Correction TD8

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. $N_s = 1 + \min\{N_{s-p_i}, i \in [\![1,k]\!], s - p_i \geqslant 0\}$

4. Les sous problèmes sont : déterminer les N_{s-p_i}.
Si $s = 49$ et `P=[10,5,2,1]`, parmi les sous problèmes, on doit déterminer N_{39} (si on utilise la pièce 10) et N_{44} (avec la pièce 5) qui demandera lui même de déterminer aussi N_{39} (à nouveau avec la pièce 5).

5. On commence par chercher les pièces utilisables (pour lesquelles $s - p_i \geqslant 0$) puis on cherche le rendu avec le moins de pièces (donc une liste de longueur minimale). La liste `L` contient les couples des rendus possibles de $s - p_i$ et la valeur de p_i pour pouvoir reconstruire le rendu final

```
def dynamique(s,P) :
    d = {}
    def dyn(s,P) :
        if s not in d :
            if s == 0 :
                r = []
            else :
                # recherche des pièces utilisables
                P_poss = []
                for k in P :
                    if k<=s :
                        P_poss.append(k)
                # liste de tous les rendus avec ces pièces
                L = [[dyn(s-k,P_poss),k] for k in P_poss]
                # recherche du rendu avec le moins de pièces
                m = L[0]
                for k in range(1,len(L)) :
                    if len(L[k][0])<len(m[0]) :
                        m = L[k]
                r = [m[1]]+m[0]
            d[s] = r
        return d[s]
    return dyn(s,P)
```

6. On obtient cette fois les solutions optimales

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

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

8. Si on note k le nombre de pièces (qui est fixé), les listes `P_poss` et `M` possèdent au plus k éléments donc la complexité de `BasEnHaut` est $O(s)$.

9. On commence par la pièce 5, puis la 2 et on est bloqué pour rendre 1 donc la démarche gloutonne ne fournit pas de solution ; il en existe pourtant une qui est `[2,2,2,2]`

10.
```
def dynamique(s,P) :
    d = {}
    def dyn(s,P) :
        if s not in d :
            if s == 0 :
                r = []
            else :
                # recherche des pièces utilisables
                P_poss = []
                for k in P :
                    if k<=s :
                        P_poss.append(k)
                if len(P_poss) == 0 : # plus aucune pièce utilisable
                    r = [-1]
                else :
                    L = [[dyn(s-k,P_poss),k] for k in P_poss]
                    # on cherche un premier rendu possible
                    i = 0
                    trouve = False
                    while not trouve and i<len(L) :
                        if L[i][0] != [-1] :
                            trouve = True
                        i += 1
                    if not trouve : # il n'y en a pas
                        r = [-1]
                    else :
                        # on cherche le rendu avec le moins de pièces
                        m = L[i-1]
                        for k in range(i,len(L)) :
                            if L[k][0] != [-1] and len(L[k][0])<len(m[0]) :
                                m = L[k]
                        r = [m[1]]+m[0]
            d[s] = r
        return d[s]
    return dyn(s,P)
```