

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
дисциплины «Анализ данных»
Вариант №22

Выполнил:
Михеева Елена Александровна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика
и вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., канд. техн. наук,
доцент, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: Тестирование в Python [unittest].

Цель: приобрести навыки написания автоматизированных тестов на языке программирования Python версии 3.x.

Порядок выполнения работы:

1. Приступили к выполнению индивидуального задания №1. Модифицировано индивидуальное задание из работы №7 дисциплины “Анализ Данных”. Были добавлены тесты, позволяющие проверить операции по работе с базой данных SQLite3.

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import sqlite3
5 from pathlib import Path
6 import gradebook as operations
7 import unittest
8
9
10 class TestStudentDatabase(unittest.TestCase):
11
12     @classmethod
13     def setUpClass(cls):
14         """Set up for class"""
15         print("start testing the gradebook")
16         print("=====")
17
18     @classmethod
19     def tearDownClass(cls):
20         """Tear down for class"""
21         print("successfully")
22         print("=====")
23         print("end of testing\n")
24
25     def setUp(self):
26         self.store_tests = Path("for_tests.db")
27
28     def tearDown(self):
29         if self.store_tests.exists():
30             conn = sqlite3.connect(self.store_tests)
31             conn.close()
32             self.store_tests.unlink()
33
34     def test_create_db(self):
35         """Тестирование создания базы данных."""
36         operations.create_db(self.store_tests)
37
38         conn = sqlite3.connect(self.store_tests)
39         cursor = conn.cursor()
40
41         cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
42         tables = cursor.fetchall()
43         self.assertIn('groups', tables)
44         self.assertIn('students', tables)
45         conn.close()
46
47
48 class Check_student(unittest.TestCase):
49
50     @classmethod
51     def setUpClass(cls):
52         print("start testing adding test")
53         print("=====")
54
55     @classmethod
56     def tearDownClass(cls):
57         """Вызывается один раз для всего класса, в конце."""
58         print("successfully")
59         print("=====")
60         print("tests finished\n")
61
62     def setUp(self):
63         self.store_tests = Path("test_add.db")
64
65     def tearDown(self):
66
67
68     def test_add_student(self):
69         """Метод test_add вызывает после каждого теста для очистки окружения.
70         В данном случае он удаляется от временной базы данных после проведения теста"""
71         if self.store_tests.exists():
72             conn = sqlite3.connect(self.store_tests)
73             conn.close()
74             self.store_tests.unlink()
75
76     def test_add_student(self):
77         """Получить добавление нового человека."""
78         operations.create_db(self.store_tests)
79         operations.add_student(self.store_tests, "A",
80                                "1", (1, 2, 3, 4, 5))
81         conn = sqlite3.connect(self.store_tests)
82         cursor = conn.cursor()
83
84         # Запрос. Сначала получение всех 4-х параметров, после чего отсортируем по
85         # 4-й колонке группы, складируем (Правильнее сказать сортируем) фамилии в 1
86         cursor.execute(
87             """
88             SELECT students.student_name, groups.group_number, students.performance
89             FROM students
90             INNER JOIN groups ON groups.group_id = students.group_id
91             """
92         )
93         student = cursor.fetchone()
94
95         # Проверка на не пустоту, после чего дополнительная проверка содержимого на
96         self.assertIsNotNone(student)
97         self.assertEqual(student[0], "A")
98         self.assertEqual(student[1], "1")
99         self.assertEqual(list(map(int, student[2].split(','))), [1, 2, 3, 4, 5])
100         conn.close()
101
102
103 class Check_All_Selecting(unittest.TestCase):
104
105     @classmethod
106     def setUpClass(cls):
107         print("start testing all select test")
108         print("=====")
109
110     @classmethod
111     def tearDownClass(cls):
112         """Вызывается один раз для всего класса, в конце."""
113         print("successfully")
114         print("=====")
115         print("tests finished\n")
116
117     def setUp(self):
118         self.store_tests = Path("test_all_select.db")
119
120     def tearDown(self):
121
122
123     def test_select_all(self):
124         """Метод test_select_all вызывает после каждого теста для очистки окружения.
125         В данном случае он удаляется от временной базы данных после проведения теста"""
126         if self.store_tests.exists():
127             conn = sqlite3.connect(self.store_tests)
128             conn.close()
129             self.store_tests.unlink()
130
131     def test_select_all(self):
132         """Получить все данные всех записей о людях."""
133         operations.create_db(self.store_tests)
134         # Запрос 3 записей.
135
136         operations.add_student(self.store_tests, "A",
137                                "1", (1, 2, 3, 4, 5))
138         operations.add_student(self.store_tests, "B",
139                                "2", (1, 2, 3, 4, 5))
140         operations.add_student(self.store_tests, "C",
141                                "3", (1, 2, 3, 4, 5))
142         test = operations.select_all(self.store_tests)
143         # Проверка, 2-й элемент полученно, после чего все 3 проверятся на правильность
144         self.assertEqual(len(test), 3)
145         self.assertEqual(test[0][0], "A")
146         self.assertEqual(test[0][1], "1")
147         self.assertEqual(test[0][2], [1, 2, 3, 4, 5])
148         self.assertEqual(test[1][0], "B")
149         self.assertEqual(test[1][1], "2")
150         self.assertEqual(test[1][2], [1, 2, 3, 4, 5])
151         self.assertEqual(test[2][0], "C")
152         self.assertEqual(test[2][1], "3")
153         self.assertEqual(test[2][2], [1, 2, 3, 4, 5])
154
155
156 class Check_FindUnitTest(unittest.TestCase):
157
158     @classmethod
159     def setUpClass(cls):
160         print("start testing find test")
161         print("=====")
162
163     @classmethod
164     def tearDownClass(cls):
165         """Вызывается один раз для всего класса, в конце."""
166         print("successfully")
167         print("=====")
168         print("tests finished\n")
169
170     def setUp(self):
171         self.store_tests = Path("test_find.db")
172
173     def tearDown(self):
174
175
176     def test_find(self):
177         operations.create_db(self.store_tests)
178         operations.add_student(self.store_tests, "A",
179                                "1", (1, 2, 3, 4, 5))
180         operations.add_student(self.store_tests, "B",
181                                "2", (1, 3, 4, 5))
182         operations.add_student(self.store_tests, "C",
183                                "3", (1, 2, 3, 4, 5))
184         test = operations.find(self.store_tests)
185         self.assertEqual(len(test), 2)
186         self.assertEqual(test[0][0], "A")
187         self.assertEqual(test[0][1], "1")
188         self.assertEqual(test[0][2], [1, 2, 3, 4, 5])
189         self.assertEqual(test[1][0], "C")
190         self.assertEqual(test[1][1], "3")
191         self.assertEqual(test[1][2], [1, 2, 3, 4, 5])
192
193
194 if __name__ == "__main__":
195     unittest.main()
```

Рисунок 2. Код индивидуального задания №1

```
UK
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.22 % python3 programs /test.py
start testing all select test
=====
. successfully
=====
tests finished

start testing find test
=====
. successfully
=====
tests finished

start testing adding test
=====
. successfully
=====
tests finished

start testing the gradebook
=====
. successfully
=====
end of testing

Ran 4 tests in 0.044s

OK
```

Рисунок 3. Результат работы программы задания №1

Ответы на контрольные вопросы.

1) Для чего используется автономное тестирование?

Ответ: автономное тестирование используется для автоматизации проверки работоспособности программного обеспечения, позволяя выполнять тесты без необходимости ручного вмешательства.

2) Какие фреймворки Python получили наибольшее распространение для решения задач автономного тестирования?

Ответ: unittest, pytest и nose – наиболее популярные фреймворки Python.

3) Какие существуют основные структурные единицы модуля unittest?

Ответ: основные структурные единицы модуля unittest – тестовые классы, методы тестирования (Начинаются на test_) и assertIn для проверки ожидаемых результатов.

4) Какие существуют способы запуска тестов unittest?

Ответ: тесты unittest можно запускать как из командной строки, так и из среды разработки (Visual Studio Code, PyCharm, Visual Studio и т.д.), а также можно использовать специализированные инструменты для автоматического запуска тестов (Cricket).

5) Каково назначение класса TestCase?

Ответ: класс TestCase определяет тестовые случаи, которые содержат методы для проверки конкретных аспектов работы программы.

6) Какие методы класса TestCase выполняются при запуске и завершении работы тестов?

Ответ: при запуске и завершении работы тестов запускаются методы setUp() и tearDown().

7) Какие методы класса TestCase используются для проверки условий и генерации ошибок?

Ответ: методы assertIn() используются для проверки различных условий и генерации ошибок в случае несоответствия ожидаемых и получаемых результатов.

8) Какие методы класса TestCase позволяют собирать информацию о самом тесте?

Ответ: методы setUp(), tearDown(), getName() и id() позволяют собирать информацию о самом тесте, его имени и идентификаторе.

9) Каково назначение класса TestSuite? Как осуществляется загрузка тестов?

Ответ: класс TestSuite группирует тестовые случаи в единую структуру. Загружаются тесты путём добавления тестовых случаев в объект TestSuite, который выполняется после этого.

10) Каково назначение класса TestResult?

Ответ: класс TestResult отслеживает результаты выполнения тестов. В эту информацию включены сведения о пройденных, проваленных и пропущенных тестах.

11) Для чего может понадобиться пропуск отдельных тестов?

Ответ: пропуск отдельных тестов может понадобиться, когда тестируемый код, временно недоступен или в процессе разработки\доработки.

12) Как выполняется безусловный и условных пропуск тестов? Как выполнить пропуск класса тестов?

Ответ: безусловный пропуск теста выполняется с помощью декоратора @unittest.skip(), а условный пропуск с помощью декоратора @unittest.skipIf(). Для пропуска класса тестов можно использовать декоратор @unittest.skip() перед определением класса.

13) Обобщённый алгоритм проведения тестирования с помощью PyCharm?

Ответ: обобщённый алгоритм проведения тестирования с помощью PyCharm включает: создание тестовых случаев и их методов; настройка окружения тестирования; написание необходимых тестов (И их запуск через PyCharm); анализ результатов и исправление выявленных проблем.

Вывод: в ходе выполнения лабораторной работы, исследовано взаимодействие с базами данных SQLite3 с помощью языка программирования Python.