Звіт

З лабораторної роботи №2

Студента групи МІТ-31

Добровольського Арсенія Михайловича

Варіант №9

**Тема роботи:** Тестування програмного забезпечення

**Мета роботи:** розглянути принципи модульного тестування програмного забезпечення; навчитися створювати модульні тести з використанням unittest.

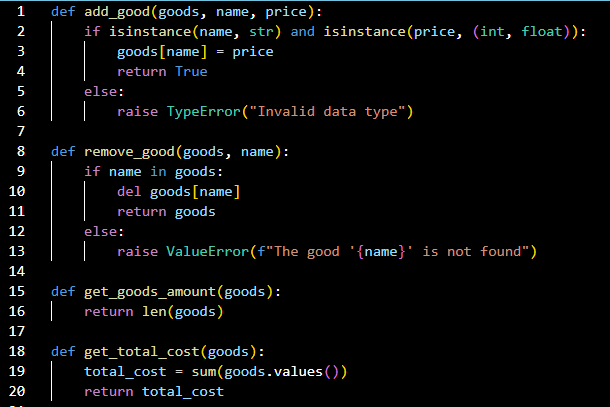
**Посилання на Github репозиторій**: <https://github.com/arsgooo/pt2023>

Пояснення до коду

Створюю два файли: один містить код основної програми (тобто функції, що реалізовують облік речей ломбарду), а інший – методи тестування кожної з цих функцій.

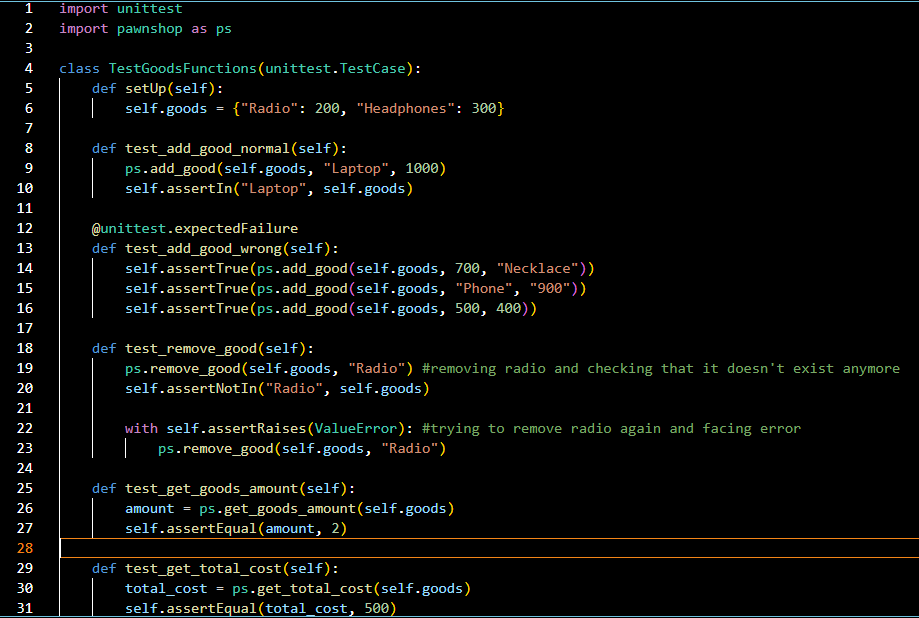
Структура даних для зберігання елементів – словник, в якому ключ – це назва предмету (string), а значення – його ціна (int/float).

Файл основної програми має вигляд:



* add\_good – додає новий предмет, при цьому перевіряє формат введених даних, щоб назва предмету була строго рядком, а ціна – числом. Інакше виникає помилка TypeError, яка сповіщає про неправильний формат даних, внаслідок чого такий предмет відповідно не може бути додано.
* remove\_good – видаляє предмет за ключем (тобто за його назвою). Якщо предмета з такою назвою не існує або його вже було видалено раніше, то виникає помилка ValueError, яка повідомляє, що предмет з таким іменем не знайдено.
* get\_goods\_amount – дозволяє отримати поточну кількість речей у «ломбарді».
* get\_total\_cost – повертає загальну вартість усіх речей, які нині зберігаються в «ломбарді».

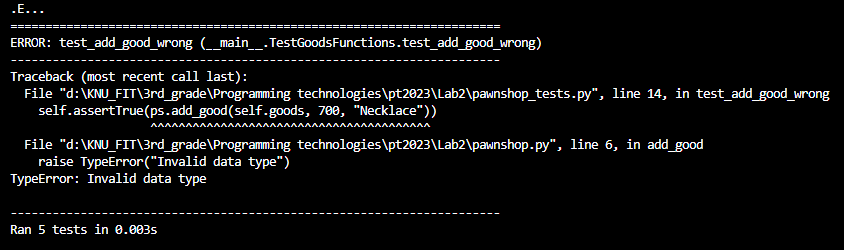
Розгляньмо файл з тестами:



* setUp – метод, що автоматично викликається перед кожним тестовим методом. В цьому коді його суть полягає в ініціалізації словника, дані якого буде використано для тестування. Це необхідно для того, щоб усі тестові методи працювали в однакових умовах і в разі чого можна було легко визначити, які саме тести провалюються.
* test\_add\_good\_normal – тестує функцію add\_good за умови правильного формату введених даних, і перевіряє наявність даного предмету у словнику.
* test\_add\_good\_wrong – так само тестує функцію add\_good, проте, на відміну від попередньої функції, робить це з неправильними наборами вхідних даних. Декоратор @unittest.expectedFailure використовується для уникнення повідомлень про помилки при виконанні тесту. Таким чином, ми ніби «передбачаємо», що цей тест буде провалено і вказуємо інтерпретатору, щоб він сприйняв помилку, як очікувану і не відображав її у вікні терміналу.

Порівняємо результати тестування з використанням декоратора і без:

Без:



З:



* test\_remove\_good – тестує функцію видалення предмету і перевіряє, що він був дійсно видалений. Оператор with означає, що видалення неіснуючого предмету спричиняє помилку ValueError. Тут явно вказано те, що помилка повинна бути викликана. Отже, тест буде пройдено успішно, тому використання декоратора @unittest.expectedFailure в цьому методі не має сенсу.
* test\_get\_goods\_amount – тестує функцію отримання поточної кількості предметів.
* test\_get\_total\_cost – тестує функцію отримання загальної вартості предметів.

**Висновок:** в ході виконання лабораторної роботи ми розглянули принципи модульного тестування програмного забезпечення та на практиці засвоїли створення модульних тестів з використанням Python-фреймворку unittest.