PYTHON TEST

1 INTRODUCTION

Your task is to implement utilities that are going to be part of a bookshop's inventory system. As these utilities will be integrated into a larger system, it is important that you follow the instructions closely, including the specified interfaces, and do the work to an industry standard level. Try the write your solution as if you were working on a project for an actual client.

Test Cases

- Implement test cases for your functions.
- You are free to use third-party packages to support your tests, but make sure that you document the packages that you have used, including their versions.

Your submission

- Submit the deliverables as electronic source code files in a zip-archive. Please do not include binary files. You can submit your solution by replying to the email in which you received the test instructions.
- Please submit all files that you think are necessary to do your solution justice.
- Ensure that you include the names and versions of any third-party frameworks you've used to support your tests.

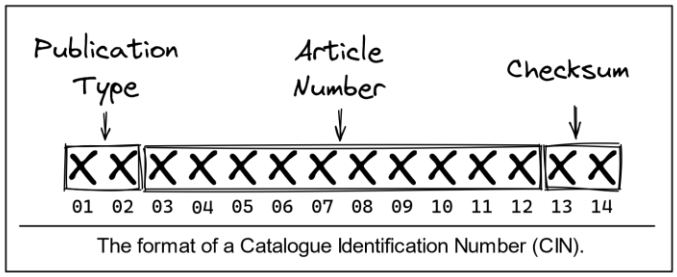
Ahead of change

Version 2021-11-08

2 EXERCISES

2.1 Catalogue Identification Number (CIN)

Each item in the bookshop's catalogue can be identified by its unique Catalogue Identification Number (CIN). A CIN is a sequence of 14 digits that follows a fixed structure: The first two digits identify the type of publication (e.g., "17" for book or "42" for a magazine), the next 10 digits form an article number, and the last 2 digits form a checksum that can be used to validate the CIN. Leading zeros are included as part of the CIN.



The format of a Catalogue Identification Number (CIN).

The checksum is computed by taking the first 12 digits of CIN, multiplying each individual digit by its position (note: the position number starts at one), calculating the sum of all those multiplications, and, finally, by computing the modulo 97 of that sum. For instance, for the CIN `17000372214424`, you can validate its checksum, 24, as follows:

```
(1*1 + 2*7 + 3*0 + 4*0 + 5*0 + 6*3 + 7*7 + 8*2 + 9*2 + 10*1 + 11*4 + 12*4) % 97 = 24
```

Exercise 1.

Write a function, `is_valid_cin`, that validates a CIN. If a CIN follows the correct format and its checksum checks out, the function should return `True`. In all other cases, the function should return `False`. The function should have the following signature:

```
def is_valid_cin(cin: str) -> bool:
    ...
```

2.2 Taking Stock

The bookshop wants to update their inventory at the end of the day based on the incoming and outgoing transactions. The transactions are recorded in a transaction log, which is used as input for the function that updates the inventory.

In this log, each line represents the transaction of a single item with the item's CIN, whether it's `INCOMING` or `OUTGOING`, and the quantity of the transaction. Here's a sample of such a transaction log:

```
17000372214424 INCOMING 9
17000372214424 OUTGOING 1
17000372214424 INCOMING 3
42100551007977 OUTGOING 3
42100551007977 INCOMING 1
17000372214424 OUTGOING 4
```

The six lines in this sample indicate that the inventory of item `17000372214424` has gone up by 7 copies (+9, -1, +3, -4), while the inventory of item `42100551007977` has gone down by 2 copies (+1, -3).

Exercise 2.

Write a function, **`calculate_inventory`**, that calculates the new inventory at the end of the day based on the day's transaction log. The function takes two arguments:

- **`start_inventory:`** a mapping of CINs to their quantity at the start of the day
- **`transaction_log:`** the transaction log (a single, potentially multiline, **`str`**)

The function returns a *new dictionary* that maps CIN to their updated inventory count.

Additional requirements

- The **`start_inventory`** mapping should not be changed.
- Items that have their inventory count drop to 0 are still *included* in the returned inventory.
- Assume that CINs not present in the **`start_inventory`** have a starting count of 0.
- It is okay for an item's inventory count to (temporarily) drop below zero while processing a transaction log, as employees may enter transactions out of order. However, if there are still items with a negative inventory count after processing the entire transaction log, raise an appropriate exception with an informative message, as having a negative quantity in stock at the end of the day should not be possible.

Function signature

```python
def calculate_inventory(
    start_inventory: collections.abc.Mapping[str, int],
    transaction_log: str,
) -> dict[str, int]:
    ...
```

2.3 Daily Best Sellers

Another feature requested by the bookshop is a function that extracts a daily best sellers list from the daily transaction log. They want to be able to specify the number of items that should be reflected in the list, and they also want to have the option to restrict the daily best sellers list to a specific publication type (as specified by the first two digits of the CIN; see section).

Which items are best sellers is based solely on the *outgoing* transactions in the daily transaction log.

Exercise 3.

Write a function, `calculate_best_sellers`, that returns a list of the `n` best selling items sorted by the number of copies sold (highest first). Ties are broken by using the ascending lexicographical (alphabetical) ordering of the CINs. The elements in the list should be instances of the `BestSeller` class described below.

The function takes two required arguments and one optional argument:

- **`transaction_log:`** the transaction log (a single, potentially multiline, **`str`**)
- **`n:`** the requested number of items in the best sellers list
- **`publication_type:`** an (optional) publication type to filter the items by

The function returns a list of `BestSeller` instances, sorted as described above.

Additional requirements

- The function may return a list with fewer than `n` elements if the transaction log does not contain enough items to fill the list.
- If the quantity sold of an item is 0, it is *not* included in a best sellers list even if that means returning a list with fewer than `n` elements.

The `BestSeller` class

Create a class, `BestSeller`, with three *read-only* properties:

- **`cin`** (str): the item's Catalogue Identification Number
- **`publication_type`** (str): the item's publication type (see exercise 2.1)
- **`quantity_sold`** (int): The number of copies sold of this item

You are free to implement other methods, if necessary or convenient.

Function signature

```
def calculate_best_sellers(
    transaction_log: str,
    n: int,
    publication_type: typing.Optional[str] = None,
) -> list[BestSeller]:
    ...
```