Nebenläufigkeiten - Lagerhaltung

Folgende Kenntnisse werden vorausgesetzt:
- Threads in Python
- Synchronisationstechniken für Threads

In dieser Aufgabe verwenden wir Threads, um das Füllen und Leeren eines Lagers zu programmieren. Ein Lager (realisiert als globale Liste) enthält Produkte (realisiert als ganzzahlige Zufallszahlen), die von mehreren Verbrauchern immer wieder entnommen werden. Gleichzeitig wird geprüft, ob noch Produkte im Lager sind. Falls das Lager leer ist, werden neue Produkte (d.h. Zufallszahlen) in das Lager (Liste) gelegt.

a) Definieren Sie die globale Variable *lager* als leere Liste im Programm
Hinweis: global lager

***Anmerkung**: Hier wäre eine objektorientierte Lösung wesentlich eleganter, aber aufgrund der Kürze des Programms verzichten wir an dieser Stelle darauf und verwenden globale Variablen. Dies sollten Sie jedoch nie in größeren, komplexen Programmen machen! Siehe auch unten „Kleiner Exkurs zum Thema globaler und lokaler Variablen „scope“*

b) Fügen Sie die Methode *pruefeLager()* hinzu. Diese wird später durch einen separaten Thread ausgeführt. Innerhalb der Methode prüfen Sie in einer while-Schleife, ob das Lager leer ist. Falls ja, fügen Sie 10 zufällige Elemente hinzu. Anschließend geben Sie aus, dass das Lager gefüllt wurde. Geben Sie zudem die Liste aus.
Danach schläft der Thread für 100 ms (Methode *sleep(..)*)
Fügen Sie nun noch eine Variable hinzu, die bei jedem befüllen der Liste um den Wert 1 hochgezählt wird. Beenden Sie die while-Schleife, wenn diese Variable den Wert 20 übersteigt.

Hinweis:
- Random importieren
- Erzeugen und Anfügen:

```
e = random.randint(0,100)
lager.append(e)
```

c) Nun fügen Sie die Methode *verbraucher(verbraucherNummer)* hinzu. Diese prüft ebenfalls in einer Endlosschleife, ob das Lager leer ist. Falls nicht, suchen Sie das Maximum aus dem Lager. Dieses Element geben Sie auf der Konsole aus, zusammen mit der Nummer des Verbrauchers (verbraucherNummer). Anschließend wird das Element aus der Liste gelöscht (! Achten Sie bitte darauf die Befehle in dieser Reihenfolge zu implementieren).

Hinweis: Löschen von Elementen aus der Liste: `lager.remove(e)`

d) Ergänzen Sie nun in Ihrem Hauptprogramm: starten Sie einen Lager-Prüfe-Thread („Erzeuger“) mit der Thread-Methode *pruefeLager()* und drei Verbraucher-Threads, Methode *verbraucher(verbraucherNummer)*

Hinweis: Beim Starten der Verbraucher müssen Sie der Thread-Methode *verbraucher*(…) die Nummer (1, 2 bzw. 3) als Argument mitgeben.

e) Starten Sie das Programm und beobachten Sie die Ausgabe eine Weile.
→ Was passiert? Wie kann die aufgetretene Problematik beschrieben werden? Versuchen Sie anhand der Prints / Fehlermeldungen nachzuvollziehen was passiert ist.

f) Beheben Sie das Problem mittels Synchronisation. Testen Sie anschließend, ob Ihre Maßnahme funktioniert.

Kleiner Exkurs zum Thema globaler und lokaler Variablen „scope“

In der Aufgabe definieren wir Variablen als „global“. Was genau bedeutet das eigentlich? In Python gibt es verschiede „scopes“ in denen eine Variable verfügbar ist. Wird eine Variable beispielsweise in einer Methode definiert, so ist sie auch nur in dieser Methode verfügbar. Sie befindet sich dann im „local scope“ dieser Methode. In unserer Laboraufgabe definieren wir Variablen in einem Skript außerhalb der Methoden oder einer Klasse. Diese Variablen gehören dann automatisch zum sogenannten „global scope“. Auf diese Variablen kann jetzt auch aus einem „local scope“ zugegriffen werden. Für den Zugriff auf den Wert dieser Variablen müssen wir nichts weiter tun. Will man innerhalb einer Methode einer globalen Variable einen Wert zuweisen, muss diese zuvor als „global“ innerhalb der Methode deklariert werden.

Aber weisen wir nicht auch der Liste „lager“ einen neuen Wert zu, ohne diese davor innerhalb der Methode als „global“ zu deklarieren?
Nein, denn eine Liste ist ein veränderbares Objekt. Das heißt wir verändern zwar den Inhalt des Objektes (mit .append() und .remove()) allerdings weisen wir der Variablen an sich keinen neuen Wert zu, dieser bleibt ein Listen-Objekt.

Weitere Informationen zu diesem Thema finden Sie hier:
https://realpython.com/python-scope-legb-rule/#:~:text=In%20Python%2C%20the%20concept%20of,dictionaries%20are%20commonly%20called%20namespaces.