B4B33RPH: Řešení problémů a hry
Automatické testování softwaru

Petr Pošík

Katedra kybernetiky
ČVUT FEL

Testujte svůj kód!

Nemůžete 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 s naším vlastním testovacím "frameworkem", tj. s funkcemi, které nám usnadní tvorbu a spouštění testů.

Zadání

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

Řešení: Vytvoříme požadovaný modul a doplníme kód:

```
1 def sum_digits(string):
2     """Return the sum of all digits in the string"""
3     sum = 0
4     # Your code here
5     return sum
```

A jsme hotoví? Jak si můžete být jistí, že náš kód funguje?

Jak otestovat kód? 6 / 22

Možnost 1: V konzoli Pythonu

Vyzkoušet kód spustit přímo v konzoli Pythonu:

```
>>> sum_digits('1, 2, 3, dee, dah, dee')
6
>>>
```

Kvíz: Co musím udělat předtím, než tento test provedu?

A Nic, takhle to bude fungovat.

B `import tools`

C `from tools import sum_digits`

D `import tools.sum_digits`

OK, výsledek testu je správně, ale:

- Vyzkoušeli jsme *jediný* testovací případ.
- Sami musíme rozhodnout, zda je výsledek správně.
- Co když chceme test spustit znovu?

Možnost 2: V sekci `__main__` v modulu `tools.py`

Testovací kód vložíme přímo do těla `if __name__ == "__main__"` v modulu `tools.py`:

```
def sum_digits(string):
    """Return the sum of all digits in the string"""
    ...


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'))
```

Předpokládejme, že je funkce `sum_digits()` správně naimplementovaná.
Co se stane, když v Pythonu modul importujeme?

```
>>> import tools
```

A Nastane chyba.

B Nestane se nic.

C Modul se spustí (pokud už předtím nebyl naimportovaný), ale nic viditelného se nestane.

D Modul se spustí a vypíše se číslo 6.

Předpokládejme, že

- funkce `sum_digits()` je správně naimplementovaná,
- interpret Pythonu (`python`) je z příkazového řádku dostupný a
- jsme aktuálně v adresáři, v němž je umístěn modul `tools.py`.

Co se stane, když modul spustíme jako hlavní program, např. z příkazové řádky?

```
$ python tools.py
```

A Nastane chyba.

B Nestane se nic.

C Modul se spustí, ale nic viditelného se nestane.

D Modul se spustí a vypíše se číslo 6.

Jak jsme si pomohli?

- Stále testujeme *jediný* případ.
- Stále musíme sami rozhodnout, zda je výsledek správně.
- **Můžeme test snadno spouštět opakovaně, kolikrát chceme!**

Možnost 3: V samostatném modulu `test_tools.py`

Testovaný kód v modulu `tools.py`:

```
def sum_digits(string):
    """Return the sum of all digits in the string"""
    ...
```

Testovací kód v odděleném modulu `test_tools.py`:

```
from tools import sum_digits

print(sum_digits('1, 2, 3, dee, dah, dee'))
```

Vlastnosti:

- Modul s kódem importujeme pomocí
  ```
  >>> import tools
  ```
  nebo
  ```
  >>> from tools import sum_digits
  ```
- Testy spustíme pomocí
  ```
  $ python test_tools.py
  ```
  Vypíše hodnotu 6.

Jak jsme si pomohli?

- **Pro každý modul máme jediný důvod měnit kód v něm.**
- Stále testujeme *jediný* případ.
- Stále musíme sami rozhodnout, zda je výsledek správně.
- **Můžeme test snadno spouštět opakovaně, kolikrát chceme!**

Automatická kontrola správnosti výsledku

Proč jen tisknout výsledek, když můžeme přímo otestovat, zda je správný?
Testovací kód v modulu `test_tools.py`:

```
from tools import sum_digits

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))
```

Po spuštění modulu s testy vidíme výsledek:

```
$ python test_tools.py
.
```

Jak jsme si pomohli?

- Stále testujeme *jediný* případ.
- **Nemusíme pokaždé složitě rozhodovat, zda je výsledek správný. Okamžitě vidíme, zda test prošel nebo selhal!**
- **Můžeme test snadno spouštět opakovaně, kolikrát chceme!**

Vlastní testovací 'framework'

Kontrola správnosti výsledku se dá extrahovat do funkce, která

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

Vytvoříme modul `testing.py` s funkcí `assert_equal()`:

```
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
```

A modul s testy `test_tools.py` se pak změní na:

```
from tools import sum_digits

assert_equal(sum_digits('1, 2, 3, dee, dah, dee'), 6)
```

Vlastní testovací 'framework' (cont.)

Jak jsme si pomohli?

- Stále testujeme *jediný* případ, **ale je snadné další testovací případy přidat!**
- **Nemusíme pokaždé složitě rozhodovat, zda je výsledek správný. Okamžitě vidíme, zda test prošel nebo selhal!**
- **Můžeme test snadno spouštět opakovaně, kolikrát chceme!**

Více testovacích případů

Otestujme naši funkci důkladněji na více testovacích případech:

- Vytvořme pro ně oddělené funkce se jmény začínajícími na `test\_`.
- Ty mohou sloužit k logickému členění testů a k dokumentaci testů.

Modul s testy `test\_tools.py` se pak změní na:

```
from testing import assert_equal
from tools import sum_digits

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()
```

Více testovacích případů (cont.)

Výstup testovacího skriptu nyní odhalil chybu!

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

Jak jsme si pomohli?

- **Testujeme funkci důkladněji a je snadné další testovací případy přidat!**
- Ale funkce obsahující testy musíme spuštět ručně.
- **Nemusíme pokaždé složitě rozhodovat, zda je výsledek správný. Okamžitě vidíme, zda test prošel nebo selhal!**
- **Můžeme test snadno spouštět opakovaně, kolikrát chceme!**

Automatické vyhledání funkcí s testy

Na konci testovacího skriptu jsme museli všechny testové funkce zavolat ručně.
Vybavíme náš 'framework' funkcí `run\_tests()`, která všechny testy najde a spustí sama!

Do modulu `testing.py` přidáme funkci:

```
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.')
```

Automatické vyhledání funkcí s testy (cont.)

Modul s testy `test\_tools.py` se pak změní na:

```
from testing import assert_equal, run_tests
from tools import sum_digits

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()
```


Automatické vyhledání funkcí s testy (cont.)

Po spuštění testovacího skriptu

```
$ python test_tools.py
```

teď dostaneme výsledek (pokud jsme ještě neopravili chybu v kódu):

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

Díky našemu 'frameworku' umíme

- snadno vytvářet sadu testů,
- snadno opakovaně spouštět celou sadu testů,
- snadno vizuálně zkontrolovat, zda všechny testy prošly nebo nějaký selhal,
- snadno přidávat nové testy.

Další testovací frameworky

Náš modul `testing.py` není originální nápad:

- standardní moduly Pythonu
  - `doctest` a
  - `unittest`,
- nebo frameworky třetích stran
  - pytest,
  - nosetest,
  - ...

Doctest

Modul `doctest` umí najít a spustit testy v docstringu.
Předpokládejme, že modul `tools.py` vypadá např. takto:

```
def sum_digits(string):
    """Return the sum of all digits in the string

    >>> sum_digits('1, 2, 3, dee, dah, dee')
    6
    >>> sum_digits('')
    0
    >>> sum_digits('123456789')
    45
    """
    sum = 0
    for ch in string:
        if ch in '012346789':
            sum += int(ch)
    return sum

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

Kód v sekci `main` najde „volání“ funkce v dokumentačním řetězci, spustí je a výsledky porovná s výsledky uvedenými na dalších řádcích.

Doctest (cont.)

Po spuštění modulu

```
$ python tools.py
```

dostaneme výsledek (pokud jsme ještě neopravili chybu v kódu):

```
**********************************************************************
File "tools_doctest.py", line 8, in __main__.sum_digits
Failed example:
    sum_digits('123456789')
Expected:
    45
Got:
    40
**********************************************************************
1 items had failures:
   1 of   3 in __main__.sum_digits
***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 způsob testování kódu.
- Znalost **testovacího frameworku** (např. `unittest`) je přenositelná i do jiných jazyků.