Nom	
Prénom	
Groupe	

Note	

Algorithmique Python

Sup S2 Epita

Examen B2

7 janvier 2025

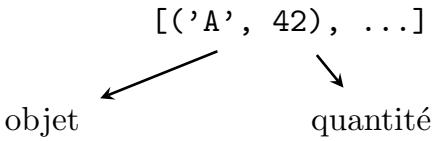
Consignes (à lire) :

- ☐ Vous devez répondre directement **sur ce sujet**.
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées**.
  - Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Code :**
  - Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - **Tout code `Python` non indenté ne sera pas corrigé.**
  - Les seules classes, fonctions, méthodes que vous pouvez utiliser sont données en **annexe**.
  - Vos fonctions doivent impérativement respecter les exemples d'applications donnés.
  - Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
    Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.
  - Comme d'habitude l'optimisation est notée. Si vous écrivez des fonctions non optimisées, vous serez notés sur moins de points. [1]
- ☐ Durée : 1h30

1. Des fois, il vaut mieux moins de points que pas de points.

Introduction

Le but des exercices suivants est de gérer le stock d'un entrepôt représenté par une liste de couples. Le premier élément de chaque couple correspond à l'objet stocké et le second élément indique la quantité de cet objet dans le stock.



Les objets stockés sont supposés être ordonnables, quel que soit leur type, et la liste représentant le stock est triée par ordre d'objets croissants.

```
>>> 'A' < 'B'
True
```

Il n'y a pas deux fois le même objet dans la liste représentant le stock. La quantité d'un objet dans le stock est toujours supérieure à 0.

Tous les listes représentant des stocks sont supposées valides.

Ci-dessous des exemples de listes représentant des stock valides ou non.

```
>>> stock1 = [('B', 4), ('M', 5), ('O', 1)]
>>> not_a_stock1 = [('M', 5), ('O', 1), ('B', 4)]
>>> stock2 = [('Y', 8)]
>>> not_a_stock2 = [('Y', 8), ('Y', 8)]
>>> not_a_stock3 = [('L', 8), ('Y', -1)]
```

— `not_a_stock1` n'est pas un stock valide car la liste n'est pas triée par objets croissants.
— `not_a_stock2` n'est pas un stock valide car l'objet `'obj42'` est présent plus d'une fois dans la liste.
— `not_a_stock3` n'est pas un stock valide car l'objet `'obj42'` a une quantité $<= 0$.

Exercice 1 (Vérification– *4 points*)

Écrire la fonction `check_stock(`*stock*`,` *capacity*`)` qui prend en paramètres :

- `stock`, une liste représentant un stock valide
- `capacity`, un entier supposé ≥ 0

et vérifie si le stock représenté par la liste `stock` contient au plus `capacity` objets (nombre d'objets du stock $\leq$ *capacity*).

Exemples d'applications :

```
>>> check_stock([], 0)
True
>>> check_stock([('B', 4), ('M', 5), ('O', 1)], 10)
True
>>> check_stock([('B', 4), ('M', 5), ('O', 1)], 9)
False
```

Exercice 2 (Construction– *6 points*)

Écrire la fonction `build_stock(`L`)` qui prend en paramètres :

— `L`, une liste d'objets triée en ordre croissant

et construit et retourne un stock valide (voir introduction).

Exemples d'applications :

```
>>> build_stock([])
[]
>>> build_stock(['A', 'A', 'A', 'D', 'O', 'O'])
[('A', 3), ('D', 1), ('O', 2)]
>>> build_stock(['C', 'C', 'C', 'C'])
[('C', 4)]
```

Exercice 3 (Get– *10 points*)

1. Écrire la fonction `get_stock(`*stock*`,` *obj*`,` *request*`)` qui prend en paramètres :
   — `stock`, une liste représentant un stock valide
   — `obj`, un objet
   — `request`, un entier supposé > 0
   et retire `request` unités de l'objet `obj` du stock `stock` si cet objet est présent.
   Si le nombre d'unités disponibles de l'objet `obj` dans le stock `stock` est $\leq$ *request*, l'objet `obj` est supprimé du stock. Sinon, le nombre d'unités disponibles de l'objet `obj` dans le stock `stock` est simplement mis à jour.
   La fonction retourne `True` si `obj` est toujours dans le stock `stock` après modification, `False` sinon.

   Exemples d'applications :

```
>>> get_stock([], 'Y', 24)
False

>>> stock = [('B', 4), ('M', 5), ('O', 1), ('Z', 10)]

>>> get_stock(stock, 'Y', 24)
False
>>> stock
[('B', 4), ('M', 5), ('O', 1), ('Z', 10)]

>>> get_stock(stock, 'M', 3)
True
>>> stock
[('B', 4), ('M', 2), ('O', 1), ('Z', 10)]

>>> get_stock(stock, 'O', 5)
False
>>> stock
[('B', 4), ('M', 2), ('Z', 10)]

>>> get_stock(stock, 'B', 4)
False
>>> stock
[('M', 2), ('Z', 10)]
```

2. Écrire la fonction `shopping(`*stock*`,` *requests*`)` qui prend en paramètres :
   — `stock`, une liste représentant un stock valide
   — `requests`, une liste de couples `(obj, request)` où `obj` est un objet et `request` est un entier supposé > 0

   et pour chaque couple `(obj, request)` de la liste `requests`, retire `request` unités de l'objet `obj` du stock `stock` tant que c'est possible.

   La fonction retourne `False` dès qu'un objet n'était pas présent ou a été supprimé du stock suite à une requête, `True` sinon.

   La fonction précédente `get_stock(`*stock*`,` *obj*`,` *request*`)` doit être utilisée, qu'elle soit écrite ou non.

   Exemples d'applications :

```
>>> stock = [('B', 4), ('M', 5), ('O', 1)]

>>> shopping(stock, [])
True
>>> stock
[('B', 4), ('M', 5), ('O', 1)]

>>> shopping(stock, [('M', 1)])
True
>>> stock
[('B', 4), ('M', 4), ('O', 1)]

>>> shopping(stock, [('B', 1), ('M', 1), ('M', 1)])
True
>>> stock
[('B', 3), ('M', 2), ('O', 1)]

>>> shopping(stock, [('M', 4), ('B', 1), ('O', 2)])
False
>>> stock
[('B', 3), ('O', 1)]

>>> shopping(stock, [('Z', 4)])
False
>>> stock
[('B', 3), ('O', 1)]
```

Annexes

Fonctions et méthodes autorisées

Vous pouvez utiliser les méthodes `append` et `pop` (sans argument), la fonction `len` sur les listes ainsi que la fonction `range` :

```
>>> L = []

>>> for i in range(5):
        L.append(i)

>>> L
[0, 1, 2, 3, 4]

>>> len(L)
5

>>> for i in range(5, 10):
    L.append(i)
>>> L
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> L.pop()
9
>>> L
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

Aucun opérateur n'est autorisé sur les listes (`+`, `*`, `==` ...).