

Corrigé sujet 13 - Année : 2023

[Sujet 13 - 2022 ↴](#)

1. Exercice 1



```

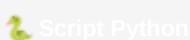
1 def recherche(a, tab):
2     nb_occurrence = 0
3     for elt in tab:
4         if elt==a:
5             nb_occurrence +=1
6     return nb_occurrence

```

Commentaires

C'est un exercice classique de parcours d'un itérable en comptant les occurrences d'apparition d'une valeur. Un parcours par élément suffit, les indices des occurrences n'étant pas utilisées.

2. Exercice 2



```

1 def rendu_monnaie(somme_due, somme_versee):
2     pieces = [1, 2, 5, 10, 20, 50, 100, 200]
3     rendu = [] #(1)
4     a_rendre = somme_versee - somme_due #(2)
5     i = len(pieces) - 1
6     while a_rendre > 0 : #(3)
7         if pieces[i] <= a_rendre :
8             rendu.append(pieces[i]) #(4)
9             a_rendre = a_rendre - pieces[i]
10        else :
11            i = i-1
12    return rendu

```

1. La liste des pièces à rendre, initialisée à []
2. La somme à rendre, initialisé à s_versee - s_due
3. La condition d'arrêt, plus rien à rendre
4. C'est l'algorithme glouton classique pour le rendu de monnaie (les pieces sont rangées dans l'ordre). Si la pièce est inférieure à la somme à rendre, on l'ajoute au rendu et on diminue la somme à rendre. Sinon on passe à la pièce

suivante.

Attention

1. On utilise ici une liste de pièces classées par ordre croissant de valeurs, cela oblige donc à commencer par la fin de la liste. C'est ce qui explique le parcours de la liste "à l'envers" : initialisation de `i` à `len(pièces)-1` puis décrémentation de `i`.
2. La fonction utilise deux arguments `s_versee` et `s_due` pour calculer la somme à rendre (`s_versee-s_due`), on pourrait directement une fonction qui prend en argument la somme à rendre.