Algorithme Glouton

Le grand classique : Le monnayeur

On dispose des pièces de monnaie. Pour chaque valeur le nombre de pièces est non borné. Etant donnée somme à rendre, on veut que le nombre de pièces soit minimum.

- **Les pièces 1€, 2€, 5€, 10€, 20€ → la somme à payer 26€**

  1 billet de 20€

  1 billet de 5€

  1 pièce de 1€

- **Les pièces 1€, 6€, 7€ → la somme à payer 26€**
  2 billets de 7€ : 14€
  2 billets de 6€ : 12€

Quelle stratégie doit-on mettre en place pour déterminer à chaque fois le nombre de pièces/billets minimum que l'on doit rendre ?

- 26 / 20 → 1 billet de 20€
- 26 % 20 → il reste 6€ à rendre
- 6 / 5 → 1 billet de 5€
- 6 % 5 → il reste 1€ à rendre
- 1 // 1 → 1 pièce de 1€
- 1 % 1 → il reste 0€ à rendre…fin de l'algorithme

- 26 // 7 → 3 billets de 7€
- 26 % 7 → il reste 5€ à rendre
- 5 / 1 → 5 pièces de 1€
- 5 % 1 → il reste 0€ à rendre…fin de l'algorithme

Conclusion : *On vient de tester l'algorithme glouton. L'algorithme ne donne pas une solution optimale dans tous les cas.*

Challenge : Les pièces 2€, 5€, 10€, 20€

11 :

- 11 / 20 ⟶ 0 billet de 20€
- 26 % 20 ⟶ il reste 11€ à rendre
- 11 // 10 ⟶ 1 billet de 10€
- 11 % 10 ⟶ il reste 1€ à rendre
- 11 // 2 ⟶ 0 pièces de 2€
- 1 % 2 ⟶ il reste 1€...on ne peut plus continuer, fin de l'algorithme

Exercice préparatif au mini DS : les pièces 1€, 5€ et 10€

Comment l'algorithme glouton va rendre la monnaie qui est de 52€

- 50 / 10 = 5 billets de 10
- 50%10 = 2

Algorithme sac à dos

On dispose d'un sac à dos pouvant contenir au maximum une masse de 15kg.
On dispose de 15 boîtes :

- Une boîte verte : 12kg -> valeur : 12€
- Une boîte bleue : 2kg -> valeur : 2€
- Une boîte grise : 1kg -> valeur : 2€
- Une boîte orange : 1kg -> valeur : 1€
- Une boîte jaune : 4kg -> valeur : 10€

Il faut remplir le sac avec la plus grande quantité possible mais avec un gain le plus grand possible. On dispose d'autant de boîtes nécessaires de chaque couleur.

Couleur	Verte	Bleue	Grise	Orange	Jaune
Masse (kg)	12	2	1	1	4
Valeur (€)	4	2	2	1	10
$\frac{Valeur}{Masse}$ (€/kg)	0.33	1	2	1	2.5

Mettre en place une stratégie pour répondre à la question.

On choisit la boîte jaune (densité la plus forte) : 15//4 = 3 boîtes, il reste 15%3 = 3kg.

On choisit la boîte grise (densité la plus forte après la boîte jaune) : 3/1 = 3 boîtes et il va rester 3%1 = 0kg.

Question 1 :

Combien pèse le sac ?

Le sac pèse 15 kg car 3*4 + 1*3 = 15.

Question 2 :

Quelle est la valeur du sac en € ?

36 car 3*10 + 3*2 = 36.

A vous de jouer :

On possède le même type de boîtes mais le sac utilisé peut contenir 23kg au maximum.

Couleur	Verte	Bleue	Grise	Orange	Jaune
Masse (kg)	12	2	1	1	4
Valeur (€)	4	2	2	1	10
$\frac{Valeur}{Masse}$ (€/kg)	0.33	1	2	1	2.5

Stratégie :

On choisit la boîte jaune : 23//4 = 5 boîtes (50€), il reste 23%4 = 3kg.

On choisit la boîte grise (densité la plus forte après la boîte jaune) : 3/1 = 3 boîtes (6€) et il va rester 3%1 = 0kg.

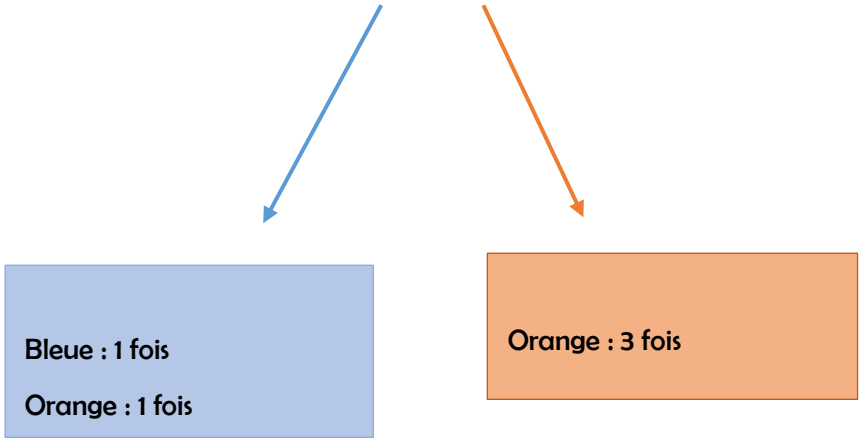
Mais, on rencontre un problème, on ne dispose plus de boîtes grises. Comment remplir alors efficacement ce sac de 23kg ? Existe-t-il plusieurs solutions ?

Couleur	Verte	Bleue	~~Grise~~	Orange	Jaune
Masse (kg)	12	2	~~1~~	1	4
Valeur (€)	4	2	~~2~~	1	10

$\frac{Valeur}{Masse}$ (€/kg)	0.33	1	2	1	2.5

Stratégie :

Boîte jaune : 23//4 = 3kg à remplir.



Pour choisir une des deux solutions possibles, on pourrait ajouter une contrainte supplémentaire : utiliser le moins de types de boîtes différentes.

Les outils utiles :

```
>>> liste = [1, 6, 9, 5, 7]
>>> liste.sort()
>>> liste
[1, 5, 6, 7, 9]
>>> liste.sort(reverse=True)
>>> liste
[9, 7, 6, 5, 1]
```

Attention, la méthode sort modifie la liste en place.

```
>>> liste = [1, 6, 9, 5, 7]
>>> liste_croissant = sorted(liste)
>>> liste
[1, 6, 9, 5, 7]
>>> liste_croissant
[1, 5, 6, 7, 9]
>>> liste_decroissant = sorted(liste, reverse=True)
>>> liste
[1, 6, 9, 5, 7]
>>> liste_decroissant
[9, 7, 6, 5, 1]
```

Attention, la fonction sorted ne modifie pas la liste passée en paramètres. Elle renvoie la liste rangée dans l'ordre croissant (dans l'ordre décroissant si le paramètre nommé reverse à la valeur True). Il faut donc assigner la valeur renvoyée à une variable.

Comprendre les dictionnaires :

```
>>> dictionnaire = {1: 5, 7: 0, 9: 3}
>>> dictionnaire[9]
3
>>> dictionnaire.keys()
dict_keys([1, 7, 9])
>>> dictionnaire.values()
dict_values([5, 0, 3])
```

```python
lt_vals = [1, 2, 5, 10, 20]
monnaie = 26


def glouton(listeValeurs: list, aRendre: int) -> dict:
    '''
    >>> lt_vals = [1, 2, 5, 10, 20]
    >>> monnaie = 26
    >>> glouton(lt_vals, monnaie)
    {20: 1, 10: 0, 5: 1, 2: 0, 1: 1}

    '''
    # On doit ranger la liste dans l'ordre décroissant
    liste_decroissant = sorted(listeValeurs, reverse=True)
    # Dictionnaire des résultats
    resultat = {}

    for piece_billet in liste_decroissant:
        cle = piece_billet
        nb_piece_billet = aRendre//piece_billet
        aRendre = aRendre%piece_billet
        resultat[cle] = nb_piece_billet

    return resultat
```

```
>>> glouton(lt_vals, monnaie)
{20: 1, 10: 0, 5: 1, 2: 0, 1: 1}
```

- Liste : décroissant
- Création dictionnaire vide
- Pour chaque valeur de pièce, de la liste ordonnée dans l'ordre décroissant, on réalise les opérations suivantes :
    - La valeur de la pièce que l'on teste est une clé du dictionnaire.
    - On calcule le nombre de pièces à rendre (division euclidienne).
    - On met à jour la monnaie à rendre. (Modulo)
    - On met à jour le dico (clé : valeur).
- On renvoie le résultat. (Return)

```
>>> liste_pieces_billets = [2, 5, 10, 50]
>>> rendre = 87
>>> glouton(liste_pieces_billets, rendre)
{50: 1, 10: 3, 5: 1, 2: 1}
```