Jazyk Python - tvorba testů s využitím knihovny Behave

- Pavel Tišnovský
  - ptisnovs@redhat.com

- Slajdy a demonstrační příklady:
  - https://github.com/tisnik/python-programming-courses

Obsah kurzu (1/3)

- Úvod
  - "pyramida" s různými typy testů
  - BDD: behavior-driven development
- Jazyk Gherkin
  - Klauzule when/then
  - Parametry v popisech testů
  - Tabulky
  - Tabulky použité pro specifikaci několika běhů testů
- Praktická část
  - Knihovna Behave
  - Struktura projektu s BDD testy
  - Testování nativních funkcí/knihoven
  - Testování REST API

Úvod

- Problematika testování stále složitějších aplikací a systémů
- CI/CD
- Základní problém
  - čím později je chyba odhalena, tím dražší je její oprava
  - z jiného oboru:
    - triviální úprava ventilu při návrhu motoru
    - vs svolávání aut do servisu
    - vs případné žaloby v případě, že chyba způsobí nehody
- Další časté problémy dnešních aplikací
  - velký vývojářský tým
  - používá se větší množství jazyků (jak se domluvit?)
  - zákazník a jeho role při vývoji
  - někdy nejasné role (vývojář či tester?)

"Pyramida" s různými typy testů

- Business část
  - Beta testy
  - Alfa testy
  - Akceptační testy
- Technologická část
  - UI testy
  - API testy
  - Integrační testy
  - Testy komponent
  - Unit testy
- Další typy testů
  - Benchmarky

BDD: behavior-driven development

- Na pomezí mezi
  - integračními testy
  - API testy
  - akceptačními testy
- Popis očekávaného chování systému z pohledu zákazníka/uživatele
  - samotný systém je z pohledu BDD většinou černá skříňka
- Lze použít pro backend i pro frontend
- Testovací scénáře může psát i poučený zákazník
- Mají i dokumentační funkci

Jazyk Gherkin

- Založen na použití několika klíčových slov a běžných vět
- V určitém ohledu podobný způsob zápisu jako v Pythonu
  - odsazení
  - klíčová slova, nikoli speciální znaky
- Existuje i možnost překladu klíčových slov do jiných jazyků
- Není pevně spojen s žádným konkrétním programovacím jazykem
- Výsledek: mohou ho používat i neprogramátoři

Ukázka jednoduchého testovacího scénáře

```
Given the customer has logged into their current account
  And the balance is shown to be 100 euros
 When the customer transfers 75 euros to their savings account
 Then the new current account balance should be 25 euros
```

Části testovacího scénáře:

- Klíčová slova Given, And, When, Then
- Následuje část věty v libovolném jazyce (praktická je angličtina)
- Ve větě se mohou objevovat proměnné části: 100, 75, 25

Víceřádkový text

```
Feature: Count words function test

  Scenario: Check the function count_words()
    Given a sample text
       """
       Velmi kratka veta.
       """
    When I count all words in text
    Then I should get 3 as a result

  Scenario: Check the function count_words()
    Given a sample text
       """
       Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
       eiusmod tempor incididunt ut labore et dolore magna aliqua.
       """
    When I count all words in text
    Then I should get 19 as a result
```

Tabulky

```
Feature: Sum function test 1

  Scenario: Check the function sum()
    Given a list of integers
      |value |
      | 1    |
      | 10   |
      | 100  |
      | 1000 |
    When I summarize all those integers
    Then I should get 1111 as a result
```

Tabulky (druhý příklad)

```
  Scenario: Check the exchange rate calculation
    Given the following exchange rate table
      | currency |  rate  |
      | CZK      |  1.000 |
      | CAD      | 16.172 |
      | HRK      |  3.407 |
      | USD      | 20.655 |
    When I sell 10 CAD
    Then I should receive 161.72 CZK
```

Tabulky použité pro specifikaci několika běhů testů

```
  Scenario Outline: Check the user search feature, perform the search for more
users
    Given GitHub is accessible
    When I search for user with nick <nick>
    Then I should receive 200 status code
     And I should receive proper JSON response
     And I should find the user with full name <fullname>
     And I should find that the user works for company <company>

     Examples: users
     |nick|fullname|company|
     |torvalds|Linus Torvalds|Linux Foundation|
     |brammool|Bram Moolenaar|Zimbu Labs|
     |tisnik|Pavel Tišnovský|Red Hat, Inc.|
```

Tabulky použité pro specifikaci několika běhů testů

```
  Scenario Outline: Check the exchange rate calculation
    Given the following exchange rate table
      | currency |  rate  |
      | CZK      |  1.000 |
      | CAD      | 16.172 |
      | HRK      |  3.407 |
      | USD      | 20.655 |
    When I sell <sold> <currency>
    Then I should receive <amount> CZK

    Examples: sold
        | sold | currency | amount |
        | 1    |   CZK    |    1.000 |
        | 10   |   CZK    |   10.000 |
        | 1    |   CAD    |   16.172 |
        | 100  |   CAD    | 1617.200 |
        | 2    |   HRK    |    6.814 |
```

Praktická část

- Knihovna Behave
- Struktura projektu s BDD testy
  - testovaný modul
  - testovací scénář
  - implementace testovacího scénáře
  - specifikace prostředí testů

Repositář s demonstračními příklady

- https://github.com/tisnik/python-behave-demos

```
git clone https://github.com/tisnik/python-behave-demos
```

Knihovna Behave

- Knihovna Behave
  - určena pro Python 2.x i Python 3.x
  - implementuje většinu funkcionality jazyka Gherkin
  - snadné napojení popisu testů na jejich implementaci
  - používají se dekorátory
  - automatické odvození parametrů z textu dekorátoru

Struktura projektu s BDD testy

```
├── feature_list.txt
├── features
│   ├── adder.feature
│   └── steps
│       └── common.py
├── requirements.in
├── requirements.txt
├── run_tests.sh
└── src
    └── adder.py
```

Význam souborů v projektu:

```
src/adder.py                     vlastní modul, který budeme chtít otestovat
requirements.in/requirements.txt soubory pro pip (instalátor balíčků)
feature_list.txt                 seznam testovacích scénářů, které se mají spustit
features/                        adresář obsahující testovací scénáře i
implementaci jednotlivých kroků testů
run_tests.sh                     pomocný skript pro spuštění testovacích scénářů
```

Testovaný modul

```
def add(x, y):
    return x + y
```

Popis testovacího scénáře

```
Feature: Adder test

  Scenario: Check the function add()
    Given The function add is callable
    When I call function add with arguments 1 and 2
    Then I should get 3 as a result
```

Implementace jednotlivých kroků testu

- Povšimněte si použití argumentu *context*.

```
from behave import given, then, when
from src.adder import add


@given('The function {function_name} is callable')
def initial_state(context, function_name):
    pass


@when('I call function {function} with arguments {x:d} and {y:d}')
def call_add(context, function, x, y):
    context.result = add(x, y)


@then('I should get {expected:d} as a result')
def check_integer_result(context, expected):
    assert context.result == expected, \
        "Wrong result: {r} != {e}".format(r=context.result, e=expected)
```

Prostředí testů

- Příklad implementace prostředí ve chvíli, kdy se testuje nativní knihovna

```
from behave.log_capture import capture
import ctypes


def _load_library(context, library_name):
    if context.tested_library is None:
        context.tested_library = ctypes.CDLL(library_name)


def before_all(context):
    """Perform setup before the first event."""
    context.tested_library = None
    context.load_library = _load_library
```

Skript pro spuštění testů

```
#!/bin/bash -ex

export NOVENV=1
function prepare_venv() {
    virtualenv -p python3 venv && source venv/bin/activate && python3 `which pip3`
install -r requirements.txt
}

[ "$NOVENV" == "1" ] || prepare_venv || exit 1

PYTHONDONTWRITEBYTECODE=1 python3 `which behave` --tags=-skip -D dump_errors=true
@feature_list.txt $@
```

Vlastní spuštění textu

```
Feature: Adder test # features/adder.feature:1

  Scenario: Check the function add()                # features/adder.feature:3
    Given The function add is callable              # features/steps/common.py:20
0.000s
    When I call function add with arguments 1 and 2 # features/steps/common.py:25
0.000s
    Then I should get 3 as a result                 # features/steps/common.py:30
0.000s

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
```

```
3 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.000s
```

Testování nativních funkcí/knihoven

- Použití knihovny *ctypes*

Testovací scénář

```
  @smoketest
  Scenario: Check the function int add(int, int)
    Given The library libadder.so is loaded
    When I call native function add with arguments 1 and 2
    Then I should get 3 as a result

  Scenario Outline: Thorough checking function int add(int, int)
    Given The library libadder.so is loaded
    When I call native function add with arguments <x> and <y>
    Then I should get <result> as a result

     Examples: results
     |x|y|result|
     # basic arithmetic
     |          0| 0|          0|
     |          1| 2|          3|
     |          1|-2|         -1|
     # no overflows at 16 bit limits
     |      32767| 1|      32768|
     |      65535| 1|      65536|
     # integer overflow
     | 2147483648| 1|-2147483647|
     |-2147483647|-1|-2147483648|
     |-2147483648|-1| 2147483647|
```

Prostředí testů

```
from behave.log_capture import capture
import ctypes


def _load_library(context, library_name):
    if context.tested_library is None:
        context.tested_library = ctypes.CDLL(library_name)


def before_all(context):
    """Perform setup before the first event."""
    context.tested_library = None
    context.load_library = _load_library
```

Implementace kroků testu

```
from behave import given, then, when


@given('The library {library_name} is loaded')
def initial_state(context, library_name):
    context.load_library(context, library_name)


@when('I call native function add with arguments {x:d} and {y:d}')
def call_add(context, x, y):
    context.result = context.tested_library.add(x, y)


@then('I should get {result:d} as a result')
def check_integer_result(context, result):
    assert context.result == result, "Expected result: {e}, returned value:
{r}".format(e=result, r=context.result)
```

Testování REST API

- Použití knihovny *requests*

Testovací scénář

```
  @smoketest
  Scenario: Check the GitHub API entry point
    Given GitHub is accessible
    When I access the API endpoint /
    Then I should receive 200 status code

  Scenario: Check the user search feature
    Given GitHub is accessible
    When I search for user with nick torvalds
    Then I should receive 200 status code
     And I should receive proper JSON response
     And I should find the user with full name Linus Torvalds
     And I should find that the user works for company Linux Foundation
```

Prostředí testů

```
import json
import os.path

from behave.log_capture import capture
import requests


def _is_accessible(context, accepted_codes=None):
    accepted_codes = accepted_codes or {200, 401}
```

```
    url = context.api_url
    try:
        res = requests.get(url)
        return res.status_code in accepted_codes
    except requests.exceptions.ConnectionError as e:
        print("Connection error: {e}".format(e=e))
    return False


def before_all(context):
    """Perform setup before the first event."""
    context.is_accessible = _is_accessible
    context.api_url = "https://api.github.com"


@capture
def before_scenario(context, scenario):
    """Perform setup before each scenario is run."""
    pass


@capture
def after_scenario(context, scenario):
    """Perform cleanup after each scenario is run."""
    pass


@capture
def after_all(context):
    """Perform cleanup after the last event."""
    pass
```

Implementace kroků testu

```
import json

from behave import given, then, when
from urllib.parse import urljoin
import requests


@given('GitHub is accessible')
def initial_state(context):
    assert context.is_accessible(context)


@given('System is running')
def running_system(context):
    """Ensure that the system is accessible."""
    assert is_accessible(context)


@when('I access the API endpoint {url}')
def access_endpoint(context, url):
    context.response = requests.get(context.api_url + url)
```

```
@when('I search for user with nick {nick}')
def search_for_user(context, nick):
    url = urljoin(urljoin(context.api_url, "users/"), nick)
    context.response = requests.get(url)


@then('I should receive {status:d} status code')
def check_status_code(context, status):
    """Check the HTTP status code returned by the REST API."""
    assert context.response.status_code == status


@then('I should receive proper JSON response')
def check_json_response(context):
    content_type = context.response.headers.get('content-type')
    assert content_type.startswith('application/json')
    context.data = context.response.json()


@then('I should find the user with full name {fullname}')
def check_user_full_name(context, fullname):
    assert context.data is not None
    assert 'name' in context.data
    value = context.data.get('name')
    assert value == fullname, "{e} != {v}".format(e=fullname, v=value)


@then('I should find that the user works for company {company}')
def check_company(context, company):
    assert context.data is not None
    assert 'company' in context.data
    value = context.data.get('company')
    assert value == company, "{e} != {v}".format(e=company, v=value)
```

Užitečné odkazy

- Knihovna behave 1.2.6: https://pypi.org/project/behave/
- Welcome to behave!: https://behave.readthedocs.io/en/latest/
- Getting Started with Behavior Testing in Python with Behave https://semaphoreci.com/community/tutorials/getting-started-with-behavior-testing-in-python-with-behave
- What is Gherkin – BDD Language? http://toolsqa.com/cucumber/gherkin/
- Python Quick Reference: http://rgruet.free.fr/#QuickRef
- Python docs: http://www.python.org/doc/
- PEP 8: http://www.python.org/dev/peps/pep-0008/
- pep8.py: http://pypi.python.org/pypi/pep8/
- pylint: http://www.logilab.org/project/pylint
- Epydoc: http://epydoc.sourceforge.net/
- Sphinx: http://sphinx-doc.org/

- Python in Python: http://pypy.org/
- The key differences between Python 2.7.x and Python 3.x with examples: http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html
- Language differences and workarounds: http://python3porting.com/differences.html
- Everything you did not want to know about Unicode in Python 3: http://lucumr.pocoo.org/2014/5/12/everything-about-unicode/
- Unicode (Wikipedia): https://en.wikipedia.org/wiki/Unicode
- Dive Into Python: http://www.diveintopython.net/
- Dive into Python 3: http://www.diveintopython3.net/