Rappelons que, dans Python, le quotient et le reste de la division euclidienne (dans $\mathbb{Z}$) de `a` par `b` sont donnés respectivement par `a//b` et `a%b`.

Partie I. Coupe d'un ruban avec répétition

Question 1

Il y a en général plusieurs valeurs de M possibles selon le choix de découpage ; on recherche la valeur maximale. Dans chaque cas, je liste les tableaux $k = [k_0, k_1]$ possibles, où $k[i]$ est le nombre d'exemplaires du produit i que l'on choisit de découper. Je me limite évidemment aux cas où il ne reste plus assez de tissu pour découper un nouveau produit ! C'est-à-dire que pour chaque valeur possible de k_0 (de 0 à $L//a_0$), je choisis $k_1 = (L - k_0 a_0)//a_1$.

- Cas $L = 14$: pour $k = [0, 1]$, $M = 5$; pour $k = [1, 1]$, $M = 8$; pour $k = [1, 1]$, $M = 6$; maximum 8
- Cas $L = 16$: pour $k = [0, 2]$, $M = 10$; pour $k = [1, 1]$, $M = 8$; pour $k = [2, 0]$, $M = 6$; maximum 10
- Cas $L = 18$: pour $k = [0, 2]$, $M = 10$; pour $k = [1, 1]$, $M = 8$; pour $k = [3, 0]$, $M = 9$; maximum 10
- Cas $L = 24$: pour $k = [0, 3]$, $M = 15$; pour $k = [1, 2]$, $M = 13$; pour $k = [4, 0]$, $M = 12$; maximum 15

Question 2

Même principe. Les vecteurs k restent les mêmes, je rectifie les valeurs de M.

- Cas $L = 14$: pour $k = [0, 1]$, $M = 5$; pour $k = [1, 1]$, $M = 9$; pour $k = [1, 1]$, $M = 8$; maximum 9
- Cas $L = 16$: pour $k = [0, 2]$, $M = 10$; pour $k = [1, 1]$, $M = 9$; pour $k = [2, 0]$, $M = 8$; maximum 10
- Cas $L = 18$: pour $k = [0, 2]$, $M = 10$; pour $k = [1, 1]$, $M = 9$; pour $k = [3, 0]$, $M = 12$; maximum 12
- Cas $L = 24$: pour $k = [0, 3]$, $M = 15$; pour $k = [1, 2]$, $M = 14$; pour $k = [4, 0]$, $M = 16$; maximum 16

Question 3

L'algorithme a été décrit à la **Question 1**! Je calcule M pour chaque valeur de a_0 et je mets à jour si besoin la variable $Mmax$ qui contiendra le maximum. Initialisation à 0 pour le cas où il n'y aurait pas assez de tissu pour le moindre produit. Surtout pas de double boucle ici, je choisis directement la valeur maximale de a_1 pour chaque a_0 comme expliqué précédemment. La variable k_1 n'est pas nécessaire mais elle améliore la lisibilité.

```
def coutRuban2Produits(a0,v0,a1,v1,L):
    Mmax=0;
    for k0 in range(L//a0+1):
        k1=(L-k0*a0)//a1
        M=k0*v0+k1*v1
        if M>Mmax:
            Mmax=M
    return Mmax
```

Question 4

Même principe et je peux appeler la fonction précédente pour chaque valeur de k_0 selon le principe général détaillé à la question suivante. Les formules sont ainsi similaires aux précédentes.

```
def coutRuban3Produits(a0,v0,a1,v1,a2,v2,L):
    Mmax=0;
    for k0 in range(L//a0+1):
        k1=(L-k0*a0)//a1
        M=k0*v0+coutRuban2Produits(a1,v1,a2,v2,L-k0*v0)
        if M>Mmax:
            Mmax=M
    return Mmax
```

Question 5

Lorsque n augmente, l'imbrication des boucles selon le principe précédent conduit en effet à une complexité exponentielle (et à une programmation récursive). Dans ce type de situation, pour conserver un temps de calcul acceptable, on choisit de consommer de la mémoire pour stocker les résultats successifs obtenus pour les valeurs croissantes de L.

Le principe général de la "programmation dynamique" est le suivant : une solution optimale pour le "gros problème" est issue de solutions optimales pour des "sous-problèmes". Ce principe se traduit ici par la formule de récurrence de l'énoncé, bien mal écrite d'ailleurs. En effet, si $M(x)$ n'est pas nul, il s'obtient en découpant l'un des produits i, i appartenant à l'ensemble E_x des valeurs $i \in [\![0, n[\![$ telles que $a_i \leq x$, puis en découpant de façon optimale le tissu restant, de longueur $x - a[i]$. Si l'on ne fait pas ce dernier découpage de façon optimale, le découpage global ne sera évidemment pas optimal ! Le gain associé à ce découpage, optimal pour le choix i du premier produit, est $v[i] + M(x - a[i])$. C'est pourquoi l'on obtiendra $M(x)$ en déterminant

$$\max_{i \in E_x} \left\{ v[i] + M(x - a[i]) \right\}.$$

C'est la "commutativité" du découpage, évoquée dans l'énoncé, qui justifie dans le fond ce principe et évite de recalculer de nombreuses fois les mêmes valeurs de M comme on le faisait dans les questions précédentes.

La justification ci-dessus n'était pas demandée le jour du concours ! Bien lire l'énoncé : "... en prenant pour acquise la formule..." !

Il ne reste plus qu'à remplir le tableau M selon le principe précédent, dans l'ordre bien sûr des valeurs croissantes de x, pour n'accéder ensuite qu'à des valeurs déjà calculées.

```
def coutRuban(a,v,n,L):
    M=[0]*(L+1)
    for x in range(1,L+1):
        for i in range(n):
            if a[i]<=x:
                Mi=v[i]+M[x-a[i]]
                if Mi>M[x]:
                    M[x]=Mi
    return M[L]
```

Compte tenu des deux boucles imbriquées, le temps d'exécution est un $O(n \cdot L)$.

Question 6

Il suffit de compléter le programme précédent, en mettant à jour $D[x]$ à chaque fois que l'on modifie $M[x]$. Une fois les deux tableaux M et D remplis, je récupère dans le tableau D la liste des découpes $D[x]$ successives comme indiqué dans l'énoncé.

Le temps d'exécution reste un $O(n \cdot L)$, puisque l'on a juste ajouté un $O(L)$ avec la dernière boucle.

```
def decoupageRuban(a,v,n,L):
    M=[0]*(L+1)
    D=[-1]*(L+1)
    for x in range(1,L+1):
        for i in range(n):
            if a[i]<=x:
                Mi=v[i]+M[x-a[i]]
                if Mi>M[x]:
                    M[x]=Mi;D[x]=i
    x=L;decoupe=[]
    while D[x]>=0:
        decoupe.append(D[x])
        x -= a[D[x]]
    return decoupe
```

Partie II. Coupe d'un ruban sans répétition

Question 7

Soit $i \in [\![0, n[\![$. Puisqu'ici l'on peut découper au plus une fois au plus un produit donné, le coût optimal $M(i+1, x)$ est obtenu :

- soit par découpage du produit i (à condition bien sûr que $x \geq a_i$) suivi d'un découpage optimal (même raisonnement que précédemment) des produits $0..i-1$ dans le tissu restant, donc pour un coût de $v[i] + M(i, x - a[i])$
- soit sans le produit i, donc pour un coût de $M(i, x)$

$M(i+1, x)$ est donc le plus grand de ces deux coûts :

$$M(i+1, x) = \max(v[i] + M(i, x - a[i]), M(i, x)).$$

Question 8

D'après la relation précédente, je peux calculer les valeurs $M(i, x)$, pour $i \in [\![0, n]\!]$ et $x \in [\![0, L]\!]$, en remplissant le tableau M ligne par ligne, suivant les valeurs croissantes de i. L'initialisation ($i = 0$) est triviale, puisqu'il n'y a aucun produit à découper ! Noter que le remplissage de la ligne pour $i = 1$ correspondra à une situation plus "palpable" : il s'agit de découper 0 ou 1 fois le produit 0, selon qu'on a, ou pas, suffisamment de tissu.

Pour pouvoir utiliser la syntaxe `M[i,x]`, j'utilise un tableau `numpy`, en supposant le module `numpy` chargé avec l'alias habituel `np`.
Le temps d'exécution est de nouveau ici un $O(n \cdot L)$.

```
def coutRubanSansRepetitions(a,v,n,L):
    M=np.zeros((n+1,L+1))
    for i in range(n):
        for x in range(L+1):
            if a[i]>x:
                M[i+1,x]=M[i,x]
            else:
                M[i+1,x]=max(v[i]+M[i,x-a[i]],M[i,x])
    return M[n,L]
```

Attention ! On peut choisir d'utiliser une liste de listes au lieu d'un tableau `numpy`, mais **il ne faut pas** l'initialiser par `[[0]*(L+1)]*(n+1)` où toutes les listes partageraient les mêmes données. Préférer dans ce cas `[[0]*(L+1) for k in range(n+1)]`.

Question 9

Le programme précédent utilise un tableau de taille $(n+1)\cdot(L+1) = O(n\cdot L)$, on essaye ici de réduire cet encombrement à un $O(L)$. Noter que le $O(2\cdot(L+1))$ de l'énoncé est un peu ricidule...

On s'aperçoit dans la version ci-dessus que seule la ligne précédente est utile, on peut donc facilement se contenter de deux lignes, qui contiendront alternativement la ligne précédente et la ligne courante. Pour éviter de tester la parité de i, j'utilise le couple de valeurs (ip, ic) qui vaudra alternativement $(0,1)$ et $(1,0)$, avec bascule à chaque passage dans la boucle :

```
def coutRubanSansRepetitionsV2(a,v,n,L):
    M=np.zeros((2,L+1))
    ip,ic=0,1
    for i in range(n):
        for x in range(L+1):
            if a[i]>x:
                M[ic,x]=M[ip,x]
            else:
                M[ic,x]=max(v[i]+M[ip,x-a[i]],M[ip,x])
        ip,ic=ic,ip
    return M[ic,L]
```