Soda Machine Unit Testing

The purpose of this project is to write and run automated Unit Tests on a code base.

Create a separate .py test file for each Class being tested. Be sure to import the **unittest** module along with any other classes/modules from the SodaMachine project that will be required for the tests.

Within each .py test file, you will create a TestCase class for each function being tested (one for each of the numeric bullets).

Each TestCase class will include a test function for each separate test being run.

If testing a class, a TestCase should include a setUp method that instantiates an object of the class being tested, which can then be used in each test. The setUp should only instantiate objects that will be used in every test within the TestCase.

For example, you should create one test_customer.py file. Inside this file, you would create 3 TestCase classes, one for each Customer function being tested. Each TestCase would get a setUp method that instantiates a customer object. The TestCase class for get_wallet_coin would also contain 5 other methods, one for each specific test.

Customer class:

1. **get_wallet_coin** – 5 tests
    a. Test each type of coin can be returned from wallet
    b. Test that passing in a string that is not a valid coin name will return None
2. **add_coins_to_wallet** – 2 tests
    a. Pass in a list of 3 coins, test that the len of the customer's wallet's money list went up by 3
    b. Pass in an empty list, test that the len of money list remained the same
3. **add_can_to_backpack** – 3 tests
    a. Pass in a Cola object, test that the len of the customer's backpack's purhcased_cans list went up by 1

Wallet class:

1. **fill_wallet**
    a. Instantiate a Wallet object, test that its money list has a len of 88

Soda Machine class:

1. **fill_register** – 1 test
    a. Instantiate a SodaMachine object, test that its register list has a len of 88
2. **fill_inventory**- 1 test
    a. Instantiate a SodaMachine object, test that its inventory list has a len of 30
3. **get_coin_from_register**- 5 tests
    a. Test each type of coin can be returned from register
    b. Test that passing in a string that is not a valid coin name will return None
4. **register_has_coin** - 5 tests
    a. Test that each type of coin will return True
    b. Test that a non-valid coin name will return False

5. **determine_change_value** – 3 tests
    a. Test with total payment higher
    b. Test with select_soda_price higher
    c. Test with two equal values
6. **calculate_coin_value** - 2 tests
    a. Instantiate each of the 4 coin types and append them to a list. Pass the list into this function, ensure the returned values is .41
    b. Pass in an empty list. Ensure the returned value is 0.
7. **get_inventory_soda** - 4 tests
    a. Pass in each of the 3 soda names, ensure the returned can has the same name
    b. Pass in "Mountain Dew" and ensure None is returned
8. **return_inventory** - 1 test
    a. Instantiate a can and pass it into the method. Test that the len of self.inventory is now 31
9. **deposit_coins_into_register** - 1 test
    a. Instantiate each of the 4 coins and append them to a list. Pass the list into the function, ensure the len of self.register is 92

user_interface module:

(Note: Since user_interface is a module, not a class, these TestCases will not require a setUp method)

1. **validate_main_menu** – 5 tests
    a. Pass in each number 1-4, ensure the tuple of (True, number) is returned
    b. Pass in a different number, ensure (False, None) is returned
2. **try_parse_int** – 2 tests
    a. Pass in "10", ensure the int value 10 is returned
    b. Pass in "hello", ensure 0 is returned
3. **get_unique_can_names** – 2 tests
    a. Instantiate 6 cans (two of each type) and append them to a list. Pass the list into this function, ensure the returned list only contains 3 names
    b. Pass in an empty list. Ensure an empty list is returned.
4. **display_payment_value** – 2 tests
    a. Instantiate each of the 4 coin types and append them to a list. Pass the list into this function, ensure the returned values is .41
    b. Pass in an empty list. Ensure the returned value is 0.
5. **validate_coin_selection** – 6 tests
    a. Pass in each int 1-5, ensure the appropriate tuple is returned.
    b. Pass in a different number, ensure (False, None) is returned.