Défis logiques 2

# But du laboratoire

* Utiliser les collections
* Utiliser les conditions
* Utiliser les boucles
* Utiliser TOUT!

# Temps d’attente en file[[1]](#footnote-1)

Nicolas est un bon ami. Il va aller faire la file pour acheter les nouveaux souliers que vous désirez tant. Sa seule condition est que vous lui disiez exactement combien de temps il va devoir attendre.

Chaque personne peut acheter seulement une paire à la fois et la transaction dure exactement une minute. Ensuite, si elle en veut plus, elle retourne au bout de la ligne. Si elle a tout ce qu’elle désire, elle quitte la file tout simplement.

**Important :**

Votre application va toujours recevoir une liste qui contient la file d’attente complète avec le nombre de personnes et le nombre de paires de souliers qu’ils veulent acheter et la position de votre ami.

**Exemple :**

Imaginez que vous recevez le tuple initiale ([2, 5, 3, 4, 6], 2). La liste représente 5 personnes qui veulent acheter des souliers, la quantité qu’elles vont acheter et leur position dans la file. Le nombre entier seul, lui, désigne la position de départ de votre ami, il est le 3e dans la file (index 2). la première personne achète une paire et s’en retourne au bout de la file pour donner ce résultat : [5, 3, 4, 6, 1]. Le tout se poursuit avec [3, 4, 5, 1, 4] et ainsi de suite jusqu’à ce que notre ami est acheté 3 paires de souliers et attendu exactement 12 minutes.

Construisez cette application en sachant que la liste sera toujours remplie d’entiers positifs et que la position de votre ami sera valide. Soyez prêt, cependant, à certaines très longues files d’attente.

Files à calculer :

ls\_files\_attente = [

([2, 5, 3, 6, 4], 0),

([2, 5, 3, 6, 4], 1),

([2, 5, 3, 6, 4], 2),

([2, 5, 3, 6, 4], 3),

([2, 5, 3, 6, 4], 4)

]

Bonnes réponses :

ls\_reponses\_prof = [6, 18, 12, 20, 17]

# Cryptomonnaie[[2]](#footnote-2)

Nicolas gère des serveurs qui minent de la cryptomonnaie. Il a un nombre variable de calculs à effectuer et de serveurs pour les faire. L’information sur son travail ainsi que ses ressources lui est transmis sous forme de tuple (x, y) où x représente le nombre de **serveurs** qu’il a de disponible et y le nombre de **calculs** à faire.

**Important :**

Nicolas veut toujours répartir le travail le plus équitablement possible sur ses serveurs pour de meilleures performances. Si les calculs ne peuvent pas être divisés également ce sont les premiers serveurs qui prennent un calcul supplémentaire.

Ce qu’il faut obtenir comme résultat est une liste de liste qui affiche quels calculs sont effectués sur quels serveurs.

**Exemples :**

Nicolas a 2 serveurs et 4 calculs à faire.

Il reçoit (2, 4) et la réponse doit être [[0, 1], [2, 3]]

Une autre journée, Nicolas a 4 serveurs et 3 calculs à faire.

Il reçoit (4, 3) et la réponse doit être [[0], [1], [2], []]

Quelques jours plus tard, un autre scénario de 4 serveurs et 10 calculs.

Il reçoit (4, 10) et la réponse doit être [[0, 1, 2], [3, 4, 5], [6, 7], [8, 9]]

Vous devez créer une application qui va répartir la charge des serveurs selon les préférences de Nicolas. Ne vous en faites pas, les demandes sont toujours des entiers positifs.

Voici la liste des tuples qu’il aura à gérer, affichez la liste de répartition adéquate :

ls\_a\_gerer = [

(2, 4), (3, 3), (3, 9),

(2, 5), (4, 10), (4, 5),

(1, 1), (2, 1), (5, 4), (5, 1)

]

Bonne réponse:

ls\_reponse\_prof = [

[[0, 1], [2, 3]],

[[0], [1], [2]],

[[0, 1, 2], [3, 4, 5], [6, 7, 8]],

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

[[0, 1, 2], [3, 4, 5], [6, 7], [8, 9]],

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

[[0]],

[[0], []],

[[0], [1], [2], [3], []],

[[0], [], [], [], []]

]

1. Exercice trouvé sur Codewars : [Queue Time Counter](https://www.codewars.com/kata/5b538734beb8654d6b00016d) [↑](#footnote-ref-1)
2. Exercice trouvé sur Codewars : [Distribute server workload](https://www.codewars.com/kata/59f22b3cf0a83ff3e3003d8c) [↑](#footnote-ref-2)