```python
import numpy as np

ListeMasses=[13,12,8,10]
ListeValeurs=[5,4,3,3]

L1=[i for i in range(1,100)]
L2=[abs(np.floor(50*np.sin(i))) for i in range(1,100)]

def imax(liste):
    '''indice du maximaum dans une liste'''
    maxi=-float('inf')
    ind=0
    n=len(liste)
    for i in range(n):
        if liste[i]>maxi:
            ind=i
            maxi=liste[i]
    return ind

def Glouton1(M,ListeMasses,ListeValeurs):
    n=len(ListeMasses)
    pris=[0 for i in range(n)] #aucun objet n'est pris au début
    Mrestante=M  #la masse restante pour remplir le sac
    valeur=0  #la valeur totale des objets du sac
    test=True  #pour savoir s'il reste des objets que l'on peut mettre dans le sac

    while test: #on tourne tant qu'il reste des objets possibles à mettre
        #recherche de l'objet de plus grande valeur qui reste et qui passe
        maxi=-float('inf')
        for i in range(n):
            if ListeValeurs[i]>maxi and pris[i]==0 and ListeMasses[i]<=Mrestante:
                ind=i
                maxi=ListeValeurs[i]
        if maxi==-float('inf'): #aucun objet ne peut plus passer
            test=False #on va arrêter le programme
        if test==True: #si on a trouvé un objet à rajouter
            pris[ind]=1
            Mrestante-=ListeMasses[ind]
            valeur+=ListeValeurs[ind]

    liste_objets=[i for i in range(n) if pris[i]==1]
    return liste_objets,valeur

def Glouton2(M,ListeMasses,ListeValeurs):
    n=len(ListeMasses)
    ListeRatio=[ListeValeurs[i]/ListeMasses[i] for i in range(n)]
    pris=[0 for i in range(n)] #aucun objet n'est pris au début
    Mrestante=M #la masse restante pour remplir le sac
    valeur=0  #la valeur totale des objets du sac
    test=True  #pour savoir s'il reste des objets que l'on peut mettre dans le sac

    while test: #on tourne tant qu'il reste des objets possibles à mettre
        #recherche de l'objet de plus grande valeur qui reste et qui passe
        maxi=-float('inf')
        for i in range(n):
            if ListeRatio[i]>maxi and pris[i]==0 and ListeMasses[i]<=Mrestante:
                ind=i
                maxi=ListeRatio[i]
        if maxi==-float('inf'): #aucun objet ne peut plus passer
            test=False #on va arrêter le programme
        if test==True: #si on a trouvé un objet à rajouter
            pris[ind]=1
            Mrestante-=ListeMasses[ind]
            valeur+=ListeValeurs[ind]

    liste_objets=[i for i in range(n) if pris[i]==1]
    return liste_objets,valeur

def Dynamique1(M,ListeMasses,ListeValeurs):
    n=len(ListeMasses)
    V=np.zeros((n+1,M+1),dtype=int)
    for k in range(1,n+1):
```

```
73          for m in range(M+1):
74              if ListeMasses[k-1]>m:
75                  V[k,m]=V[k-1,m]
76              else:
77                  V[k,m]=max([V[k-1,m],V[k-1,m-ListeMasses[k-1]]+ListeValeurs[k-1]])
78      return V[n,M]
79
80  def Dynamique2(k,M,ListeMasses,ListeValeurs):
81      n=len(ListeMasses)
82      if k==0:
83          return 0
84      if ListeMasses[k-1]>M:
85          return Dynamique2(k-1,M,ListeMasses,ListeValeurs)
86      else:
87          return max([Dynamique2(k-1,M,ListeMasses,ListeValeurs),
88                      Dynamique2(k-1,M-ListeMasses[k-1],ListeMasses,ListeValeurs)+
                        ListeValeurs[k-1]])
89
90  #nous proposons aussi une version avec memoisation (sinon dès que le problème devient
    grand on est bloqué)
91  #on suppose dans ce cas que ListeMasses et ListeValeurs ont été déclarées comme des
    variables globales.
92  dico={}
93
94  def Dynamique2_mem(k,M):
95      if (k,M) in dico:
96          return dico[(k,M)]
97
98      if k==0:
99          dico[(k,M)]=0
100         return 0
101     if ListeMasses[k-1]>M:
102         dico[(k,M)]=Dynamique2_mem(k-1,M)
103         return dico[(k,M)]
104     else:
105         dico[(k,M)]=max([Dynamique2_mem(k-1,M),
106                     Dynamique2_mem(k-1,M-ListeMasses[k-1])+ListeValeurs[k-1]])
107
108         return dico[(k,M)]
109
110
111
112
113
```