|  |  |  |
| --- | --- | --- |
| Защищено:  Папин А.В..    "\_\_" \_\_\_\_\_\_\_\_\_\_\_2022 г. |  | Демонстрация:  Папин А.В..    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. |

**Отчет по лабораторной работе №5 по курсу**

**базовые компоненты интернет-технологий (БКИТ)**

#### Тема работы: "Модульное тестирование в Python"

31

(количество листов)

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5Ц-54Б Папин Алексей | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Гапанюк Ю.Е. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. |

Москва, МГТУ - 2022

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ ОТЧЕТА**

[1. Цель лабораторной работы 2](#_Toc120623464)

[2. Описание задания. 2](#_Toc120623465)

[3. Листинг программы: 3](#_Toc120623466)

[3.1. Для unittest 3](#_Toc120623467)

[3.2. unique.py 3](#_Toc120623468)

[3.2.1.test.py 4](#_Toc120623469)

[3.3. Для Behave 5](#_Toc120623470)

[3.3.1.check\_unique.feature 5](#_Toc120623471)

[4.1.1.Unique.py 6](#_Toc120623472)

[5. Результаты работы программы: 7](#_Toc120623473)

[5.1. В IDE JetBrains PyCharm 7](#_Toc120623474)

[5.1.1.Unittest 7](#_Toc120623475)

[5.1.2.Behave 7](#_Toc120623476)

[5.2. Через cmd / powershell 8](#_Toc120623477)

[5.2.1.Unittest 8](#_Toc120623478)

[5.2.2.Behave 8](#_Toc120623479)

# **Цель лабораторной работы**

Изучение возможностей возможностей модульного тестирования в языке Python.

# **Описание задания.**

Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4.

1. Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования.
2. 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).

# **Листинг программы:**

# **Для unittest**

# **unique.py**

# Итератор для удаления дубликатов  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 # Нужно реализовать конструктор  
 # В качестве ключевого аргумента, конструктор  
 # должен принимать bool-параметр ignore\_case,  
 # в зависимости от значения которого будут считаться  
 # одинаковыми строки в разном регистре  
 # Например: ignore\_case = True, Aбв и АБВ - разные строки  
 # ignore\_case = False, Aбв и АБВ - одинаковые строки,  
 # одна из которых удалится  
 # По-умолчанию ignore\_case = False  
  
 self.arr = []  
  
 # используя кортежи, получаем ключ и значения  
 for key, value in kwargs.items():  
 # если ключ пустой и значение ИСТИНА, то  
 if key == 'ignore\_case' and value == True:  
 # в текущем списке все символы преобразуем в нижний регистр через функции lower  
 items = [i.lower() for i in items]  
  
 for index in items:  
 # Если текущее значение с списка item не совпадает / не существует в созданном списке arr  
 if index not in self.arr:  
 # то присвоем несуществующее значение в созданном списке arr  
 self.arr.append(index)  
 pass  
  
 def \_\_next\_\_(self):  
 try:  
 x = self.arr[self.begin]  
 self.begin += 1  
 return x  
 except:  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 self.begin = 0  
 return self

# **test.py**

# Подключаем библиотеку unitetest для тестирования  
import unittest  
import math  
  
'''  
assertEqual(self, first, second)  
first - передаваемое значение  
second - полученное значение (в тело функции должен быть return, если вы там не оставили, тогда прописать здесь как None)  
если передаваемое значение совпадает с полученным значением, то тест пройден успешно  
'''  
  
from function.unique import Unique  
  
class test\_unique(unittest.TestCase):  
 # Проверка на чисел  
 def test\_value(self):  
 # Дан список с числами  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 # Получаем уникальные значения и сохраним его в переменной  
 arr\_unique = Unique(data).arr  
 # Проверяем  
 self.assertEqual(  
 arr\_unique,  
 [1, 2]  
 )  
  
 # Проверка на буквы  
 def test\_letters(self):  
 # Дан список с числами  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 # Получаем уникальные значения и сохраним его в переменной  
 arr\_unique = Unique(data).arr  
 # Проверяем  
 self.assertEqual(  
 arr\_unique,  
 ['a', 'A', 'b', 'B']  
 )  
  
 # Проверка на буквы без чувствительного регистра  
 def test\_letters\_ignore\_case(self):  
 # Дан список с числами  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 # Получаем уникальные значения и сохраним его в переменной  
 arr\_unique = Unique(data, ignore\_case = True).arr  
 # Проверяем  
 self.assertEqual(  
 arr\_unique,  
 ['a', 'b']  
 )  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

# **Для Behave**

# **check\_unique.feature**

1. Feature: Calculating and getting unique values  
     
    *# Уникальные значения числового типа  
    # If <CASE> is 1 then is True* Scenario Outline: We get unique values from the list of the contained number  
    Given I have a class of unique values  
    And Getting the list: **<list>** When Finding unique values, case: **<CASE>** Then Output unique values: **<unique>** Examples:  
    | **list** | **unique** | **CASE** |  
    | **[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]** | **[1, 2]** | **0** |  
    | **[1, 3, 1, 1, 1, 3, 2, 2, 2, 2]** | **[1, 3, 2]** | **0** |  
     
     
    *# Уникальные значения символьного типа* Scenario Outline: We get unique values from the list of the contained char  
    Given I have a class of unique values  
    And Getting the list: **<list>** When Finding unique values, case: **<CASE>** Then Output unique values: **<unique>** Examples:  
    | **list** | **unique** | **CASE** |  
    | **['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']** | **['a', 'A', 'b', 'B']** | **0** |  
    | **['a', 'C', 'b', 'B', 'c', 'A', 'b', 'B']** | **['a', 'C', 'b', 'B', 'c', 'A']** | **0** |  
     
     
    *# Уникальные значения символьного типа без чувствительного регистра* Scenario Outline: We get unique values from the list of the contained char ignore\_case  
    Given I have a class of unique values  
    And Getting the list: **<list>** When Finding unique values, case: **<CASE>** Then Output unique values: **<unique>** Examples:  
    | **list** | **unique** | **CASE** |  
    | **['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']** | **['a', 'b']** | **1** |  
    | **['a', 'C', 'b', 'B', 'c', 'A', 'b', 'B']** | **['a', 'c', 'b']** | **1** |  
     
     
    *# Уникальные значения смешанного типа* Scenario Outline: We get unique values from the list of the contained all type  
    Given I have a class of unique values  
    And Getting the list: **<list>** When Finding unique values, case: **<CASE>** Then Output unique values: **<unique>** Examples:  
    | **list** | **unique** | **CASE** |  
    | **['a', 'A', 'b', 'B', '1', '1', '2', '2']** | **['a', 'A', 'b', 'B', '1', '2']** | **0** |  
    | **['a', 'A', 'b', 'B', '1', '1', '2', '2']** | **['a', 'b', '1', '2']** | **1** |

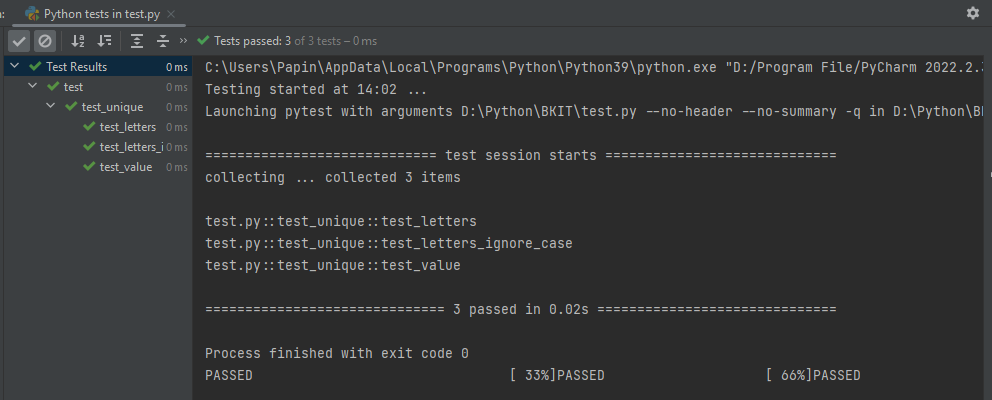
# **Unique.py**

from behave import Given, When, Then  
from function.unique import Unique  
import ast  
  
  
@Given('I have a class of unique values')  
def step\_impl(context):  
 pass  
  
  
@Given("Getting the list: {LIST}")  
def given\_increment(context, LIST):  
 context.LIST = list(ast.literal\_eval(LIST))  
 print(f'Список: {LIST}')  
  
  
@When("Finding unique values, case: {CASE}")  
def given\_increment(context, CASE):  
 check = bool(int(CASE))  
 if (check == True):  
 unique\_list = Unique(context.LIST, ignore\_case=check)  
 else:  
 unique\_list = Unique(context.LIST)  
  
 context.results = unique\_list  
 # print(f'Уникальные значения: {unique\_list}')  
  
  
@Then("Output unique values: {UNIQUE}")  
def then\_results(context, UNIQUE):  
 assert context.results.arr == ast.literal\_eval(UNIQUE)  
 print(f'Уникальные значения: {context.results.arr}')

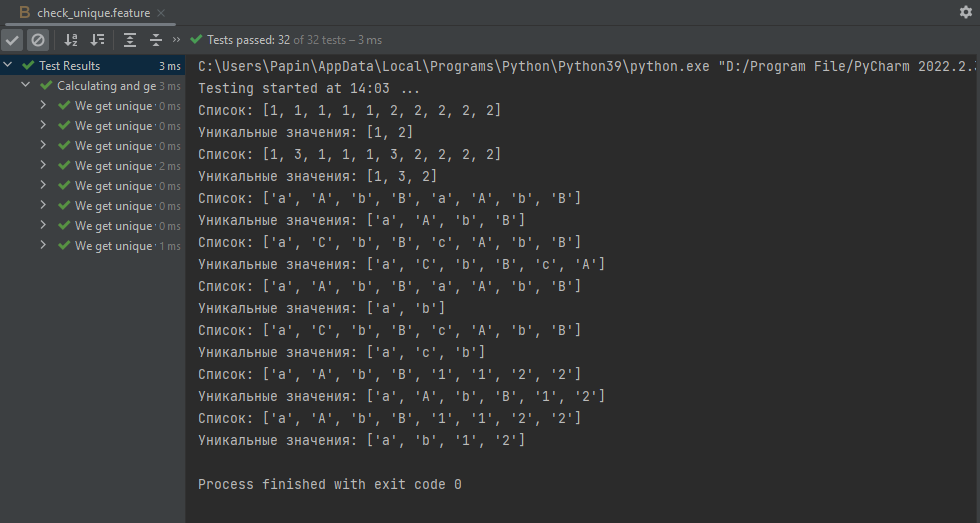
# **Результаты работы программы:**

# **В IDE JetBrains PyCharm**

# **Unittest**

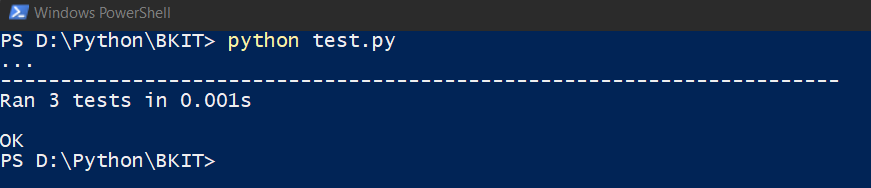


# **Behave**



# **Через cmd / powershell**

# **Unittest**



# **Behave**

