Supply management

A well-automatized stock control is an important value in an enterprise. Maintaining plenty of stock results in the very important continuity in production and/or sales. On the other hand, one must see to it that the stock absorbs as little of the enterprise's budget as possible.

For an efficient stock control, a correct product list and a correct registration of the articles sold is necessary. The data that must be kept up to date depends for every article on various factors. Our enterprise buys articles, and sells them for a certain profit. When the supply of an article is below the minimum boundary, its stocks must be replenished. At any given moment, an overview of stocks that need replenishing can be asked.

Assignment

In order to solve the problem above, you must first define a class Article that supports the following methods:

1. An initializing method __init__ to which five arguments must be given: a code that consists of one letter (A,B or C - defending on the sales frequency) followed by a number, the complete name of the article, a purchaseprice, the stock and a minimum_number_in_stock and a maximum_number_in_stock. The constructor must respectively appoint these arguments to the attributes code, name, purchaseprice, stock and minimumstock and maximumstock. If no maximum stock was given, this is equated with the minimum stock. If a minimum stock is given that is larger than the maximum stock, an error message appears 'minimum stock may not be larger than maximum stock' (see example).
2. A method __str__ with no arguments that prints a string representation of the article. Look at the example below to determine what the string representation should look like.
3. A method __repr__ with no arguments that prints a string representation of the article. The method __repr__ prints a syntactically correct Python expression, that — if it were to be evaluated — makes an object that is equal to the object that was originally given to __repr__.
4. A method value, that prints the total purchase value of the stock for this article.
5. A method shortage, that determines whether or not the stock has decrease below the minimum value. If so, the number below the minimum is printed (as a negative number), otherwise, the value 0 (zero) is printed.
6. A method reorder that indicates how many items must be ordered, should a shortage arise. If no items must be ordered, a 0 (zero) is printed.

Define a second class Stock that provides the following methods:

1. An initializing method __init__ to which a dictionary of articles is given. This dictionary contains the article code as a key and the article as a corresponding value. Standard, the empty dictionary is appointed to the attribute articles.
2. A method addArticle that takes an object of the class Article as a parameter and that adds this element to a product list.
3. A method value that prints the purchase value of the entire stock at that moment.
4. A method purchase, to which two arguments can be given: the code of the article involved and the number of articles purchased. The methods sees to it that, for the article involved from the list, the stock is adjusted with number pieces. If the given number is negative, the error

message 'number must be positive' should appear. Furthermore, if a non-existing code is given, an error message should appear ('article does not exist').

5. A method sales, to which two arguments should be given as well: the code of the article and the number of articles sold. If the given number is negative, or if the given code does not exist, a similar error message should appear as in purchase. The stock of the article involved must be adjusted if there are enough pieces to meet an order, if not, you will also receive an error message. Look at the example below.
6. A method replenish that sees to it that a list that was sorted on code is printed for the articles that need replenishing. The number that the article is short is always stated. Look at the example to see how this list must be printed.

Example

```
>>> article1 = Article('A207', 'rug', 45, 25, 20, 30)
>>> print(article1)
article: rug
code: A207
purchaseprice: 45
stock: 25
minimumstock: 20
maximumstock: 30
>>> article1
Article('A207', 'rug', 45, 25, 20, 30)
>>> article1.value()
1125
>>> article1.shortage()
0

>>> article2 = Article('C130', 'closet', 230, 10, 3)
>>> print(article2)
article: closet
code: C130
purchaseprice: 230
stock: 10
minimumstock: 3
maximumstock: 3
>>> article2
Article('C130', 'closet', 230, 10, 3, 3)
>>> article2.value()
2300
>>> article2.shortage()
0

>>> stock = Stock()
>>> stock.addArticle(Article('A207', 'rug', 45, 25, 20, 30))
>>> stock.addArticle(Article('B734', 'sofa', 186, 12, 8, 16))
>>> stock.addArticle(Article('C130', 'closet', 230, 10, 3))
>>> stock.sales('A207', 7)
>>> stock.sales('B734', 14)
Traceback (most recent call last):
AssertionError: not enough supply of article B734 (2 pieces short)
>>> stock.sales('C130', 8)
>>> stock.value()
3502
>>> stock.replenish()
article: rug
code: A207
```

```
purchaseprice: 45
stock: 18
minimumstock: 20
maximumstock: 30
--> 12 reorders
==============================
article: closet
code: C130
purchaseprice: 230
stock: 2
minimumstock: 3
maximumstock: 3
--> 1 reorders
```

Een goed geautomatiseerd stockbeheer is een belangrijke pijler voor een onderneming. Het aanhouden van voldoende voorraad moet zorgen voor de zeer belangrijke continuïteit in de productie en/of verkoop. Anderzijds moet er op gelet worden dat de voorraad zo weinig mogelijk van het budget van een onderneming opslorpt.

Voor een efficiënt voorraadbeheer is een correcte artikellijst en een correcte registratie van de verkochte artikels noodzakelijk. Welke gegevens moeten bijgehouden worden voor elk artikel hangt af van verschillende factoren. Ons bedrijf koopt artikels aan, en verkoopt deze verder door met een bepaalde winst. Wanneer de voorraad van een artikel onder een minimumgrens komt, moet dit artikel aangevuld worden. Er kan op elk ogenblik een overzicht van de aan te vullen artikels opgevraagd worden.

Opgave

Definieer, om bovenstaand probleem op te lossen, vooreerst een klasse Artikel die ondersteuning biedt voor volgende methoden:

1. Een initialisatiemethode __init__ waaraan vijf argumenten moeten doorgegeven worden: een code die bestaat uit één letter (A,B of C - naargelang de verkoopfrequentie) gevolgd door een volgnummer, de volledige naam van het artikel, een aankoopprijs, de voorraad en een minimum_aantal_in_voorraad en een maximum_aantal_in_voorraad. De constructor moet deze argumenten respectievelijk toekennen aan de attributen code, naam, aankoopprijs, voorraad en minimumvoorraad en maximumvoorraad. Als de maximumvoorraad niet opgegeven wordt, wordt deze gelijk gesteld aan de minimumvoorraad. Wordt echter een minimumvoorraad opgegeven die groter is dan de maximumvoorraad, dan verschijnt de foutboodschap 'minumumvoorraad mag niet groter zijn dan de maximumvoorraad' (zie voorbeeld).
2. Een methode __str__ zonder argumenten die een stringvoorstelling van het artikel teruggeeft. Bekijk onderstaand voorbeeld om te achterhalen hoe deze stringvoorstelling er moet uitzien.
3. Een methode __repr__ zonder argumenten die een stringvoorstelling van het artikel teruggeeft. De methode __repr__ geeft een syntactisch correcte Python expressie terug, die — wanneer deze geëvalueerd zou worden — een object aanmaakt dat gelijk is aan het object dat origineel werd doorgegeven aan __repr__.
4. Een methode waarde, die voor dit artikel de totale aankoopwaarde van de voorraad berekent en teruggeeft.
5. Een methode tekort, die bepaalt of de voorraad onder de minimumwaarde gedaald is. Indien dit het geval is, wordt het aantal onder het minimum teruggegeven (als een negatief getal),

zoniet wordt als waarde 0 (nul) teruggegeven.
6. Een methode bijbestellen die aangeeft hoeveel stuks eventueel moeten besteld worden, wanneer er een tekort is ontstaan. Moet er van dit artikel niets bijbesteld worden, dan wordt een 0 (nul) teruggegeven.

Definieer een tweede klasse Voorraad die voorziet in volgende methoden:

1. Een initialisatiemethode __init__ waaraan een dictionary van artikels wordt doorgegeven. Deze dictionary bevat als sleutel de artikelcode en als corresponderende waarde het artikel. Standaard wordt een lege dictionary toegekend aan het attribuut artikels.
2. Een methode artikelToevoegen die als parameter een object van de klasse Artikel neemt en dit element toevoegt aan de artikellijst.
3. Een methode waarde die van de volledige voorraad op dat ogenblik de totale aankoopwaarde teruggeeft.
4. Een methode aankoop, waaraan twee argumenten moeten doorgegeven worden: de code van het betreffende artikel en het aantal aangekochte artikels. Deze methode zorgt ervoor dat, voor het betreffende artikel uit de lijst, de voorraad wordt aangepast met aantal stuks. Als het opgegeven aantal negatief is moet als foutboodschap verschijnen 'aantal moet positief zijn'. Ook als een niet-bestaande artikelcode wordt opgegeven moet je een foutboodschap krijgen ('artikel bestaat niet').
5. Een methode verkoop, waaraan eveneens twee argumenten worden doorgegeven: de code van het artikel en het aantal verkochte items. Als het opgegeven aantal negatief is of de opgegeven code niet bestaat, moet eenzelfde foutboodschap als bij aankoop worden weergegeven. De voorraad van het betreffende artikel wordt aangepast indien er voldoende stuks in voorraad zijn om aan de bestelling te voldoen, zoniet krijg je eveneens een foutboodschap. Bekijk hiervoor het onderstaand voorbeeld.
6. Een methode aanvullen die ervoor zorgt dat een op code gesorteerde lijst wordt afgedrukt van die artikels waarvoor een tekort is. Er wordt telkens ook vermeld hoeveel artikels er moeten worden bijbesteld. Bekijk het voorbeeld om te zien hoe de lijst moet afgedrukt worden.

Voorbeeld

```
>>> artikel1 = Artikel('A207', 'tapijt', 45, 25, 20, 30)
>>> print(artikel1)
artikel: tapijt
code: A207
aankoopprijs: 45
voorraad: 25
minimumvoorraad: 20
maximumvoorraad: 30
>>> artikel1
Artikel('A207', 'tapijt', 45, 25, 20, 30)
>>> artikel1.waarde()
1125
>>> artikel1.tekort()
0

>>> artikel2 = Artikel('C130', 'kast', 230, 10, 3)
>>> print(artikel2)
artikel: kast
code: C130
aankoopprijs: 230
```

```
voorraad: 10
minimumvoorraad: 3
maximumvoorraad: 3
>>> artikel2
Artikel('C130', 'kast', 230, 10, 3, 3)
>>> artikel2.waarde()
2300
>>> artikel2.tekort()
0

>>> voorraad = Voorraad()
>>> voorraad.artikelToevoegen(Artikel('A207', 'tapijt', 45, 25, 20, 30))
>>> voorraad.artikelToevoegen(Artikel('B734', 'zetel', 186, 12, 8, 16))
>>> voorraad.artikelToevoegen(Artikel('C130', 'kast', 230, 10, 3))
>>> voorraad.verkoop('A207', 7)
>>> voorraad.verkoop('B734', 14)
Traceback (most recent call last):
AssertionError: niet voldoende voorraad van artikel B734 (2 stuks te kort)
>>> voorraad.verkoop('C130', 8)
>>> voorraad.waarde()
3502
>>> voorraad.aanvullen()
artikel: tapijt
code: A207
aankoopprijs: 45
voorraad: 18
minimumvoorraad: 20
maximumvoorraad: 30
--> 12 stuks bijbestellen
============================
artikel: kast
code: C130
aankoopprijs: 230
voorraad: 2
minimumvoorraad: 3
maximumvoorraad: 3
--> 1 stuks bijbestellen
```