Московский государственный технический
Университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»
Отчет по лабораторной работе №3
«Модульное тестирование в Python»

Выполнил:
студент группы ИУ5-36Б
Ковалев Е.А.

Проверил:
Преподаватель каф. ИУ5
Гапанюк Ю.Е.

2024 г.

Описание задания

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).
    - Создание Mock-объектов (необязательное дополнительное задание).

Текст программы

```python
#unique.py

class Unique:
    def __init__(self, items, **kwargs):
        self.items = items
        self.values = set()
        self.ignore_case = kwargs.get('ignore_case', False)

    def __iter__(self):
        return self

    def __next__(self):
        for item in self.items:
            if (not self.ignore_case and item not in self.values) or (self.ignore_case and str(item).lower() not in self.values):
                self.values.add(item if not self.ignore_case else str(item).lower())
                return item
        raise StopIteration
```

```python
#test_unique.py

import unittest
from unique import Unique

class TestUnique(unittest.TestCase):

    def test1(self): # Тестирует на пропуск дупликатов
        items = [1, 2, 2, 3, 1, 4]
        unique = Unique(items)
        result = list(unique)
        self.assertEqual(result, [1, 2, 3, 4])

    def test2(self): # Тестирует IgnoreCase = False
        items = ["apple", "Apple", "banana", "APPLE"]
        unique = Unique(items, ignore_case = False)
        result = list(unique)
        self.assertEqual(result, ["apple", "Apple", "banana", "APPLE"])

    def test3(self): # Тестирует IgnoreCase = True
        items = ["apple", "Apple", "banana", "APPLE"]
        unique = Unique(items, ignore_case = True)
        result = list(unique)
        self.assertEqual(result, ["apple", "banana"])

if __name__ == "__main__":
    unittest.main()
```

```python
#field.py

def field(items, *args):
    assert len(args) > 0
```

```
    for item in items:
        if len(args) == 1:
            res = item.get(args[0])
            if res != None:
                yield res
        else:
            result = {arg: item.get(arg) for arg in args if item.get(arg) != None}
            if result:
                yield result

if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]
    print(list(field(goods, 'title')))
    print(list(field(goods, 'title', 'price')))
```

```
#test_field.py

import pytest
from pytest_bdd import scenario, given, when, then
from field import field

@pytest.fixture
def test_data():
    return [
        {"имя": "Анна", "возраст": 30, "email": "anna@mail.ru"},
        {"имя": "Борис", "возраст": 25, "email": None},
        {"имя": "Владимир", "возраст": None, "email": "vladimir@gmail.com"},
    ]

@pytest.fixture
def context():
    return {}

@scenario('field.feature', 'Выборка по одному полю из массива словарей')
def test_single_field():
    pass

@given('массив словарей')
def data(context, test_data):
    context['data'] = test_data

@when('происходит выборка по имени')
def names(context):
    context['result'] = list(field(context['data'], "имя"))

@then('выводится массив имен')
def check_names(context):
    check = ["Анна", "Борис", "Владимир"]
    assert context['result'] == check, f"Ожидалось {check}, получено {context['result']}"
```

```python
@scenario('field.feature', 'Выборка по нескольким полям из массива словарей')
def test_multiple_fields():
    pass

@given('массив словарей')
def data(context, test_data):
    context['data'] = test_data

@when('происходит выборка по полям "имя" и "возраст"')
def names_ages(context):
    context['result'] = list(field(context['data'], "имя", "возраст"))

@then('выводится массив полей "имя" и "возраст"')
def check_names_ages(context):
    check = [
        {"имя": "Анна", "возраст": 30},
        {"имя": "Борис", "возраст": 25},
        {"имя": "Владимир"}
    ]
    assert context['result'] == check, f"Ожидалось {check}, получено {context['result']}"

@scenario('field.feature', 'Пропуск полей "None"')
def test_field_with_none():
    pass

@given('массив словарей')
def data(context, test_data):
    context['data'] = test_data

@when('происходит выборка по полю "email"')
def emails(context):
    context['result'] = list(field(context['data'], "email"))

@then('выводится массив полей email без значений "None"')
def check_emails(context):
    check = ["anna@mail.ru", "vladimir@gmail.com"]
    assert context['result'] == check, f"Ожидалось {check}, получено {context['result']}"
```

#field.Feature

Feature: Тестирование генератора field

Scenario: Выборка по одному полю из массива словарей
Given массив словарей
When происходит выборка по имени
Then выводится массив имен

Scenario: Выборка по нескольким полям из массива словарей
Given массив словарей
When происходит выборка по полям "имя" и "возраст"
Then выводится массив полей "имя" и "возраст"

Scenario: Пропуск полей "None"
Given массив словарей
When происходит выборка по полю "email"
Then выводится массив полей email без значений "None"

Экранные формы с примерами выполнения программы