**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Радиотехнический»

Кафедра «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования »

Отчет по лабораторной работе №3

«Модульное тестирование в Python.»

|  |  |
| --- | --- |
| Выполнил: |  |
| студент группы  РТ5-31Б |  |
| Ермаков И.А. |  |
|  |  |

Москва, 2024 г

**Описание задания:**

Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4. Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования.

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:**

**Unique.py**

class Unique:

def \_\_init\_\_(self, items, \*\*kwargs):

self.items = iter(items) # Преобразуем входные данные в итератор

self.seen = set() # Множество для отслеживания уникальных элементов

self.ignore\_case = kwargs.get('ignore\_case', False) # Получаем параметр ignore\_case

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

while True:

current = next(self.items) # Получаем следующий элемент из итератора

# Преобразуем в нижний регистр, если ignore\_case=True и элемент — строка

check\_value = current.lower() if self.ignore\_case and isinstance(current, str) else current

# Проверяем, был ли элемент уже обработан

if check\_value not in self.seen:

self.seen.add(check\_value) # Добавляем в seen, чтобы не повторять

return current # Возвращаем исходное значение без изменения регистра

if \_\_name\_\_ == "\_\_main\_\_":

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

print(list(Unique(data)))

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

print(list(Unique(data)))

print(list(Unique(data, ignore\_case=True)))

**Test\_unique.py**

import unittest

from unique import Unique

class TestUnique(unittest.TestCase):

def test\_unique\_integers(self):

data = [1, 1, 2, 2, 3, 3]

unique\_iterator = Unique(data)

result = list(unique\_iterator)

self.assertEqual(result, [1, 2, 3])

def test\_unique\_strings\_case\_sensitive(self):

data = ['a', 'A', 'b', 'B']

unique\_iterator = Unique(data)

result = list(unique\_iterator)

self.assertEqual(result, ['a', 'A', 'b', 'B'])

def test\_unique\_strings\_ignore\_case(self):

data = ['a', 'A', 'b', 'B']

unique\_iterator = Unique(data, ignore\_case=True)

result = list(unique\_iterator)

self.assertEqual(result, ['a', 'b'])

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**Test\_unique\_bdd.py**

from unique import Unique

from pytest\_bdd import given, when, then, scenario

@scenario("unique.feature", "Remove duplicates from a list of integers")

def test\_unique\_integers():

pass

@scenario("unique.feature", "Remove duplicates from a case-sensitive list of strings")

def test\_unique\_strings\_case\_sensitive():

pass

@scenario("unique.feature", "Remove duplicates from a case-insensitive list of strings")

def test\_unique\_strings\_ignore\_case():

pass

@given("a list of integers with duplicates")

def data\_integers():

return [1, 1, 2, 2, 3, 3]

@given("a case-sensitive list of strings with duplicates")

def data\_strings\_case\_sensitive():

return ['a', 'A', 'b', 'B']

@given("a case-insensitive list of strings with duplicates")

def data\_strings\_ignore\_case():

return ['a', 'A', 'b', 'B']

@when("we create a Unique iterator for integers")

def unique\_integers(data\_integers):

return list(Unique(data\_integers))

@when("we create a Unique iterator for case-sensitive strings")

def unique\_strings\_case\_sensitive(data\_strings\_case\_sensitive):

return list(Unique(data\_strings\_case\_sensitive))

@when("we create a Unique iterator for case-insensitive strings")

def unique\_strings\_ignore\_case(data\_strings\_ignore\_case):

return list(Unique(data\_strings\_ignore\_case, ignore\_case=True))

@then("the result should only contain unique integers")

def verify\_unique\_integers(unique\_integers):

assert unique\_integers == [1, 2, 3]

@then("the result should only contain unique case-sensitive strings")

def verify\_unique\_strings\_case\_sensitive(unique\_strings\_case\_sensitive):

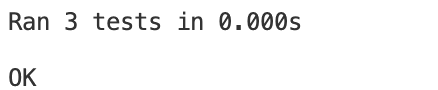
assert unique\_strings\_case\_sensitive == ['a', 'A', 'b', 'B']

@then("the result should only contain unique case-insensitive strings")

def verify\_unique\_strings\_ignore\_case(unique\_strings\_ignore\_case):

assert unique\_strings\_ignore\_case == ['a', 'b']

**Пример выполнения программы**

****