Redbubble Coding Test - CLI Price Calculator

CLI Price Calculator is a Python command-line program for calculating the total price of items in a cart given the `cart` and `base-prices` information JSON files. The expected formats can be found under the `schema` directory.

The program is implemented in response to the **Redbubble Software Engineer** coding test and, accordingly, all elements of the program specification, including the provided sample files, are under the ownership of **Redbubble**.

http://take-home-test.herokuapp.com/new-product-engineer

Directory Structure

```
├── Schema
├──  cli_price_calculator_pkg  [Main Module]
    ├── __init__.py
    ├── __main__.py
    ├── cart_product.py
    ├── cart.py
    ├── cli_price_calculator.py
    ├── exceptions.py
    └── product_data.py
├── tests
    ├── fixtures
        └── [sample tests and expected *.json files]
    ├── __init__.py
    ├── test_calculator.py
    ├── test_cart.py
    └── test_product_data.py
├── .gitignore
├── README.md
├── README.pdf
└── setup.py
```

Requirement

Python version >= 3.2.5, setup.py for more information.

Developed with Python 3.8.5.

Libraries:

- unittest (built-in)
- json (built-in)
- os & os.path (built-in) - for testing
- argparse (built-in for Python >= 2.7)

Installation | Usage

Extract the zip-file to a new folder or clone this repository.

To run the module, run from the top-level directory (i.e., folder containing README.md), depending on your Python env and/or config settings:

```
python3 -m cli_price_calculator_pkg <path_to_cart_json> <path_to_base_prices_json>
```

or (if Python >= 3.2.5 is default or aliased):

```
python -m cli_price_calculator_pkg <path_to_cart_json> <path_to_base_prices_json>
```

The order of the arguments is important.

Run `python -m cli_price_calculator_pkg -h` for a more verbose description.

For all commands mentioned, keyword `python` will serve as a placeholder for `python` or `python3`. Use the Python command that you used to run the module.

For the purposes of building the package and generating distribution archives (for Package Index), a setup.py file is also included. However, it is not necessary for running the module or the tests.

Testing

Automated testing is implemented via. Python's `unittest` framework. All testing files are located under directory `tests`. Test classes `(test_cart.py, test_product_data.py, and test_calculator.py)` are designed to test functionality across the `Cart (cart.py)`, `BaseProductData (product_data.py)`, `CLIPriceCalculator (cli_price_calculator.py)` classes inside `cli_price_calculator_pkg` package directory.

Sample testing files and corresponding expected data are stored under `.\tests\fixtures` as JSON files.

To run the entire test-suite (again from the top-level folder):

```
python -m unittest discover tests -v
```

To run a single test file:

```
python -m unittest tests.<test_file_name> -v
```

Example:

```
python -m unittest tests.test_cart -v
```

Test cases implemented for each testing file are mentioned in the respective file's documentation. As mentioned in the specification, the tests do not check for schema or formatting-related errors. Each test class has 'test-runner' function definition(s) to allow easier addition of new test cases.

Automated-Testing-Workflow

To add a new test cart file to be automatically tested make sure its hyphenated suffix matches the suffix of a corresponding test base-price file, starting with 'base-prices-'. And there must be a file with the same name but with "-expected" appended containing the correct attributes for that cart file (specifically, at the moment, "count", "cart_str", and "total_price").

Example:

```
cart-4560-normal.json - [cart file]
cart-4560-normal-expected.json - [expected properties of cart file]
base-prices-normal.json - [the corresponding base-price file]
```

Add the new cart, expected, and base-prices files in `.\tests\fixtures` and insert name of the new cart file in `.\tests\fixtures\NORMAL_FILES.json`. The newly added files will automatically get tested for cart counts & content, and total-calculated-cart-price under the current workflow for 'normal' cart files, whenever tests are run.

[Again sample test files already present in `.\tests\fixtures`]

Key Algorithms

Price-tree generation algorithm in `product_data.py`

The algorithm generates a nested-dict structure for the supplied base-price JSON file. At the top level, the keys refer to the base product-types, and at each level below, the keys consist of the subsequent option-values based on sorted order of option-types until the lowest-level where the base-prices reside. For the price-tree below, the order for `hoodie` is `colour -> size -> base-prices`.

Therefore, for a small, white hoodie, the base-price will be located at `price_tree['hoodie']['white']['small']`. **Essentially, once the tree is generated, the time to retrieve the base-price of a cart-item will not be strictly dependent on the number of base-prices.**

The generated price-tree for `base-prices-normal.json`:

```
{
"hoodie": {
    "white": {
        "small": 3800,
        "medium": 3800,
```

```
            "large": 3848,
            "xl": 4108,
            "2xl": 4108,
            "3xl": 4108
        },
        "dark": {
            "small": 3800,
            "medium": 3800,
            "large": 4212,
            "xl": 4368,
            "2xl": 4368,
            "3xl": 4368
        }
    },
    "sticker": {
        "small": 221,
        "medium": 583,
        "large": 1000,
        "xl": 1417
    },
    "leggings": 5000
}
```