a) L'instruction `len( donnees[1] )` renvoie 6, l'instruction `donnees[0]` renvoie l'élément d'indice 0 du tableau `donnees` soit `['patient','genre','age','taille(cm)','poids(kg)','etat']`, enfin l'instruction `donnees[1][4]`renvoie l'élément d'indice 4 du tableau `donnees[1]`, soit "49".

b) On corrige ainsi :

```
def convert(tab : list) -> None : # deux-points
    tab[0] = int( tab[0] )    # crochets, égalité, parenthèses, crochets
    for i in range( 2, len(tab) ) : # in, range, parenthèses, deux-points
        tab[i] = int( tab[i] ) # crochets, égalité, parenthèses, crochets
```

c) Il s'agit d'un algorithme de type somme pour lequel on rappelle la fonction précédente sur chaque tableau de `donnees` (*i.e.* à partir de l'indice 1),

```
def convertDonnees(donnees : list) -> None :
    for i in range( 1, len(donnees) ) :
        convert( donnees[i] )
```

d) Puisque `donnees[1][0]` renvoie 1, la commande `donnees[ donnees[1][0] ] = donnees[1]` , qui renvoie le tableau `[1,'F',35,167,49,87]`.

De même, la commande `donnees[ donnees[i][0] ]` renvoie le tableau `donnees[i]`.

On remarque que le tableau `donnees[i]` s'auto-référence dans le tableau de tableaux `donnees`. Autrement dit connaissant n'importe quel tableau `[a,B,c,d,e,f]`, la valeur a nous donne sa position dans `donnees`.

e) Il faut 1 octet par élément du tableau, soit 6 octets.

f) La variable `donnees` contient une 1ère ligne à 38 octets puis que des lignes à 6. Ainsi puisque $1024-38 = 6\times 164+2$, on pourra enregistrer 164 données patients sur 1 Kio.

g)

ligne	donnees	i	col	j	sortie
1	[["patient", ...], ..., [..., 66]]	3			
2	-	-	[]		
3	-	-	-	1	
4	-	-	[167]	-	
3	-	-	-	2	
4	-	-	[167,176]	-	
3	-	-	-	3	
4	-	-	[167,176,176]	-	
5	-	-	-	-	[167,176,176]

h) En réécrivant à l'aide d'une boucle `while`,

```
def colonneI(donnees : list, i : int) -> list :
1   col = []
1   j = 1
2   while j < len(donnees) :
2       col = col + [ donnees[j][i] ]
2       j = j + 1
1   return col
```

Ainsi $ct(n) = 5+6n$, soit une complexité temporelle linéaire selon n (la longueur du tableau `donnees`).

i) L'âge correspondant à la colonne d'indice 2 de `donnees`, on propose la commande `ages = colonneI(donnees, 2)`.

j) Il s'agit d'un algorithme de type somme, on propose ensuite l'appel suivant.

```
def ageMax(tab : list) -> int} :
    vmax = -float( 'inf' )    # on pourrait mettre 0 car les âges sont positifs
```

```
    for i in range( len( tab ) ) :
        if vmax < tab[i] :
            vmax = tab[i]
    return vmax

ages = colonneI(donnees, 2)
print( f"L'âge maximal est : {ageMax(ages)}" ) # ou print( ageMax(ages) )
```

k) Chaque sous-tableau de `donnees` s'auto-référençait, mais l'appel de `colonneI()` a décalé de 1 ce référencement. Autrement dit `ages[i]` correspond au patient $i+1$.
On peut donc écrire l'un des algorithmes de type somme suivants (le 1er étant plus efficace) :

```
def agesMax(tab : list) -> list},
    vmax = -float( 'inf' )
    tPatients = []
    for i in range( len( tab ) ) :
        if vmax < tab[i] :
            vmax = tab[i]
            tPatients = [ i+1 ]
        elif vmax == tab[i] :
            tPatients = tPatients + [ i+1 ]
    return tPatients
```

```
def agesMax(tab : list) -> list},
    vmax = ageMax( tab )
    tPatients = []
    for i in range( len( tab ) ) :
        if vmax == tab[i] :
            tPatients += [ i+1 ]
    return tPatients
```

l) Il s'agit là encore d'un algorithme de type somme, cette fois-ci sans décalage d'indice.

```
def bonneSante(donnees : list, binf : int, bsup : int) -> list :
    tPatients = []
    for i in range( 1, len( donnees ) ) :
        if binf <= donnees[i][5] and donnees[i][5] <= bsup :
            tPatients += [ i ]
    return tPatients
```

m) Bien que ce ne soit pas optimal (mais ça restera en $\mathcal{O}(n)$), on peut réutiliser la fonction précédente pour faire un prétraitement des groupes. Il suffira ensuite de couper les groupes dès que l'on a atteint 10 femmes et 10 hommes. Cette 'coupure' correspond à un algorithme de type seuil.

```
def groupes(donnees : list) -> tuple :
    gpSanteP = bonneSante(donnees, 0, 79)
    gpBSante = bonneSante(donnees, 80, 100)
    n = len(donnees)
    nbF, nbH = 0, 0
    gp1, i = [], 0
    while i < n and ( nbF < 10 or nbH < 10 ) :
        gp1 += [ gpSanteP[i] ]
        if donnees[ gpSanteP[i] ][1] == 'F' :
            nbF += 1
        else :
            nbH += 1
        i += 1
    # on recommence avec le groupe en bonne santé
    nbF, nbH = 0, 0
    gp2, i = [], 0
    while i < n and ( nbF < 10 or nbH < 10 ) :
        gp1 += [ gpBSante[i] ]
        if donnees[ gpBSante[i] ][1] == 'F' : nbF += 1
        else : nbH += 1
        i += 1
    return gp1, gp2
```