1. Si le cavalier va en $b1$, les celles seule cases permises ensuite ($a3$, $c3$ et $d2$) ont déjà été atteintes. Comme il reste de cases non visitées, il n'accomplit pas sa mission.

Si, par l'absurde, il devait accomplir sa mission, alors la seule possibilité est de terminer sur la case $b1$. Or, cette case n'est plus accessible car aucune case non atteinte ne permet d'y accéder. Il y a une impossibilité ; le cavalier a mal débuté son parcours et ne peut accomplir sa mission.
2. On constate que l'indice de $[i,j]$ est $8i+j$. La fonction est donc élémentaire.

```
indice=lambda l:8*l[0]+l[1]
```

3. Si $n = 8i+j$ avec $j \in [\![0,7]\!]$ alors cette égalité est une division euclidienne. i et j sont respectivement le quotient et le reste de la division de n par 8..

```
coord=lambda n:[n//8,n%8]
```

4. La fonction renvoie l'ensemble des indices des cases atteignables (en un coup) depuis une case d'indice donné (dans un ordre fixé : on part du point situé au dessus et à droite et on tourne dans le sens horaire). Ainsi

```
CasA(0)=[17,10]
CasA(39)=[45,54,22,29]
```

5. La case numéro n de `ListeCA` contient `CasA(n)`. On peut donc utiliser une définition "en compréhension" pour cette liste.

```
def Init():
 global ListeCA,ListeCoups
 ListeCoups=[]
 ListeCA=[CasA(n) for n in range(64)]
```

6. Comme `CasA[0]=[17,10]`, c'est cette valeur qui est correcte et aucune de celles proposées.
7. a) On suit l'algorithme donné. Notons que si une case est accessible en un coup à partir d'une autre, la réciproque est aussi vraie ; ainsi, `remove` n'engendrera pas d'erreur. Sinon, il aurait fallu tester la présence de l'élément avant de le retirer (`if e in L: ...`).
Remarque : On notera l'introduction de la variable locale `test` qui permet de garder en mémoire l'apparition d'une liste vide dans le processus (situation critique).

```
def OccupePosition(n):
    global ListeCA,ListeCoups
    ListeCoups.append(n)
    test=False
    for k in ListeCA[n]:
        ListeCA[k].remove(n)
        if len(ListeCA[k])==0:
                test=True
    return(test)
```

b) Appliquons l'algorithme donné :

```
def LiberePosition():
    global ListeCA,ListeCoups
    n=ListeCoups.pop()
    for k in ListeCA[n]:
        ListeCA[k].append(n)
```

c) Comme indiqué dans l'énoncé, on est attentif à utiliser une copie locale de `ListeCA[n]` et non un alias.

```
def TestePosition(n):
    global ListeCA,ListeCoups
    Test=OccupePosition(n)
    if Test:
        if len(ListeCoups)==63:
                return(True)
        else:
            LiberePosition()
            return(False)
    else:
        liste=1*ListeCA[n]
        for k in liste:
            if TestePosition(k):
                return(True)
        LiberePosition()
        return(False)
```

8. a) Il suffit de retourner la longueur de `listeCA[n]`.

```
def valuation(n):
    global ListeCA
    return(len(ListeCA[n]))
```

b) On procède récursivement. Si l'une des listes est vide, c'est facile. Sinon, on repère le plus grand élément (dernier élément de `A` ou de `B`), on fusionne les autres et on ajoute l'élément mis à part (en bout de liste puisque c'est le plus grand).

Remarque : On peut aussi travailler à partir du plus petit élément.

```
def Fusion(A,B):
 if len(A)==0: return(B)
 elif len(B)==0: return(A)
 else:
  a,b=A.pop(),B.pop()
  if valuation(B[-1])>valuation(A[-1]):
   return(Fusion(A,B[:-1])+B[-1:])
  else:
   return(Fusion(A[:-1],B)+A[-1:])
```

c) On procède récursivement en découpant la liste en deux et en fusionnant la version triée des deux morceaux.

```
def TriFusion(L):
 if len(L)<2:
  return(L)
 else:
  return(Fusion(TriFusion(L[:len(L)//2]),
        TriFusion(L[len(L)//2:])))
```

d) On ne copie pas `ListeCA[n]` avant de tester les coups jouables après n mais plutôt une version triée de cette liste.

```
def TestePosition(n):
    global ListeCA,ListeCoups
    Test=OccupePosition(n)
    if Test:
        if len(ListeCoups)==64:
                return(True)
        else:
            LiberePosition()
            return(False)
    else:
        liste=TriFusion(ListeCA[n])
        for k in liste:
            if TestePosition(k):
                return(True)
        LiberePosition()
        return(False)
```