

```
1. def glouton(s,P) :
    if s == 0 :
        return []
    else :
        i = 0
        while s < P[i] :
            i += 1
        return [P[i]] + glouton(s-P[i],P)
```

2. Si $P=[10,5,2,1]$, `glouton(49,P)` renvoie $[10, 10, 10, 10, 5, 2, 2]$ qui est un rendu optimal alors que si $P=[30,24,12,6,3,1]$, `glouton(49,P)` renvoie $[30, 12, 6, 1]$ qui n'est pas optimal car $[24,24,1]$ comporte une pièce de moins.

3. Il faut commencer par une pièce $p_i \leq s$, il restera alors la somme $s - p_i$ à rembourser. Le nombre minimal de pièces nécessaires pour rembourser $s - p_i$ est par définition N_{s-p_i} donc pour pouvoir rembourser de façon optimale la somme s , il faut la pièce p_i et N_{s-p_i} pièces, pour la valeur de l'indice i donnant la plus petite valeur.

4. On commence par chercher la liste L des pièces utilisables puis on applique la formule précédente récursivement :

```
def rendu(s,P) :
    if s == 0 :
        r = 0
    else :
        L = []
        for p in P :
            if p <= s :
                L.append(p)
        r = 1 + min([rendu(s-p,P) for p in L])
    return r
```

5. On « mémorise » la fonction précédente avec un dictionnaire et une sous-fonction :

```
def renduMemo(s,P) :
    d = {}
    def renduDico(s,P) :
        if s not in d :
            if s == 0 :
                r = 0
            else :
                L = []
                for p in P :
                    if p <= s :
                        L.append(p)
                r = 1 + min([renduDico(s-p,P) for p in L])
            d[s] = r
        return d[s]
    return renduDico(s,P)
```

6. On construit successivement tous les rendus pour les valeurs $0, 1, \dots, s$ que l'on mémorise dans une liste L :

```
def BasEnHaut(s,P) :
    L = [0 for _ in range(s+1)]
    for k in range(1,s+1) :
        M = []
        for p in P :
            if p <= k :
                M.append(p)
        L[k] = 1 + min([L[k-p] for p in M])
    return L[-1]
```

7. L'initialisation de L est en $O(s)$ puis on effectue $s + 1$ passages dans la boucle `for` : la construction de M est en $O(1)$ (ou plutôt $O(p)$ où p est le nombre de pièces, que l'on considère fixé), le calcul du minimum est lui aussi en $O(1)$. La complexité de la fonction `BasEnHaut` est donc en $O(s)$.

8. On a $N_{10} = 3$ donc il faut 3 pièces pour rembourser 10. Par construction, on a $N_{10} = 1 + \min\{N_9, N_7, N_6\}$; ici, on a $N_{10} = 1 + N_6$ (ou $1 + N_7$) donc on peut rembourser 10 avec une pièce de $10 - 6 = 4$ puis 2 pièces pour rembourser les 6 restants. Comme $N_6 = 1 + N_2$, on peut rembourser 6 avec une pièce de $6 - 2 = 4$ et $N_2 = 1$ autre pièce. Comme $N_2 = 1 + N_0$, il suffit d'une pièce de $2 - 0 = 2$. Au final, une possibilité est $10 = 4 + 4 + 2$.
9. On commence par modifier la fonction **BasEnHaut** de façon à ce qu'elle renvoie la liste **L** qu'elle construit. Une fois qu'on dispose de cette liste, on raisonne comme dans l'exemple précédent : à partir de s on cherche un indice i de sorte que $L[s] = 1 + L[s - P[i]]$, on rembourse donc la pièce $P[i]$ puis on continue avec la somme $s - P[i]$ jusqu'à ce que la somme à rembourser soit nulle :

```
def listeRendus(s,P) :
    L = [0 for _ in range(s+1)]
    for k in range(1,s+1) :
        M = []
        for p in P :
            if p <= k :
                M.append(p)
        L[k] = 1+min([L[k-p] for p in M])
    return L

def Rendu(s,P) :
    L = listeRendus(s,P)
    Q = []
    while s > 0 :
        trouve = False
        i = 0
        while not trouve :
            if P[i] <= s and L[s] == 1+L[s-P[i]] :
                p = P[i]
                trouve = True
            i += 1
        Q.append(p)
        s = s-p
    return Q
```

10. Pour ne pas avoir trop de modification à faire, il suffit de prendre $N_s = +\infty$ pour un rendu de monnaie impossible ; la formule de la question 3 reste alors valable. Dans le cas de base de la fonction, les rendus impossibles sont ceux pour lesquels $s < P[-1]$ (la plus petite pièce) et on initialise aussi avec 1 les valeurs de N_s pour lesquelles s est une des pièces (s in P) :

```
def renduMemo2(s,P) :
    d = {}
    def renduDico(s,P) :
        if s not in d :
            if s < P[-1] :
                r = float('inf')
            elif s in P :
                r = 1
            else :
                L = []
                for p in P :
                    if p <= s :
                        L.append(p)
                r = 1 + min([renduDico(s-p,P) for p in L])
            d[s] = r
        return d[s]
    return renduDico(s,P)
```

Cette fonction renvoie $+\infty$ pour une somme $s = 0$; il faut rajouter $r = 0$ si $s = 0$ pour inclure ce cas si besoin.