

Code de partage avec Cappytale : 21f6-11737468

On utilisera la bibliothèque numpy sous l'alias np

Introduction

Un algorithme est dit **glouton** lorsqu'il cherche à résoudre un problème global (objectif) par étapes en prenant, à chaque étape, le « meilleur choix » possible (on parle de choix *localement optimal*), en général celui le rapprochant au maximum de son objectif.

Bien souvent, ces algorithmes ne donnent pas des résultats globalement optimaux, mais ils peuvent donner une solution approchant en un temps de calcul raisonnable, d'où leur intérêt.

Exercice 1 - rendre la monnaie

On souhaite modéliser sous Python le rendu de monnaie par un distributeur par exemple : comment rendre x euros en pièces de monnaie ?

On pose : $L=[2, 1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01]$ contenant les valeurs des pièces (en euros, 8 au total) dans l'ordre décroissant.

La liste du nombre de pièces répondant à la question sera donnée sous forme d'une liste également.

Ainsi, pour un montant 6,42 euros payé avec un billet de 10 euros, le distributeur doit rendre 3,58 euros en pièces et le programme renverra :

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

correspondant à $3,58 = 1 \times 2 + 1 \times 1 + 1 \times 0,5 + 1 \times 0,05 + 1 \times 0,02 + 1 \times 0,01$ (sous réserve de disponibilité des pièces).

L'algorithme consiste à commencer par rendre « le plus de pièces de valeur la plus élevée disponible », puis recommencer avec le reliquat et des pièces de valeurs plus faibles.

Dans l'exemple $x = 3,58$, on rend successivement :

1 pièce de 2 euros (le max) ; il reste 1,58 euros

1 pièce de 1 euro ; il reste 0,58 euro

1 pièce de 0,5 euro ; il reste 0,08 euro

0 pièce de 0,2 euro ; il reste 0,08 euro

0 pièce de 0,1 euro ; il reste 0,08 euro

1 pièce de 0,05 euro ; il reste 0,03 euro

1 pièce de 0,02 euro ; il reste 0,01 euro

1 pièce de 0,01 euro ; il reste 0,00 euro : le compte est bon.

1. on considère dans cette question les pièces de p euros (p fixé) et un montant de x euros (à rendre)
 - (a) Que renvoie la commande `np.floor(x/p)` ?
 - (b) Que renvoie la commande `x-p*np.floor(x/p)` ?
2. Compléter la fonction 'Rendu' suivante renvoyant la liste du nombre de pièces nécessaires pour rendre x euros.

On suppose que x est donné en entrée comme un nombre à 2 décimales.

```
def Rendu(x):
    reliquat=x
    L = [2, 1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01]
    R=[]
    #for k in range(8):

        #NombrePiece=...
        #R.append(...)
    return R
```

Exercice 2 - suite de Fibonacci et Z-décomposition

On considère la suite définie par $u_0 = 0$, $u_1 = 1$ et, pour tout $n \in \mathbb{N}$, $u_{n+2} = u_{n+1} + u_n$

1. Compléter le code de la fonction `u` suivante permettant de calculer sous Python u_n pour une valeur n donnée en entrée.

```
def u(n):
    v=0; w=1
    # v et w contiendront dans cet ordre deux termes consécutifs de la suite
    for j in range(...):
        u=...
        v=...
        w=...
    return ...
```

2. Compléter le code de la fonction `ListeFib` suivante renvoyant sous Python une liste contenant les termes $u_0, u_1, \dots, u_n$ vérifiant : $u_n \leq x < u_{n+1}$ où x est un entier strictement positif donné en entrée.

```
def ListeFib(x):
    v=1
    L=[0]
    while ...:
        L.append(v)
        v=L[-1]+L[-2]
    return L
```

3. On dit qu'un nombre entier $n \geq 1$ admet une Z-décomposition s'il s'écrit comme somme de termes de la suite $(u_k)_{k \geq 2}$ sans que deux termes de cette somme ne soient consécutifs dans la suite u

Par exemple : $10 = u_6 + u_3 = 8 + 2$ et $100 = u_{11} + u_6 + u_4 = 89 + 8 + 3$ sont des Z-décompositions de 10 et de 100

$u_5 + u_4 + u_3 = 5 + 3 + 2 = 10$, mais ce n'est pas une Z-décomposition de 10 : les termes u_5 et u_4 sont consécutifs.

On admettra que tout entier non nul possède une unique Z-décomposition.

L'algorithme pour trouver cette décomposition est proche de celui pour le rendu de monnaie : pour une valeur entière $x > 0$, on cherche l'indice n maximal tel que $u_n < x$, puis on pose $y = x - u_n$ et on recommence avec y au lieu de x

(a) En quoi peut-on affirmer que c'est un algorithme « glouton » ?

- (b) Compléter la fonction `ZD` suivante permettant de renvoyer une Z-décomposition d'un entier $x > 0$

```
def ZD(x):
    L=ListeFib(x)
```

```
n=len(L)-1
Z=[]
y=x
while y>0:
    while L[n]>y:
        n = n-1
    Z.append(...)
    y = y -
return Z
```