Package Challenge

Author: Asim Krticic

The purpose of this challenge was to create a program that will take a file which contains several lines that are representing packages. The idea was to find the best combination of items in the package so that the total weight is less than or equal to the package limit and the total cost is as large as possible.

To implement the program, I chose Python v2.7.10.

Usage

To run the program in the project root please execute the following command:

```
$   python run.py [FILE_PATH]
```

User can pass an absolute or relative path. If the path is invalid, an exception will be thrown and a proper message will be displayed in the console. Sample usage:

```
$   python run.py input_file/test_inputs.txt
```

To run the unit tests please execute the following command:

```
$   python -m unittest test.test_packer
```

Approach

The project is structured following Python's best practice. Folder structure:

```
├── package-challenge
│   ├── package
│   │   ├── __init__.py
│   │   ├── packer.py
│   │   ├── package.py
│   │   ├── parser.py
│   │   ├── exception.py
│   ├── test
│   ├── ├── __init.py__
│   ├── ├── test_packer.py
│   ├── docs
│   │   ├── **doc files**
│   ├── input_file
│   │   ├── test_inputs.txt
├── run.py
├── README.md
├── requirements.txt
├── setup.py
```

The `__init__.py` files are required to make Python treat the directories as containing packages.

The class Item contains properties index, weight and cost, and also setter methods for those properties. The class Package inherits the properties of class Item, and also contains property items and method add_item that handles adding of items to the item's list. The class Exception contains custom APIException which is derived from Exception class.

The file that was passed from the command line will be first parsed using `parse_file_content()` static method from the class Parser. The constraints listed in the challenge are handled in this method too. The data in the file is unstructured and the result of parsing will be a list of packages. This list is then passed to the `pack_items()` method where each package is sent to the `get_item_combinations()` method, which is a recursive function that calculates all combinations of items in a package that satisfy the conditions defined in the challenge and stored them to the global_list. After retrieving the combinations of items for a current package in the loop, the global_list is sorted by total_price in descending order and by total_weight in ascending order. This ensures that the best combination will always be the first in the list. The best combinations are then saved to the result_string in the following manner: items' index numbers are separated by comma and combinations of each package are delimited by newline.

For storing items' indexes combinations, total_weight and total_cost, dictionary was used as it provides efficient key/value structure. List data structure was used to store packages from the file and to recursively collect the item indexes. Recursive algorithm was chosen to easily combine all items that satisfy all conditions defined in the challenge.

The test directory contains unit tests. Tests are created in Python's unittest framework. The TestPackerFunctions class contains basic test cases that check for constraints and sanity test for finding the best combination of items in the package. To avoid dependency on test data stored in the files but still to test the method by passing the file, a temporary file is created in setup method before each test case and deleted in teardown after each test case.

Code was formatted using Python PEP8 coding style guide. Docstrings in methods and classes are ready for generating documentation using Sphinx.