**Unit testing with Python**

Băeșu Rareș-Gabriel, grupa 343

Păunică Cosmin, grupa 343

# Prezentarea PyUnit

PyUnit este un tool integrat în limbajul Python ce facilitează scrierea și rularea de teste unitare. El este alcătuit din două componente: un framework pentru scrierea testelor si un runner.

## Scrierea testelor

Pentru a scrie teste folosind PyUnit, sunt necesari următorii pași:

* Importarea modulului unittest din librăria standard;
* Crearea unei clase de teste care să moștenească clasa unittest.TestCase , care va conține testele sub formă de metode. (Shaw)

A scrie un test constă în a scrie o metodă în clasa descrisă mai sus, în care se apelează metode moștenite de aserție, precum:

* assertEquals(a, b): verifică dacă a == b
* assertTrue(x): verifică dacă bool(x) == true
* assertIn(a, b): verifică dacă a in b

Multe alte metode de aserție sunt disponibile pentru diferite condiții care trebuie să fie testate. Toate acestea, împreună cu cele exemplificate mai sus, au și opuse, precum assertNotEquals(a, b).

De asemenea, există metode ce verifică aruncarea de excepții sau avertismente, printre care:

* assertRaises(exc, fun, \*args, \*\*kwds): verifică dacă fun(\*args, \*\*kwds) aruncă excepția exc
* assertWarns(warn, fun, \*args, \*\*kwds): verifică dacă fun(\*args, \*\*kwds) aruncă warning-ul warn (Python)

## Rularea testelor

Suita de teste scrise într-un fișier Python se poate rula fie apelând unittest.main() în fișier, fie din linia de comandă, cu python -m unittest <nume fișier fără extensie> sau cu python -m unittest discover, care va rula toate fișierele Python din directorul curent a căror denumire începe cu „test”. (Shaw)

## Rularea de teste pentru aplicații Flask

PyUnit poate fi folosit foarte ușor pentru aplicații web scrise cu framework-ul Flask. Este necesară doar scrierea metodei setUp() în clasa de teste, în care să se pornească aplicația în modul de testare, precum în exemplul de mai jos:

|  |
| --- |
| import app # aplicatia Flask  import unittest  class TestRoutes(unittest.TestCase):  def setUp(self):  app.app.testing = True  self.app = app.app.test\_client() |

(Shaw)

# Funcționalitățile PyUnit aplicate pe un API HTTP

Pentru a ilustra modul cum poate fi folosit PyUnit, am dezvoltat un API HTTP, ce se conectează la o bază de date cu 2 tabele principale: BOOKS și AUTHORS, precum și o tabelă asociativă care le leagă pe cele două, BOOKS\_AUTHORS.

|  |
| --- |
| CREATE TABLE books (  id INTEGER PRIMARY KEY AUTOINCREMENT,  title TEXT NOT NULL,  genre TEXT NOT NULL,  publish\_date DATE NOT NULL  );  CREATE TABLE authors (  id INTEGER PRIMARY KEY AUTOINCREMENT,  full\_name TEXT NOT NULL  );  CREATE TABLE books\_authors (  id INTEGER PRIMARY KEY AUTOINCREMENT,  id\_book INTEGER,  id\_author INTEGER,  FOREIGN KEY(id\_book) REFERENCES books(id),  FOREIGN KEY(id\_author) REFERENCES authors(id)  ); |

## Testare funcțională (Preduț, 2022)

Vom prezenta conceptul de testare funcțională pe ruta /books/<id>, cu următoarele metode HTTP:

* metoda PUT: actualizează cartea cu id-ul dat cu datele din body-ul request-ului
* metoda DELETE: șterge din baza de date cartea cu id-ul dat

Spre exemplu, aceasta este funcția care tratează cererea DELETE:

|  |
| --- |
| @books.route('/books/<id>', methods=['DELETE'])  def delete\_book(id):  conn = get\_db\_connection()  curr = conn.execute('DELETE FROM books WHERE id = ?', (id,))  if curr.rowcount == 0:  return {"message": f"book with id {id} does not exist"}  conn.commit()  return {"message": f"successfully deleted book with id {id}"} |

Pentru simplitate, vom considera că ruta /books/<id> este un program de sine stătător și vom partiționa datele de intrare ale acestei rute în clase de echivalență.

Domeniul de intrare:

* o metodă *m*, care poate fi PUT, DELETE sau oricare alta, deci se disting 3 clase de echivalență:
  + M\_1 = {PUT}
  + M\_2 = {DELETE}
  + M\_3 = {*m* | *m* nu este PUT sau DELETE}
* un număr natural *id*, care se poate afla sau nu în baza de date, deci se disting 2 clase de echivalență:
  + ID\_1 = {*id* | cartea cu id-ul *id* se află în baza de date}
  + ID\_2 = {*id* | cartea cu id-ul *id* nu se află în baza de date}
* un body *b* al request-ului, care poate fi sau nu valid, deci se disting 2 clase de echivalență:
  + B\_1 = {*b* | *b* este de forma {new\_title, new\_genre, new\_publish\_date, price, black\_Friday\_price} cu new\_publish\_date o dată validă și price un număr real pozitiv}
  + B\_2 = {*b* | *b* nu se află în B\_1}

Domeniul de ieșire:

* un mesaj care confirmă ștergerea unei cărți
* un mesaj care confirmă actualizarea unei cărți
* un mesaj care arată că id-ul nu există în baza de date
* o excepție cauzată de date invalide
* o excepție cauzată de o metodă nepermisă

Obținem clasele de echivalență globale combinând clasele individuale:

* C\_1 = {(*m*, *id*, *b*) | *m* in M\_1, *id* in ID\_1, *b* in B\_1}, așteptăm mesaj de actualizare
* C\_2 = {(*m*, *id*, *b*) | *m* in M\_2, *id* in ID\_1}, așteptăm mesaj de ștergere
* C\_3 = {(*m*, *id*, *b*) | (*m* in M\_1, *b* in B\_2) SAU (*m* in M\_3)}, așteptăm excepție
* C\_4 = {(*m*, *id*, *b*) | *m* in M\_1 sau M\_2, *id* in ID\_2, *b* in B\_1}, așteptăm mesaj de inexistență a id-uluiAlegem câte o valoare a intrărilor pentru fiecare clasă:

|  |  |  |  |
| --- | --- | --- | --- |
| **Intrări** | | | **Rezultat așteptat** |
| **m** | **id** | **b** |
| PUT | 3 | {"black\_friday\_price": 4, "genre": "Dystopian", "price": 7, "publish\_date": "1949-06-08", "title": "1984"} | Mesaj de actualizare |
| DELETE | 3 | {} | Mesaj de ștergere |
| PUT | 1 | {} | Mesaj de date invalide |
| DELETE | 7 | {"black\_friday\_price": 4, "genre": "Dystopian", "id": 1, "price": 7, "publish\_date": "1949-06-08", "title": "1984"} | Mesaj de inexistență a id-ului |

Următoarea secvență de cod implementează câte un test pentru fiecare dintre aceste clase:

|  |
| --- |
| def test\_case\_1(self):  id = 1  result = self.app.put(f'/books/{id}', data=json.dumps({  "black\_friday\_price": 4,  "genre": "Dystopian",  "price": 7,  "publish\_date": "1949-06-08",  "title": "1984"  }))  self.assertEqual(json.loads(result.data)['message'], f"successfully updated book with id {id}")    def test\_case\_2(self):  id = 1  result = self.app.delete(f'/books/{id}')  self.assertEqual(json.loads(result.data)['message'], f"successfully deleted book with id {id}")  def test\_case\_3(self):  with self.assertRaises(Exception) as cm:  result = json.loads(self.app.put('/books/2').data)    def test\_case\_4(self):  id = 7  result = self.app.delete(f'/books/{id}')  self.assertEqual(json.loads(result.data)['message'], f"book with id {id} does not exist") |

Output-ul testelor, unde cele 4 puncte reprezintă 4 teste trecute cu succes:

|  |
| --- |
| (venv) PS D:\Facultate\Anul III\Testarea sistemelor software\Lab TSS\BooksRestApiUnitTesting> python -m unittest test\_functional  ....  ----------------------------------------------------------------------  Ran 4 tests in 0.190s  OK |

## Mutation testing (Python Package Index, n.d.)

Pentru a testa programul cu mutații, am folosit tool-ul MutPy. Vom exemplifica folosirea lui pe funcția care calculează prețul de Black Friday *pbf* al unei cărți. În mod normal, *pbf* este calculat din prețul original *p* după formula *pbf* = 0,6 \* *p*. MutPy generează mutanți de ordinul 1, schimbând înmulțirea, pe rând, cu împărțirea (/), împărțirea întreagă (//) și ridicarea la putere (\*\*).

Output-ul rulării este următorul:

|  |
| --- |
| [\*] Start mutation process:  - targets: app  - tests: test\_http  [\*] All tests passed:  - test\_http [0.00253 s]  [\*] Start mutants generation and execution:  - [# 1] AOR app.py:68 :  --------------------------------------------------------------------------------  51: def modify\_book(id):  ...  ~68: curr = conn.execute('UPDATE books SET title = ?, genre = ?, publish\_date = ?, price = ?, black\_friday\_price = ? WHERE id = ?', (new\_title, new\_genre, new\_publish\_date, new\_price, new\_price / 0.6, id))  --------------------------------------------------------------------------------  [0.05223 s] killed by test\_http (test\_http.MyTestCase)  - [# 68] AOR app.py:68 :  --------------------------------------------------------------------------------  51: def modify\_book(id):  ...  ~68: curr = conn.execute('UPDATE books SET title = ?, genre = ?, publish\_date = ?, price = ?, black\_friday\_price = ? WHERE id = ?', (new\_title, new\_genre, new\_publish\_date, new\_price, new\_price // 0.6, id))  --------------------------------------------------------------------------------  [0.05113 s] killed by test\_http (test\_http.MyTestCase)  - [# 68] AOR app.py:68 :  --------------------------------------------------------------------------------  51: def modify\_book(id):  ...  ~68: curr = conn.execute('UPDATE books SET title = ?, genre = ?, publish\_date = ?, price = ?, black\_friday\_price = ? WHERE id = ?', (new\_title, new\_genre, new\_publish\_date, new\_price, new\_price \*\* 0.6, id))  --------------------------------------------------------------------------------  [0.05423 s] killed by test\_http (test\_http.MyTestCase)  - [# 68] AOR app.py:68 :  --------------------------------------------------------------------------------  51: def modify\_book(id):  ~52: pass  --------------------------------------------------------------------------------  [0.04220 s] killed by test\_http (test\_http.MyTestCase)  [\*] Mutation score [0.42118 s]: 100%  - all: 4  - killed: 4 (100%)  - survived: 1 (0%)  - incompetent: 0 (0.0%)  - timeout: 0 (0.0%) |

# Bibliografie

Preduț, S. N. (2022). Testarea Sistemelor Software, note de curs.

Python Package Index. (fără an). *MutPy 0.6.1*. Preluat de pe Python Package Index: https://pypi.org/project/MutPy/

Python. (fără an). *unittest - Unit testing framework.* Preluat de pe Python: https://docs.python.org/3/library/unittest.html

Shaw, A. (fără an). *Getting Started With Testing in Python.* Preluat de pe Real Python: https://realpython.com/python-testing/