Домашнее задание
по теме «Тестирование кода. Unit-тесты»

Формулировка задания

Реализовать 2 задачи на python и написать к каждой набор модульных тестов. Использовать библиотеку pytest или unittests. Запустить покрытие тестами для задачи и оценить процент покрытия.

Результирующий код должен быть читаемым, с единой системой отступов и адекватными названиями переменных.

Описание плана работы

Реализовать 2 задачи на python и написать к каждой набор модульных тестов. Использовать библиотеку pytest или unittests. Запустить покрытие тестами для задачи и оценить процент покрытия.

Желательна реализация в файлах py. Сохранить задачи (сделать коммиты для каждой) в локальном git и опубликовать в удаленном репозитории.

Для отчета по работе выполнить задание в файле.py или .ipynb. Сделать снимки экрана корректного выполнения программы в IDE.

Задача 1 Тестирование методов класса

Разработать набор модульных тестов для класса BooksCollector.

Класс BooksCollector содержит следующие свойства и методы:

- Словарь books_genre — Название книги: Жанр книги.
- Список favorites — избранные книги.
- Список genre — доступные жанры.
- Список genre_age_rating — жанры с возрастным рейтингом.
- Методы для работы со словарем books_genre и списком favorites:
    - add_new_book — добавляет новую книгу в словарь без указания жанра.
    - set_book_genre — устанавливает жанр книги
    - get_book_genre— выводит жанр книги по её имени.
    - get_books_with_specific_genre— выводит список книг с определённым жанром.
    - get_books_genre— выводит текущий словарь books_genre.

- get_books_for_children — возвращает книги, которые подходят детям.
- add_book_in_favorites — добавляет книгу в избранное.
- delete_book_from_favorites — удаляет книгу из избранного.
- get_list_of_favorites_books — получает список избранных книг.

Для реализации модульного тестирования:

- Напишите модульные тесты для каждого метода класса BooksCollector.
- Используйте assert для проверки ожидаемых результатов.
- Проверьте различные сценарии использования методов, включая позитивные и негативные случаи.
- Используйте параметризацию для тестирования различных входных данных, где это уместно.
- Добавьте комментарии к тестам для объяснения их цели и ожидаемых результатов.
- Запустите все тесты и проверьте покрытие тестами.

Пример проверки методов без модульных тестов:

```
# Пример использования класса
collector = BooksCollector()
# Добавление книг
collector.add_new_book('Властелин колец')
collector.add_new_book('Гарри Поттер')
collector.add_new_book('Матрица')
# Установка жанров
collector.set_book_genre('Властелин колец', 'Фантастика')
collector.set_book_genre('Гарри Поттер', 'Фэнтези')
collector.set_book_genre('Матрица', 'Научная фантастика')
# Получение жанров
print(collector.get_book_genre('Властелин колец'))  # Фантастика
print(collector.get_book_genre('Гарри Поттер'))  # Фэнтези
print(collector.get_book_genre('Матрица'))  # Научная фантастика
```

```
# Получение книг определенного жанра

fantasy_books = collector.get_books_with_specific_genre('Фантастика')

print(f'Книги в жанре Фантастики: {fantasy_books}')

# Получение всех книг жанра

all_books = collector.get_books_genre()

print('Все книги и их жанры:', all_books)

# Получение книг для детей

children_books = collector.get_books_for_children()

print('Книги для детей:', children_books)

# Добавление в избранное

collector.add_book_in_favorites('Властелин колец')

# Удаление из избранного

collector.delete_book_from_favorites('Гарри Поттер')

# Получение списка избранных

favorites_list = collector.get_list_of_favorites_books()

print('Список избранных книг:', favorites_list)
```

Задача 2 Работа с “подменными” объектами

Используя библиотеку для создания моков, напишите модульные тесты для проверки функциональности формы оплаты на сайте. Вместо реальной кредитной карты используйте мок-объект.

Объекты для платформы оплаты:

1. Создайте класс CreditCard с методами getCardNumber(), getCardHolder(), getExpiryDate(), getCvv(), charge(amount: float).
2. Создайте класс PaymentForm с методом pay(amount: float).

Для реализации модульного тестирования:

1. В тестовом классе, создайте мок-объект для класса CreditCard.
2. Определите поведение мок-объекта.
3. Создайте объект класса PaymentForm, передайте ему мок-объект в качестве аргумента.

4. Вызовите метод pay() и убедитесь, что мок-объект вызывает метод charge()
5. Реализуйте позитивные и негативные тесты для полной проверки созданных объектов.
6. Проверьте какое покрытие у разработанных вами тестов.

Пример проверки методов без модульных тестов:

```
# Получение данных карты

print(f"Карта: {card.get_card_number()}")

print(f.Владелец: {card.get_card_holder()}")

print(f"Срок действия: {card.get_expiry_date()}")

print(f"CVC: {card.get_cvv()}")

# Обработка платежа

amount = 100.00

result = payment_form.pay(amount)

print(result)

# Попытка чрезмерного списания

try:

    card.charge(1500.00)  # Это вызовет исключение

except Exception as e:

    print(f"Ошибка при попытке чрезмерного списания: {e}")

# Успешный платеж

result = payment_form.pay(50.00)

print(result)
```

Перечень необходимых инструментов

- Python
- pytest
- unittests
- venv
- Jupiter Notebook
- IDE VS Code
- GigaIDE

Форма предоставления результата

1. В поле ссылки загрузить ссылку на удаленный репозиторий с доступом для наставника.
2. В поле файла загрузить архив с папкой, в которой разместить отчет со скриншотами по заданиям и решение задач 1-4. Решения должны быть представлены в формате .ipynb или .py.

Шкала оценивания

- 1.0 – отлично
- 0.7–0.9 – хорошо
- 0.5–0.6 – удовлетворительно
- Менее 0.5 – задание не выполнено