum = 0
    for ch in string:
        if ch in '012346789':
            sum += int(ch)
    return sum

if __name__ == "__main__":
    assert_equal(sum_digits('1, 2, 3, dee, dah, dee'), 6)
```

```
Overwriting tools4.py
```

```
%run tools4.py
```

.

- Stále testujeme pouze jediný případ.
- **Ale nemusíme složitě kontrolovat výsledek testu. Okamžitě vidíme, zda test prošel nebo selhal!**
- **A stačí nám trocha kódu, když chceme otestovat jeden případ!**
- **A můžeme test snadno spouštět opakovaně, a to kolikrát chceme!**

Další testovací případy

Máme-li další testovací případy, můžeme je přidat buď do

- sekce `if __name__=="__main__"` vyvíjeného modulu, nebo
- do odděleného testovacího skriptu.

Vytvořme oddělený testovací skript:

```
%%writefile test_tools.py
from testing import assert_equal
from tools4 import *

def test_dee_dah_dee():
    return assert_equal(sum_digits('1, 2, 3, dee, dah, dee'), 6)

def test_empty_string():
    return assert_equal(sum_digits(''), 0)

def test_single_numbers():
    result = True
    for i in range(10):
        num_str = str(i)
        a = assert_equal(sum_digits(num_str), i)
        result = result and a
    return result

# Run the test suite
test_dee_dah_dee()
test_empty_string()
test_single_numbers()
```

```
Overwriting test_tools.py
```

```
%run test_tools.py
```

```
.......
Test at line 14 FAILED:
- Expected: 5
- But got:  0
....
```

Ha! Máme chybu v naší funkci (nebo v našem testovacím kódu)! Dokážete chybu najít?

Automatické nalezení a spuštění testů

Bylo by hezké nemuset všechny testové funkce ručně volat v hlavním programu. Do našeho testového modulu přidáme funkci `run_tests`, která najde všechny funkce začínající na `test_` ve volajícím modulu a všechny je spustí.

```
%%writefile testing2.py
import sys
import inspect

def assert_equal(observed, expected):
    """Compare the observed and expected results"""
    if observed == expected:
        print('.', end='')
        return True
    else:
        linenum = sys._getframe(1).f_lineno   # Get the caller's line number.
        print("\nTest at line {} FAILED:".format(linenum))
        print("- Expected:", str(expected))
        print("- But got: ", str(observed))
        return False

def run_tests():
    caller_globals = sys._getframe(1).f_globals
    results = []
    for symbol, test in caller_globals.items():
        if symbol.startswith('test_'):
            result = test()
            results.append(result)
    print('\n=', len(results), 'tests executed.')
    print('- ', sum(results), 'passed.')
    print('- ', len(results) - sum(results), 'failed.')
```

```
Overwriting testing2.py
```

V našem modulu s testy stačí nyní tuto funkci zavolat, viz poslední řádek.

```
%%writefile test_tools.py
from testing2 import assert_equal, run_tests
from tools4 import *

def test_dee_dah_dee():
    return assert_equal(sum_digits('1, 2, 3, dee, dah, dee'), 6)

def test_empty_string():
    return assert_equal(sum_digits(''), 0)

def test_single_numbers():
    result = True
    for i in range(10):
        num_str = str(i)
        a = assert_equal(sum_digits(num_str), i)
        result = result and a
    return result

# Run the test suite
run_tests()
```

```
Overwriting test_tools.py
```

```
%run test_tools.py
```

```
.......
Test at line 14 FAILED:
- Expected: 5
- But got:  0
....
= 3 tests executed.
-  2 passed.
-  1 failed.
```

Testovací framework: dosažené vlastnosti

- Snadná tvorba obsáhlé sady testů.
- Snadné opakované spouštění testů.
- Snadná (vizuální) kontrola výsledků testů.
- Snadné přidání nových testů.

Další testovací frameworky

Náš modul `testing` není originální nápad. Python obsahuje několik oblíbených testovacích frameworků, např. standardní moduly

- `doctest` a
- `unittest`,

nebo frameworky třetích stran

- nosetest,
- pytest,
- ...

Testování kódu pomocí modulu `doctest`

- Vytvořte si zvyk uvádět příklady použití přímo v docstringu funkcí/tříd/metod.
- Modul `doctest` vám umožní tyto příklady snadno spustit a zkontrolovat jejich výsledky!

```
%%writefile modulewithdoctests.py
def average(x,y):
    """Return the average of 2 numbers.

    >>> average(10,20)
    15.0
    >>> average(1.5, 2.0)
    1.7
    """
    return (x + y) / 2

if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=False)
```

```
Overwriting modulewithdoctests.py
```

Když takový modul spustíte, testy se automaticky spustí a skutečné výsledky se porovnají s těmi očekávanými (také načtenými z docstringu):

```
%run modulewithdoctests.py
```

```
**********************************************************************
File "C:\P\0Teaching\rph\repos\rph-lectures\testing\modulewithdoctests.py", line 6, in __main__.av
erage
Failed example:
    average(1.5, 2.0)
Expected:
    1.7
Got:
    1.75
**********************************************************************
1 items had failures:
   1 of   2 in __main__.average
***Test Failed*** 1 failures.
```

Shrnutí

- Testování vašeho vlastního kódu je **extrémně důležité!**
- Testování validity řešení je **důležitá inženýrská schopnost a dovednost**, nejen při programování!
- Měli byste si osvojit alespoň jeden, ale raději více alternativních způsobů testování kódu.
- Znalost **testovacího frameworku**, ať už jednoduchého (jako je náš `testing`) nebo obsáhlého (jako je např. `unittest`), je nezanedbatelnou **výhodou**!
- Testovací frameworky typu `unittest` jsou běžné v mnoha jazycích. Jsou postaveny na stejné filozofii (xUnit). Naučíte-li se jej používat v jednom jazyce, snadno si jej osvojíte v dalších jazycích.