1 Information about inventory

Information about an item-code in the inventory is categorised into the following two classes

```
class BasicInfo:
    __name:  str
    __price: float
```

```
class Offer:
    __running: bool
    __n_items: np.uint64
    __coeffs: np.ndarray
    __reduction: float
```

Among the Offer class members,

- ___running is True if there is an offer for the given item-code, or False otherwise.
- ___n_items is the frequency at which the offer becomes eligible
- ___coeffs is a two element array that contains information to effectively apply the offer reduction.

  In the given objective, offer or no-offer on item-codes can be casted as

$$\text{___reduction}(p) = \text{___coeffs}[0] \times p + \text{___coeffs}[1]) \tag{1}$$

  where p is the price of an item and ___reduction is the value deducted from the item-code items total price whenever the offer becomes eligible

Code	Name	___n_items	___coeffs[0]	___coeffs[1]
A	Apple	3	1	0
B	Banana	3	3	−100
P	Pear	0	0	0

Table 1: Inventory. Note that if ___n_items for an item-code is zero in the inventory, it is interpreted as no offer.

For convenience, FullInfo class is made-up of the above two classes, BasicInfo and Offer

```
class FullInfo:
    basic_info: BasicInfo
    offer: Offer
```

An empty dictionary

```
INVENTORY_ITEMS (: Dict[str, FullInfo]) = {}
```

is instantiated to hold the inventory information, and each row, information about an item-code, in table 1 is read and added to the dictionary.

```
if (__n_items == 0):
     __running = False
else:
     __running = True
```

```
INVENTORY_ITEMS[Code] = FullInfo(
    BasicInfo(Name),
    Offer(__running, __n_items,
          __coeffs[0], __coeffs[1]
    )
```

Note that ___price(: BasicInfo) and ___reduction(: Offer) are set to 0.0 by default if values are not specified.

When price is known, at a later point, for an item-code, the corresponding item-code price and offer reductions are set using the respective class members as shown below

```
class BasicInfo:
    def set_price(self, price: float):
        self.__price = price

class Offer:
    def set_reduction(self, price: float):
        self.__reduction = self.__coeffs[0] * price + self.__coeffs[1]
```

2 Information about user collected items

So far the data-structures discussed hold the inventory information. The below Item class holds ___count (number of items user picked), and ___total (total price after applying the offer reductions) for an item-code.

```
class Item:
    __count: uint64
    __total: float
```

An empty dictionary

```
items (: Dict[str, Item]) = {}
```

is instantiated to hold the information about user picked items based on item-code.

Every time user scans an item-code, items[item-code] invokes algorithm 2 to perform necessary updates. This way, **items** dictionary holds up-to-date count and total for all items in the basket based on item-code.

Algorithm 1: Check if eligible for offer at present

Data:
user_n_items(: uint64): Number of items that user has picked so far
offer_n_items(: uint64): Frequency at which offer becomes eligible
Result: offer_eligible(: bool)
if *count is zero* **then**
    set offer_eligible to False
else
    set offer_eligible to True, if count is an exact multiple of offer_n_items (user_n_items % offer_n_items == 0)
end

Algorithm 2: Scan an item and update necessary details

Data: items: Item, item_full_info: FullInfo
Result: Updated items (count and total)
Increase the scanned item count by one
Increase the scanned item total by it's price
Set offer_eligible as the algorithm 1 return value
if *offer_eligible* **then**
    Reduce the item total by offer reduction
end

The total price is obtained by looping over all item-codes in the items dictionary and accumulating each item-code total to the total.

$$\text{total} = \sum_{\forall i} \text{items}[i].\text{get\_total}() \tag{2}$$