

# Biblioteka PyTest

Seminarski rad u okviru kursa  
Verifikacija softvera  
Matematički fakultet

Dijana Zulfikarić, 1087/2019  
dijana.zulfikaric@gmail.com

11. januar 2020

## Sažetak

U seminarskom radu obrađena je *pytest* biblioteka koja služi za testiranje jedinica Python koda. Opisane su prednosti korišćenja ove biblioteke u odnosu na ostale biblioteke iste namene. Detaljnije su opisani koncepti biblioteke i način njene upotrebe. Uz to, naveden je način instalacije i pokretanja testova uz primere korišćenja biblioteke kako bi čitalac mogao da se bliže upozna sa praktičnom upotrebom biblioteke *pytest*.

## Sadržaj

<b>1</b>	<b>Uvod</b>	<b>2</b>
<b>2</b>	<b>Koncepti biblioteke</b>	<b>2</b>
2.1	Pojedinačni testovi . . . . .	2
2.2	Grupisanje testova . . . . .	3
2.3	Fixtures . . . . .	3
<b>3</b>	<b>Primena biblioteke</b>	<b>4</b>
3.1	Instalacija . . . . .	4
3.2	Identifikacija test fajlova i pokretanje . . . . .	4
3.3	Praktični primeri . . . . .	4
3.4	Izvršavanje testa - uspeh i neuspeh . . . . .	7
<b>4</b>	<b>Zaključak</b>	<b>8</b>
	<b>Literatura</b>	<b>9</b>

# 1 Uvod

*pytest* [2] je biblioteka koja omogućuje pisanje kratkih, ali skalabilnih testova biblioteka i aplikacija pisanih u programskom jeziku Python [6].

Kako je ovo biblioteka namenjena za testiranje koda, *pytest* se može koristiti kao alat komandne linije koji automatski pronalazi napisane testove, pokreće ih i izveštava o rezultatima. Za razliku od ostalih biblioteka koje služe istoj svrsi, testovi napisani korišćenjem *pytest* biblioteke ne sadrže neophodni šablonski (eng. *boilerplate*) kod, čime se olakšava korišćenje same biblioteke. Upravo iz tog razloga raste popularnost ove biblioteke u odnosu na ostale biblioteke za iste namene, budući da su testovi napisani korišćenjem *pytest*-a veoma slični klasičnom Python kodu. Naredna prednost ove biblioteke jeste činjenica da može pokrenuti i testove napisane upotrebom drugih biblioteka (kao što su *unittest* [4] i *doctest* [1]), pa se stoga prelaskom na *pytest* ne gubi pređašnji razvoj. Sa druge strane, bitno je naglasiti da testovi pisani u *pytestu* ne mogu da se pokrenu korišćenjem neke druge biblioteke. Još neke od prednosti ove biblioteke su:

- Omogućuje paralelno izvršavanje testova, čime se skraćuje vreme izvršavanja testiranja
- Automatski detektuje test fajlove i funkcije ukoliko oni nisu eksplicitno navedeni
- Omogućuje parametrizovane testove, čime se izbegava bespotrebno dupliranje koda
- Omogućuje izvršavanje samo podskupa testova
- Biblioteka je besplatna za korišćenje i spada u projekte otvorenog koda

Zbog svih navedenih prednosti jednostavna je za upotrebu, čak i bez pređašnjeg iskustva u testiranju koda.

## 2 Koncepti biblioteke

Za razliku od većine biblioteka za testiranje koda, biblioteka *pytest* nije direktno zasnovana na *xUnit*[5] modelu. Razlog tome jeste upravo navedena jednostavnost pisanja testova, bez suvišnog opterećivanja šablonskim kodom. *pytest* ne zahteva učitavanje dodatnih modula u kod, kao ni nasleđivanje klasa za testiranje. Upravo zahvaljujući tome, jedan od glavnih doprinosa ove biblioteke jeste uvođenje načina za testiranje Python koda korišćenjem običnih Python funkcija. Iako nije zasnovan na *xUnit* modelu, *pytest* podržava pisanje testova po njemu.

### 2.1 Pojedinačni testovi

Pisanje testova za pojedinačne funkcije je krajnje jednostavno. Potrebno je ispuniti dva zahteva:

1. Uključivanje *pytest* biblioteke u modul
2. Ime funkcije koja predstavlja test (odnosno koja će biti pokrenuta prilikom testiranja) počinje rečju **test**

Funkcije za testiranje mogu ali ne moraju biti izdvojene u zaseban modul. Radi olakšavanja pisanja testova i postizanja veće preglednosti koda, omogućeno je koristiti Python naredbu *assert*. *assert* je naredba koja prima dva argumenta: uslov koji mora biti ispunjen i opcionu poruku koja će biti ispisana ukoliko uslov (odnosno pretpostavka) nije ispunjen.

## 2.2 Grupisanje testova

Grupisanje testova postiže se korišćenjem markera (Python dekoratora [3]). Ovi markeri imaju sledeću sintaksu: `@pytest.mark.<markername>`. Njihova funkcija je dodavanje skupa različitih osobina i atributa testnim funkcijama. Uz markere definisane u *pytest* biblioteci, korisnici mogu pisati i svoje markere. Navođenje istih markera pre definicije određenog skupa testnih funkcija simulira grupisanje tih funkcija. Tako grupisane funkcije pokreću se izvršavanjem naredne komande:

```
pytest -m <markername> -v
```

Neki od predefinisanih markera su:

- **skip** - test koji se uvek preskače
- **skipif** - test koji se preskače ukoliko je zadovoljen određeni uslov
- **xfail** - ukoliko je određeni uslov zadovoljen generiše se "očekivani neuspeh"
- **parametrize** - omogućuje parametrizaciju argumenata testne funkcije

## 2.3 Fixtures

*Fixtures* su funkcije koje se izvršavaju na početku svake testne funkcije na kojoj su primenjene. Koriste se za podešavanje okruženja za izvršavanje testa, na primer podešavanjem konekcije na bazu podataka, prosleđivanjem URL-ova ili nekih drugih vrsta ulaznih podataka. Ove funkcije omogućavaju izbegavanje pisanje istog bloka koda na početku svakog testa. Definišu se putem markera (dekoratora) i imaju sledeću sintaksu: `@pytest.fixture`. Testna funkcija koja koristi *fixture* funkciju označava se navođenjem imena *fixture* funkcije kao argumenta testne funkcije. Ovim se biblioteci *pytest* daje do znanja da određena testna funkcija koristi implementirani *fixture*, pa ga stoga treba pokrenuti pre samog izvršavanja testne funkcije.

Uvođenjem pojma *fixture* implicitno uvodimo i nove pojmove kao što su *setup* (metoda koja se poziva pre svakog testa i inicijalizuje okruženje za njeno izvršavanje) i *teardown* (metoda koja se poziva nakon svakog testa i deinicijalizuje okruženje u kojem je test izvršen). Druge biblioteke za testiranje Python koda takođe pružaju ovakve apstrakcije, biblioteka *pytest* ih dodatno proširuje. U slučaju korišćenja biblioteke *pytest* može se dodati kod koji se pokreće:

- Na početku i/ili kraju modula za testiranje Python koda  
(*setup\_module/teardown\_module*)
- Na početku i/ili kraju klase koja sadrži testne metode  
(*setup\_class/teardown\_class*)
- Različiti načini strukturiranja klase koja sadrži testne metode  
(*setup/teardown*)
- Pre ili posle poziva testne funkcije  
(*setup\_function/teardown\_function*)
- Pre ili posle poziva testne metode  
(*setup\_method/teardown\_method*)

Ovaj pristup ima i svojih mana. Naime, *fixture* funkcija koja je definisana u testnom fajlu može se koristiti isključivo u njemu. Ukoliko želimo da ove funkcije budu vidljive u ostalim fajlovima potrebno je izdvojiti ih

u poseban fajl sa nazivom `conftest.py`, i zatim ovaj fajl uključivati i koristiti u našim testovima.

## 3 Primena biblioteke

### 3.1 Instalacija

Biblioteka se instalira izvršavanjem naredne naredbe iz komandne linije:

```
pip install pytest
```

### 3.2 Identifikacija test fajlova i pokretanje

Izvršavanjem `pytest` naredbe bez navođenja imena fajla pokreću se svi fajlovi čiji je naziv formata `*_test.py` ili `test_*.py` u tekućem direktorijumu i svim njegovim poddirektorijumima. Biblioteka ove fajlove automatski identifikuje kao napisane testove. Možemo i eksplicitno navesti željeni fajl koji ćemo pokrenuti umesto toga.

`pytest` zahteva da imena testnih funkcija počinje rečju `test`. Ukoliko funkcija nema naziv koji odgovara tome, `pytest` je neće svrstati u grupu funkcija za testiranje. U ovom slučaju ne možemo eksplicitno navesti imena funkcija koje će ući u grupu funkcija za testiranje, već se moramo strogo pridržavati konvencije za imenovanje testnih funkcija.

### 3.3 Praktični primeri

U ovoj sekciji biće navedeni praktični primeri testova pisanih korišćenjem biblioteke `pytest`, počev od najjednostavnijih testova za pojedinačne funkcije, preko primera upotrebe markera, testne klase i *fixture* funkcija.

**Primer 3.1** *Jednostavno testiranje pojedinačnih funkcija*

```
1 import pytest
2
3 def capital_case(word):
4     return word.capitalize()
5
6 def test_capital_case():
7     assert capital_case('word') == 'Word'
```

Listing 1: Testiranje funkcije koja kapitalizuje prvo slovo reči

Uz ovo, možemo dodati i narednu funkciju kako bismo garantovali da će se izbaciti odgovarajući izuzetak:

```
1 def test_raises_exception_on_non_string_arguments():
2     with pytest.raises(TypeError):
3         capital_case(9)
```

Listing 2: Funkcija koja testira izbacivanje izuzetka

### Primer 3.2 *Primer korišćenja skip markera prilikom testiranja*

```
1 import sys
2 import pytest
3
4 if not sys.platform.startswith("win"):
5     pytest.skip("skipping windows-only tests", allow_module_level=
6                 True)
```

Listing 3: Preskakanje testova koji su samo za Windows operativni sistem

### Primer 3.3 *Primer korišćenja skipif markera prilikom testiranja*

```
1 import sys
2 import pytest
3
4 @pytest.mark.skipif(sys.version_info < (3, 6), reason="requires
5                     python3.6 or higher")
6 def test_function():
7     ...
```

Listing 4: Preskakanje testa ukoliko je Python verzija manja od zahtevane

### Primer 3.4 *Primer korišćenja xfail markera prilikom testiranja*

```
1 import sys
2 import pytest
3
4 @pytest.mark.xfail(sys.version_info >= (3, 6), reason="python3.6
5                 api changes")
6 def test_function():
7     ...
```

Listing 5: Indukovanje neuspeha ukoliko je Python verzija manja od zahtevane

### Primer 3.5 *Primer korišćenja parametrize markera prilikom testiranja*

```
1 import pytest
2
3 @pytest.mark.parametrize(
4     "test_input, expected",
5     [("3+5", 8), ("2+4", 6), pytest.param("6*9", 42, marks=pytest.
6     mark.xfail)],
7 )
8 def test_eval(test_input, expected):
9     assert eval(test_input) == expected
```

Listing 6: Parametrizovan test

### Primer 3.6 *Testiranje korišćenjem fixture funkcije*

```
1 import pytest
2
3 @pytest.fixture
4 def smtp_connection():
5     import smtplib
6
7     return smtplib.SMTP("smtp.gmail.com", 587, timeout=5)
8
9
10 def test_ehlo(smtp_connection):
11     response, msg = smtp_connection.ehlo()
12     assert response == 250
```

Listing 7: Korišćenje fixture funkcije za podešavanje smtp okruženja

### Primer 3.7 *Testiranje korišćenjem više fixture funkcija*

```
1 import pytest
2
3 order = []
4
5 @pytest.fixture(scope="session")
6 def s1():
7     order.append("s1")
8
9
10 @pytest.fixture(scope="module")
11 def m1():
12     order.append("m1")
13
14
15 @pytest.fixture
16 def f1(f3):
17     order.append("f1")
18
19
20 @pytest.fixture
21 def f3():
22     order.append("f3")
23
24
25 @pytest.fixture(autouse=True)
26 def a1():
27     order.append("a1")
28
29
30 @pytest.fixture
31 def f2():
32     order.append("f2")
33
34
35 def test_order(f1, m1, f2, s1):
36     assert order == ["s1", "m1", "a1", "f3", "f1", "f2"]
```

Listing 8: Redosled izvršavanja fixture funkcija

### 3.4 Izvršavanje testa - uspeh i neuspeh

Nakon pokrenutog testa postoje dva scenarija - uspešno i neuspešno izvršavanje. Ukoliko je test uspešno izvršen ispisaće se poruka koja sadrži obaveštenje o uspešnom izvršavanju testova, kao i vreme potrebno za izvršavanje. Primer poruke se nalazi u Listingu 3.4:

```
$ pytest test_module.py
.
1 passed in 0.12s [100%]
```

Definišimo sada primer testne funkcije za koju se očekuje neuspešno izvršavanje. Primer se može videti u Listingu 9.

```
1 import pytest
2
3 def func(x):
4     return x + 1
5
6
7 def test_answer():
8     assert func(3) == 5
```

Listing 9: Primer testa za koji se očekuje neuspeh

Nakon pokretanja testa, očekuje se naredni prikaz izveštaja o neuspehu:

```
$ pytest
===== test session starts =====
platform linux -- Python 3.x.y, pytest-5.x.y, py-1.x.y,
pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 1 item

test_sample.py F [100%]

===== FAILURES =====
----- test_answer -----

    def test_answer():
>         assert func(3) == 5
E         assert 4 == 5
E         + where 4 = func(3)

test_sample.py:6: AssertionError
===== 1 failed in 0.12s =====
```

## 4 Zaključak

*pytest* je izuzetno korisna biblioteka za testiranje Python koda. Zahvaljujući svojim mnogobrojnim prednostima (navedenih u poglavlju 1) u odnosu na ostale biblioteke za istu namenu, trenutno je najpopularnija biblioteka koja se koristi u ove svrhe. Jedna od osnovnih karakteristika koju ovu biblioteku izdvaja od drugih jeste jednostavnost njene upotrebe. Naime, jedan od osnovnih koncepata *pytest*-a jeste pisanje testova korišćenjem običnih Python funkcija, bez potrebe za uključivanjem dodatnih modula, nasleđivanjem klasa za testiranje i sličnog šablonskog koda do neophodnog u ostalim bibliotekama za testiranje. Zahvaljujući ovome, korišćenje *pytest* biblioteke je intuitivno za korisnika, pa čak i ako on nije imao predašnjeg iskustva u testiranju. Sa druge strane, biblioteka *pytest* se može koristiti za testiranje složenih softvera, tako da njena jednostavnost ne utiče na slučajeve i spektar upotrebe biblioteke, već povećava performanse izvršavanja izbegavanjem dodatnog šablonskog koda.



## Literatura

- [1] DocTest. on-line at: <https://docs.python.org/2/library/doctest.html>.
- [2] PyTest. on-line at: <http://www.pytest.org/>.
- [3] Python decorators. on-line at: <https://www.python.org/dev/peps/pep-0318/>.
- [4] UnitTest. on-line at: <https://docs.python.org/3/library/unittest.html>.
- [5] xUnit. on-line at: <https://en.wikipedia.org/wiki/XUnit>.
- [6] Python Software Foundation. Python. on-line at: <https://www.python.org/>.