Tema aleasa: Unit Testing with Python

Membru: Fusneica Florentin-Cristian, grupa 341

Prezentați facilitățile unui tool pentru testarea unitară în Python.

Am ales sa folosesc pytest pentru testarea functionala. Mi s-a parut cel mai prietenos tool de testing, pe departe. Cat timp ai functii cu “test” scris in fata, o sa le ia ca o functie speciala de testing. Dupa ce se strang mai multe functii, poti apela in linia de comanda “pytest tests.py” (unde “tests.py” ar fi fisierul de teste) si iti va trece prin fiecare test. Are si o mica implementare grafica, in care iti arata cate teste a facut, unde se afla in testare, iar la final, cat a luat sa treaca prin toate testele.

Daca un test nu e trecut, iti va da informatii suplimentare pentru motivul pentru care nu a fost trecut.

Totodata, am folosit libraria mutmut pentru generarea mutantilor pentru Mutation Testing.

Ilustrați strategiile de generare de teste pe exemple proprii (create de echipă).

Functia aleasa este algoritmul genetic facut de D4rkia.

Am inceput prin a partitiona datele de intrare, am ajuns la acestea:

```
"""
Determinating genetics_main basic() equivalence classes:

N_POPULATION, N_SELECTED > 0
    N_POPULATION and N_SELECTED > 0
    N_POPULATION or N_SELECTED <= 0

N_POPULATION >= N_SELECTED
    2 classes: N_POPULATION >= N_SELECT and N_POPULATION <
N_SELECT

0 ... 1 < MUTATION_PROBABILITY, i.e  0 <= MUTATION_PROBABILITY <=
1
    MUTATION_PROBABILITY <= 0
    0 < MUTATION_PROBABILITY <= 1 (if > 1, it is as
MUTATION_PROBABILITY == 1)

target's chars must be in genes
    all of targets's chars are in genes
    not all of targets's chars are in genes

debug:
    either true or false
"""
```

De acolo am facut multiple teste care sa identifice daca toate cazurile ar trece mai departe:

Exemple de teste:

```
def test_basic1():
    target_str = "This is a genetic algorithm to evaluate,
combine, evolve, and mutate a string! Also, TSS is amazing!"
    genes_list = list("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.,;!?+-*#@^'è
éòà€ù=)(&%$£/\\")
    N_POPULATION = 200
    N_SELECTED = 50
    MUTATION_PROBABILITY = 0.5
    assert gen(target_str, genes_list, N_POPULATION, N_SELECTED,
MUTATION_PROBABILITY, debug=False)
```

```
def test_basic2():
   target_str = "This is a genetic algorithm to evaluate,
combine, evolve, and mutate a string! Also, TSS is amazing!"
   genes_list = list("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.,;!?+-*#@^'è
éòà€ù=)(&%$£/\\")
   N_POPULATION = 20
   N_SELECTED = 50
   MUTATION_PROBABILITY = 0.5
   with pytest.raises(ValueError):
       assert gen(target_str, genes_list, N_POPULATION,
N_SELECTED, MUTATION_PROBABILITY, debug=False)
```

```
def test_basic3():
    target_str = "This is a genetic algorithm to evaluate,
combine, evolve, and mutate a string! Also, TSS is amazing!"
    genes_list = list("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.,;!?+-*#@^'è
éòà€ù=)(&%$£/\\")
    N_POPULATION = 200
    N_SELECTED = -50
    MUTATION_PROBABILITY = 0.5
    with pytest.raises(ValueError):
        assert gen(target_str, genes_list, N_POPULATION,
N_SELECTED, MUTATION_PROBABILITY, debug=False)
```

```
def test_basic4():
```

```
   target_str = "This is a genetic algorithm to evaluate,
combine, evolve, and mutate a string! Also, TSS is amazing!"
   genes_list = list("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgijklmnopqrstuvwxyz.,;!?+-*#@^'èé
òà€ù=)(&%$£/\\")
   # genes_list has no letter "h"
   N_POPULATION = 200
   N_SELECTED = 50
   MUTATION_PROBABILITY = 0.5
   with pytest.raises(ValueError):
       assert gen(target_str, genes_list, N_POPULATION,
N_SELECTED, MUTATION_PROBABILITY, debug=False)
```

```
def test_basic5():
   target_str = "This is a genetic algorithm to evaluate,
combine, evolve, and mutate a string! Also, TSS is amazing!"
   genes_list = list("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.,;!?+-*#@^'è
éòà€ù=)(&%$£/\\")
   N_POPULATION = 200
   N_SELECTED = 50
   MUTATION_PROBABILITY = -5
   with pytest.raises(ValueError):
       assert gen(target_str, genes_list, N_POPULATION,
N_SELECTED, MUTATION_PROBABILITY, debug=False)
```

```
def test_basic6():
   target_str = "This is a genetic algorithm to evaluate,
combine, evolve, and mutate a string! Also, TSS is amazing!"
   genes_list = list("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.,;!?+-*#@^'è
éòà€ù=)(&%$£/\\")
   N_POPULATION = 200
   N_SELECTED = 50
   MUTATION_PROBABILITY = 0.5
   assert gen(target_str, genes_list, N_POPULATION, N_SELECTED,
MUTATION_PROBABILITY, debug=False)
```

```
def test_basic7(capfd):
   target_str = "This is a genetic algorithm to evaluate,
combine, evolve, and mutate a string! Also, TSS is amazing!"
   genes_list = list("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.,;!?+-*#@^'è
éòà€ù=)(&%$£/\\")
   N_POPULATION = 200
   N_SELECTED = 50
   MUTATION_PROBABILITY = 2
```

```
    gen(target_str, genes_list, N_POPULATION, N_SELECTED,
MUTATION_PROBABILITY, debug=True)

    out, err = capfd.readouterr()

    assert out is not None and out != ""
```

```
def test_basic8(capfd):
    target_str = "This is a genetic algorithm to evaluate,
combine, evolve, and mutate a string! Also, TSS is amazing!"
    genes_list = list("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.,;!?+-*#@^'è
éòà€ù=)(&%$£/\\")
    N_POPULATION = 200
    N_SELECTED = 50
    MUTATION_PROBABILITY = 0.7

    gen(target_str, genes_list, N_POPULATION, N_SELECTED,
MUTATION_PROBABILITY, debug=False)

    out, err = capfd.readouterr()

    assert out is None or out == ""
```

Am ales teste in care sa treaca prin toate tipurile de input-uri si conditii, incat sa acopere toate tipurile de erori.

```
                        8 passed in 2.86s
```

Pentru generarea mutantilor am scris in consola “mutmut run” si am avut 202 teste pe langa cel original:

```
1. Running tests without mutations
⠸ Running...Done

2. Checking mutants
⠼ 8/202  🎉 0  ⏰ 0  🤔 0  🙁 8  🔇 0
```