МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет "Информатика и системы управления"
Кафедра "Системы обработки информации и управления"

Дисциплина "Парадигмы и конструкции языков программирования"

Отчет по лабораторной работе №5
"Модульное тестирование в Python"

Выполнил:
Студент группы ИУ5-35Б
Королев М.О.
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

Задание:

1. Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4
2. Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования
3. Разработайте модульные тесты. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк (не менее 3 тестов)
    - BDD - фреймворк (не менее 3 тестов)

Выбранные фрагменты: ЛР3-4: field, gen_random, unique

https://github.com/koromax/Korolev-PCPL-Labs-2025/tree/main/lab4

Листинг кода:

```python
# lab3/lab_python_fp/field.py
def field(items, *args):
    assert len(args) > 0, "Введите аргументы"

    out = []

    if len(args) == 1:
        out = [e[args[0]] for e in items if args[0] in e.keys()]
    else:
        out = [{arg:e[arg] for arg in e.keys() if arg in args} for e in items if any(arg in e.keys() for
arg in args)]

    return out

if __name__ == "__main__":
    goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'},
    {'color': 'blue'}
    ]

    print(*field(goods, "title"), sep='\n')
    print(*field(goods, "title", "price"), sep='\n')
```

```python
# lab3/lab_python_fp/gen_random.py
import random

def gen_random(n, min, max):
    return [random.randint(min, max) for i in range(n)]

if __name__ == "__main__":
    print(*gen_random(10, 1, 10))
```

```python
# lab3/lab_python_fp/unique.py
class Unique:
    def __init__(self, items, **kwargs):
        Unique._data = items
        Unique._used = set()
        Unique._index = 0
        Unique.ignore_case = kwargs["ignore_case"] if (len(kwargs) > 0 and "ignore_case" in
kwargs.keys()) else False

    def __iter__(self):
        return self

    def __next__(self):
        while self._index < len(self._data):
            item = str(self._data[self._index])
            self._index += 1

            if self.ignore_case and item.lower() in [e.lower() for e in self._used]:
                continue
            if not self.ignore_case and item in self._used:
                continue

            self._used.add(item)
```

```
            return item

        raise StopIteration


if __name__ == "__main__":
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    print(list(Unique(data)))
    print(list(Unique(data, ignore_case=True)))
```

```
# lab4/features/field.feature
Feature: Field extraction from dictionaries

    Scenario: Extract single field
        Given I have a list of dictionaries
        When I extract field "title" from the list
        Then I should get ["Ковер", "Диван для отдыха"]

    Scenario: Extract multiple fields
        Given I have a list of dictionaries
        When I extract fields "title" and "price" from the list
        Then I should get dictionaries with "title" and "price" fields
```

```
# lab4/features/genrandom.feature
Feature: Generate Random Values

    Scenario: Correct generation
        Given I want to generate numbers
        When I generate 100 numbers ranging between 1 and 100
        Then I should get 100 numbers ranging between 1 and 100

    Scenario: Generation in negative range
        Given I want to generate negative numbers
        When I generate 5 negative numbers
        Then I should get 5 negative numbers
```

```
# lab4/features/unique.feature
Feature: Unique

    Scenario: Filter copies
        Given I have a list with duplicates ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        When I filter unique values
        Then I should get ['a', 'A', 'b', 'B']

    Scenario: Filter copies with ignore case
        Given I have a list with duplicates ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        When I filter unique values ignoring case
        Then I should get ['a', 'b']
```

```
# lab4/features/steps/field_steps.py
from behave import given, when, then
import sys
import os
sys.path.append(os.path.join(os.path.dirname(__file__), "..", "..", "..", "lab3", "lab_python_fp"))
```

```python
from field import field

@given("I have a list of dictionaries")
def step_given_list_of_dicts(context):
    context.data = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'},
        {'color': 'blue'}
    ]

@when('I extract field "{field_name}" from the list')
def step_when_extract_single_field(context, field_name):
    context.result = field(context.data, field_name)

@when('I extract fields "{field_name1}" and "{field_name2}" from the list')
def step_when_extract_multiple_fields(context, field_name1, field_name2):
    context.result = field(context.data, field_name1, field_name2)

@then('I should get ["Ковер", "Диван для отдыха"]')
def step_then_get_list(context):
    assert context.result == ["Ковер", "Диван для отдыха"]

@then('I should get dictionaries with "title" and "price" fields')
def step_then_get_dicts(context):
    assert context.result == [
                {'title': 'Ковер', 'price': 2000},
                {'title': 'Диван для отдыха'}
            ]
```

```python
# lab4/features/steps/genrandom_steps.py
from behave import given, when, then
import sys
import os
sys.path.append(os.path.join(os.path.dirname(__file__), "..", "..", "..", "lab3", "lab_python_fp"))
from gen_random import gen_random

@given("I want to generate numbers")
def step_given_gen(context):
    pass

@given("I want to generate negative numbers")
def step_given_gen(context):
    pass

@when('I generate {n} numbers ranging between {mn} and {mx}')
def step_when_gen(context, n, mn, mx):
    context.result = gen_random(int(n), int(mn), int(mx))

@when('I generate {n} negative numbers')
def step_when_filter_unique_ignore_case(context, n):
    context.result = gen_random(int(n), -10, -1)

@then("I should get {n} numbers ranging between {mn} and {mx}")
def step_then_get_list(context, n, mn, mx):
    for e in context.result:
        assert int(mn) <= e <= int(mx)
    assert len(context.result) == int(n)
```

```python
@then("I should get {n} negative numbers")
def step_then_get_list_ignore_case(context, n):
    for e in context.result:
        assert e < 0
```

```python
# lab4/features/steps/unique_steps.py
from behave import given, when, then
import sys
import os
sys.path.append(os.path.join(os.path.dirname(__file__), "..", "..", "..", "lab3", "lab_python_fp"))
from unique import Unique

@given("I have a list with duplicates ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']")
def step_given_list_of_dicts(context):
    context.data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

@when('I filter unique values')
def step_when_filter_unique(context):
    context.result = list(Unique(context.data))

@when('I filter unique values ignoring case')
def step_when_filter_unique_ignore_case(context):
    context.result = list(Unique(context.data, ignore_case=True))

@then("I should get ['a', 'A', 'b', 'B']")
def step_then_get_list(context):
    assert context.result == ['a', 'A', 'b', 'B']

@then("I should get ['a', 'b']")
def step_then_get_list_ignore_case(context):
    assert context.result == ['a', 'b']
```

```python
# lab4/test_tdd.py
import unittest
import sys
import os
sys.path.append(os.path.join(os.path.dirname(__file__), "..", "lab3", "lab_python_fp"))
from field import field
from unique import Unique
from gen_random import gen_random

class TestField(unittest.TestCase):
    def test_single_field_extraction(self):
        data = [
            {'title': 'Ковер', 'price': 2000, 'color': 'green'},
            {'title': 'Диван для отдыха', 'color': 'black'},
            {'color': 'blue'}
        ]
        result = field(data, "title")
        answer = ["Ковер", "Диван для отдыха"]
        self.assertEqual(result, answer)

    def test_multiple_fields_extraction(self):
        data = [
            {'title': 'Ковер', 'price': 2000, 'color': 'green'},
            {'title': 'Диван для отдыха', 'color': 'black'},
            {'color': 'blue'}
```

```
        ]
        result = field(data, "title", "price")
        answer = [
            {'title': 'Ковер', 'price': 2000},
            {'title': 'Диван для отдыха'}
        ]
        self.assertEqual(result, answer)


class TestUnique(unittest.TestCase):
    def test_uniqueness(self):
        data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        result = list(Unique(data))
        self.assertEqual(result, ['a', 'A', 'b', 'B'])

    def test_ignore_case(self):
        data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        result = list(Unique(data, ignore_case=True))
        self.assertEqual(result, ['a', 'b'])


class TestGenRandom(unittest.TestCase):
    def test_correctness(self):
        n = 100
        mn = 1
        mx = 100
        result = gen_random(n, mn, mx)
        for e in result:
            self.assertTrue(mn <= e <= mx)
        self.assertEqual(len(result), n)

    def test_negative_range(self):
        result = gen_random(5, -10, -1)
        for e in result:
            self.assertTrue(e < 0)

unittest.main()
```

Результат выполнения программы:

```
10:Korolev-PCPL-Labs-2025 koromax$ python3 ./lab4/test_tdd.py -v
test_multiple_fields_extraction (__main__.TestField.test_multiple_fields_extraction) ... ok
test_single_field_extraction (__main__.TestField.test_single_field_extraction) ... ok
test_correctness (__main__.TestGenRandom.test_correctness) ... ok
test_negative_range (__main__.TestGenRandom.test_negative_range) ... ok
test_ignore_case (__main__.TestUnique.test_ignore_case) ... ok
test_uniqueness (__main__.TestUnique.test_uniqueness) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.000s

OK
10:Korolev-PCPL-Labs-2025 koromax$ cd lab4; behave -v
Using CONFIGURATION DEFAULTS:
           capture: None
     capture_hooks: True
       capture_log: True
    capture_stderr: True
    capture_stdout: True
             color: auto
       config_tags: None
    default_format: pretty
      default_tags:
           dry_run: False
              jobs: 1
             junit: False
    logging_format: LOG_%(levelname)s:%(name)s: %(message)s
     logging_level: 20
            runner: behave.runner:Runner
scenario_outline_annotation_schema: {name} -- @{row.id} {examples.name}
      show_skipped: True
     show_snippets: True
       show_source: True
      show_timings: True
             stage: None
     steps_catalog: False
           summary: True
tag_expression_protocol: TagExpressionProtocol.AUTO_DETECT
use_nested_step_modules: False
          userdata: {}
USING RUNNER: behave.runner:Runner
Using default path "features"
Trying base directory: /Users/koromax/Documents/Studies/PCPL/Korolev-PCPL-Labs-2025/lab4/features
Feature: Field extraction from dictionaries # features/field.feature:1

  Scenario: Extract single field                    # features/field.feature:3
    Given I have a list of dictionaries             # features/steps/field_steps.py:7 0.000s
    When I extract field "title" from the list      # features/steps/field_steps.py:15 0.000s
    Then I should get ["Ковер", "Диван для отдыха"] # features/steps/field_steps.py:23 0.000s

  Scenario: Extract multiple fields                                 # features/field.feature:8
    Given I have a list of dictionaries                             # features/steps/field_steps.py:7 0.000s
    When I extract fields "title" and "price" from the list         # features/steps/field_steps.py:19 0.000s
    Then I should get dictionaries with "title" and "price" fields # features/steps/field_steps.py:27 0.000s

Feature: Generate Random Values # features/genrandom.feature:1

  Scenario: Correct generation                              # features/genrandom.feature:3
    Given I want to generate numbers                        # features/steps/genrandom_steps.py:7 0.000s
    When I generate 100 numbers ranging between 1 and 100   # features/steps/genrandom_steps.py:15 0.000s
    Then I should get 100 numbers ranging between 1 and 100 # features/steps/genrandom_steps.py:23 0.000s

  Scenario: Generation in negative range      # features/genrandom.feature:8
    Given I want to generate negative numbers # features/steps/genrandom_steps.py:11 0.000s
    When I generate 5 negative numbers        # features/steps/genrandom_steps.py:19 0.000s
    Then I should get 5 negative numbers      # features/steps/genrandom_steps.py:29 0.000s

Feature: Unique # features/unique.feature:1

  Scenario: Filter copies                                                    # features/unique.feature:3
    Given I have a list with duplicates ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'] # features/steps/unique_steps.py:7 0.000s
    When I filter unique values                                              # features/steps/unique_steps.py:11 0.000s
    Then I should get ['a', 'A', 'b', 'B']                                   # features/steps/unique_steps.py:19 0.000s

  Scenario: Filter copies with ignore case                                   # features/unique.feature:8
    Given I have a list with duplicates ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'] # features/steps/unique_steps.py:7 0.000s
    When I filter unique values ignoring case                                # features/steps/unique_steps.py:15 0.000s
    Then I should get ['a', 'b']                                             # features/steps/unique_steps.py:23 0.000s

3 features passed, 0 failed, 0 skipped
6 scenarios passed, 0 failed, 0 skipped
18 steps passed, 0 failed, 0 skipped
Took 0min 0.001s
10:lab4 koromax$
```