```python
# correction_glouton.py

##---- Importation de Modules  -------##
from random import randint


"""
Rendu de monnaie

"""

# système monetaire
systeme_monetaire  =    [1,2,5,10,20,50,100,200]


def rendu_glouton(somme_a_rendre, systeme_monnaie):
    """
    Glouton on rend en premier la pièce de plus forte valeur
    """
    # liste des pièces à rendre
    lst_pieces = []
    # indice de la première pièce comparer à la somme à rendre
    i = len(systeme_monnaie) - 1
    while somme_a_rendre > 0:
        valeur = systeme_monnaie[i]
        if somme_a_rendre < valeur:
            i -= 1
        else:
            lst_pieces.append(valeur)
            somme_a_rendre -= valeur
    return lst_pieces


#tests de bon fonctionnement
assert rendu_glouton(55,systeme_monetaire) == [50,5]
print(rendu_glouton(55,systeme_monetaire))
assert rendu_glouton(57,systeme_monetaire) == [50,5,2]
print(rendu_glouton(57,systeme_monetaire))
assert rendu_glouton(58,systeme_monetaire) == [50,5,2,1]
print(rendu_glouton(58,systeme_monetaire))

"""
Occupation de temps

"""

def generer_intervalles(n):
    """
    creation d'une liste d'intervalles
    """
    C = []
    for i in range(n):
        deb = randint(0,9)
        inter = randint(1,10-deb)
        fin = deb + inter
```

```
        C.append([deb,fin])
    return C

C = generer_intervalles(20)

def nommer(C) :
    """
    Donne des noms aux intervalles
    """
    for i in range (len(C)):
        C[i].append("C_"+str(i))

nommer(C)


"""

for i in range(len(C)):
    print(C [i][1] , end =" ")
print()
"""

#tri des intervalle par valeur croissante de fin
def echange( t,i,j):
    temp = t[i]
    t[i]= t[j]
    t[j]=temp

def tri_par_sélection(t):
    """
    Tri la liste en fonction dans l'ordre croissant des durées de fin

    """
    for i in range(len(t)-1):
        m = i
        for j in range(i+1,len(t)):
            if t[j][1] <t[m][1] :
                m = j
        echange(t,i,m)
       # print(t)

tri_par_sélection(C)

print(C)



def organiser(liste_intervalle):
    """

    Etablit une liste des Clients qui peuvent être successivement
servi
    """
    #Tableau du planning initiation
    planning = [liste_intervalle[0]]
```

```
    j = 0
    for i in range(1,len(liste_intervalle)):
        """
        On trouve l'intervalle suivant dont l'instant de début
correspond
        au plus près de  celui de fin de l'intervalle précédent.

        """
        if liste_intervalle[i][0] >= liste_intervalle[j][1] :  # >=
pour intervalles disjoints
            planning.append(liste_intervalle[i])
            #on passe au suivant
            j = i
    return planning

print(organiser(C))



# test avec un liste fixe
test =  [[0, 2, 'C_14'], [2, 3, 'C_7'], [0, 4, 'C_10'], [0, 4,
'C_12'], [4, 5, 'C_11'], [6, 7, 'C_1'], [2, 7, 'C_3'], [8, 9, 'C_9'],
[4, 9, 'C_4'],\
     [2, 9, 'C_13'], [1, 9, 'C_15'], [1, 9, 'C_16'],\
     [1, 9, 'C_18'], [9, 10, 'C_5'], [9, 10, 'C_0'], [7, 10, 'C_2'],
[1, 10, 'C_8'], [9, 10, 'C_17'], [9, 10, 'C_6'], [9, 10, 'C_19']]
print(organiser(test))
assert organiser(test) == [[0, 2, 'C_14'],[2, 3, 'C_7'],[4, 5,
'C_11'],[6, 7, 'C_1'], [8, 9, 'C_9'],[9, 10, 'C_5']]

"""
Le glouton ne revenant pas en arrière on remarque un vide de une
heure  entre C_1 et C_9

"""

"""

Sac à dos

"""



"""
definition de différentes listes d'articles
2-uplets  poids valeurs

"""
articles = [[1,1],[2,1],[2,2],[4,12],[10,4],[2,3]]
articles2 = [[5,2],[7,3],[2,1],[12,7],[10,9]]



def valeurpoids (liste):
    """
```

```
    Ajoute le 3-uplets rapports valeur/poids

    """
    for el in liste:
        el.append(el[0]/el[1])

valeurpoids(articles)
print(articles)

def retourne_cle(liste):
    return liste[2]


articles.sort(reverse=True, key = retourne_cle)

print(articles)

def sac_a_dos(liste, poids):
    """
    optimise le remplissge du sac à dos en partant du 3-ulets
    avec le plus grand rapport poids/valeur
    retourne la liste de valeur et leur somme
    """
    valeurs = []
    deb = len(liste) -1
    print(deb)
    somme = 0
    poids_dans_le_sac =0
    while poids >= 0 and deb >=0:
        #on sélectionne le m^me article tant que c'est possible
        if poids >= liste[deb ][1]:
            valeurs.append(liste[deb][0])
            somme += liste[deb][0]
            poids -= liste[deb ][1]
            poids_dans_le_sac += liste[deb ][1]
        else :
            deb -= 1
    return valeurs, somme, poids,poids_dans_le_sac

a,b,c,d = sac_a_dos(articles2,15)

print(f"Sac à dos liste valeurs {a} valeur {b} poids total {c} poids
dans le sac {d} ")

a,b,c,d = sac_a_dos(articles,15)

print(f"Sac à dos liste valeurs {a} valeur {b} poids total {c} poids
dans le sac {d} ")
```