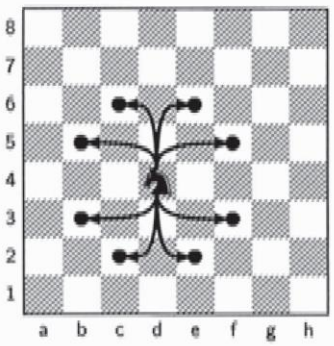
Le jeu d'échec se joue sur un échiquier, c'est à dire sur un plateau de 8×8 cases. Ces cases sont référencées de a1 à h8 (cf. figure).
Une pièce, appelée le cavalier, se déplace suivant un "L" imaginaire d'une longueur de deux cases et d'une largeur d'une case.
Exemple : un cavalier situé sur la case d4 atteint, en un seul déplacement, une des huit cases b5, c6, e6, f5, f3, e2, c2 et b3 (voir figure ci-contre).



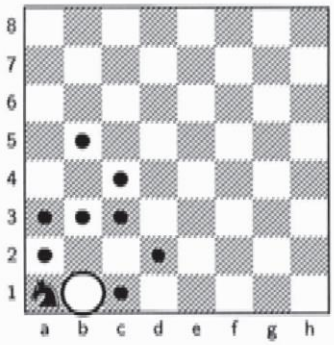
Dans toute la suite de l'exercice, on appellera **case permise** toute case que le cavalier peut atteindre en un déplacement à partir de sa position.

Le but de cet exercice est d'écrire un programme faisant parcourir l'ensemble de l'échiquier à un cavalier **en ne passant sur chaque case qu'une et une seule fois**.

Motivation et méthode retenue

Une première idée est de faire parcourir toutes les cases possibles à un cavalier en listant à chaque déplacement les cases parcourues. Lorsque celui-ci ne peut plus avancer on consulte le nombre de cases parcourues.

- Si ce nombre est égal à $64 = 8 \times 8$, alors le problème est résolu.
- Sinon, il faut revenir en arrière et tester d'autres chemins.



0. **Exemple :** on considère le parcours suivant d'un cavalier démarrant en a1 (figure ci contre) :

   a1, b3, c1, a2, c3, b5, a3, c4, d2.

   Avec ce début de parcours, au déplacement suivant :

   a. le cavalier va en b1. Peut-il accomplir sa mission ?

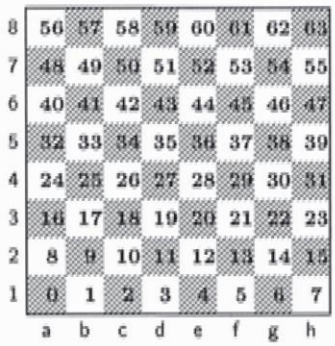
   b. le cavalier ne va pas en b1. Peut-il accomplir sa mission ?

Il convient donc dans la résolution du problème proposé d'éviter de se retrouver dans la situation repérée à la question **0.**

Dans tout ce qui suit, nous nommerons **coordonnées** d'une case la liste d'entiers $[i, j]$ où i représente le numéro de ligne et j le numéro de colonne (tous deux compris entre 0 et 7). Par exemple, la case b3 a pour **coordonnées** $[2, 1]$.

D'autre part, les cases sont numérotées de 0 à 63 en partant du coin gauche comme indiqué par la figure ci-contre.

Nous appellerons **indice** d'une case le numéro n compris entre 0 et 63 ainsi déterminé. Ainsi la case b3 a pour **indice** 17.



Les questions

1. Écrire une fonction `Indice` qui aux **coordonnées** $[i, j]$ d'une case renvoie son **indice**. Ainsi, `Indice` appliquée à $[2, 1]$ doit renvoyer 17.

2. Écrire une fonction `Coord` qui, à l'**indice** n d'une case renvoie la liste $[i, j]$ de ses **coordonnées**.
Ainsi `Coord` appliquée à 17 doit renvoyer $[2, 1]$.

3. On considère la fonction Python `CasA` suivante :

```
def CasA(n):
    Deplacements = [[1, -2], [2, -1], [2, 1], [1, 2], [-1, 2], [-2, 1], [-2, -1], [-1, -2]]
    L = []
    i, j = Coord(n)
    for d in Deplacements:
        u = i +d[0]
        v = j +d[1]
        if u >=0 and u <8 and v >=0 and v < 8:
            L.append(Indice([u, v]))
    return L
```

3.1 Que renvoient `CasA(0)` et `CasA(39)` ?

3.2 Expliquer **en une phrase** ce que fait cette fonction.

4. Écrire une fonction `Init` ne prenant aucun argument et qui modifie deux variables globales `ListeCA` et `ListeCoups`.
`ListeCoups` recevra la liste vide [].
`ListeCA` recevra une liste de 64 éléments. Chaque élément `ListeCA[n]` (pour $0 \leqslant n \leqslant 63$) devra contenir la liste des **indices** des cases qu'un cavalier peut atteindre en un coup à partir de la case d'indice n.

5. Après exécution de la fonction `Init()`, la commande `>>> ListeCA[0]` renvoie :

a - $[5]$
b - $[10, 17]$
c - $[10, 17, 0]$
d - $[17, 0, 10]$
e - []
f - Elle renvoie une autre valeur.

6. Au cours de la recherche, lorsqu'on déplace le cavalier vers la case d'indice n, cet indice n doit être retiré de la liste des **cases permises** à partir de la position n.

Exemple :
Après exécution de la fonction `Init()`, la liste des cases permises depuis b1 est [a3, c3, d2], et `ListeCA[1]` vaut $[16, 18, 11]$.
La liste des cases permises depuis a3 est [b5, c4, c2, b1] et `ListeCA[16]` vaut $[33, 26, 10, 1]$.
Puis on choisit de commencer le parcours en posant le cavalier en b1. Cette case doit donc être retirée de la liste des cases permises de a3, c3 et d2.
En particulier pour a3, la liste `ListeCA[16]` devient : $[33, 26, 10]$.

Cette méthode nous permet de détecter les blocages :
Le cavalier arrive sur la case d'indice n : n est alors retiré de toutes les listes `ListeCA[k]` pour toute case d'indice k permise pour n.
Si dès lors l'une de ces listes devient vide, nous dirons alors que nous sommes dans une **situation critique**, cela signifiera que la case d'indice k ne peut plus être atteinte que depuis la case d'indice n. Par conséquent :

- si le cavalier se déplace sur une autre case que celle d'indice k, alors cette dernière ne pourra plus jamais être atteinte ;
- si le cavalier se déplace sur la case d'indice k, le cavalier est bloqué pour le coup suivant. Et :
  - soit la mission est accomplie,
  - soit le cavalier n'a pas parcouru toutes les cases..

Le programme va réaliser la recherche en maintenant à jour la variable globale `ListeCoups` afin qu'elle contienne en permanence la liste des positions successives occupées par le cavalier au cours de ses tentatives de déplacement.
Nous avons alors besoin d'écrire trois fonctions :

6.1 Écrire une fonction `OccupePosition` qui :

- prend comme argument un entier n (indice d'une case), l'ajoute à la fin de la variable globale `ListeCoups`,
- puis enlève n de toutes les listes `ListeCA[k]` pour toutes les cases d'indice k permises depuis la case d'indice n,
- renvoie enfin la valeur `True`, si nous sommes dans une situation critique et `False` sinon.

On pourra utiliser la méthode `remove` *qui permet de retirer d'une liste le premier élément égal à l'argument fourni. Si l'argument ne fait pas partie de la liste, une erreur sera retournée.*

```
L = [1, 2, 3, 4, 2, 5]
L.remove(2)
L
```

Les commandes précédentes renvoient la liste $[1,3,4,2,5]$.

```
L = [1, 2, 3, 4, 2, 5]
L.remove(6)
```

Les commandes précédentes provoquent une erreur.

6.2 Écrire une fonction `LiberePosition` qui ne prend pas d'argument et qui

- récupère le dernier élément n de la variable globale `ListeCoups` (i.e. n est l'indice de la dernière case jouée à l'aide de la fonction `OccupePosition(n)`),
- puis l'enlève de `ListeCoups`,
- et enfin, qui ajoute n à toutes les listes `ListeCA[k]` pour toutes les cases d'indice k permises depuis la case d'indice n.

À la fin de cette fonction les listes `ListeCoups` et `ListeCA` seront donc dans le même état qu'avant l'exécution de la fonction `OccupePosition(n)`.

On pourra utiliser la méthode `pop` *qui renvoie le dernier élément d'une liste et le supprime de cette même liste.*

```
L = [1, 2, 3, 4, 2, 5, 2]
n = L.pop()
```

Les commandes précédentes affectent la valeur 2 à la variable n, la liste L étant ensuite : $[1,2,3,4,2,5]$.

6.3 Écrire une fonction `TestePosition` d'argument un entier n (indice d'une case) qui :

- occupe la position d'indice n.
- vérifie si la situation est critique.

  Si tel est le cas,
  - la fonction vérifiera si 63 cases sont occupées et dans ce cas renverra `True` pour indiquer que la recherche est terminée.
  - Si les 63 cases ne sont pas occupées, la fonction libérera la case d'indice n et renverra `False`.

  Dans le cas contraire,
  - la fonction vérifiera, à l'aide de la fonction `TestePosition(k)`, toutes les cases d'indice k jouables après la case d'indice n. On prendra garde à affecter une variable locale avec la liste `ListeCA[n]` puisque celle-ci risque d'être modifiée lors des appels suivants.
  - La fonction retournera `True` dès qu'un appel à `TestePosition(k)` retourne `True` ou libérera la case d'indice n et retournera `False` si tous les appels à `TestePosition(k)` retournent `False`.

7. Afin de réduire notablement la complexité temporelle du programme on part du principe qu'il faut tester en priorité les cases ayant le moins de cases permises possible. On appellera **valuation** d'une case d'indice n le nombre de cases permises pour cette case.

7.1 Écrire une fonction `valuation` qui prend comme argument un indice n de case en entrée et renvoie la valuation de cette case.

7.2 Écrire une fonction `Fusion` qui prend comme argument deux listes `A` et `B` constituées chacune d'entiers naturels entre 0 et 63 (`A` et `B` sont donc des listes d'indice de cases) ; on suppose que ces listes, `A` et `B` sont triées par ordre croissant de valuation de leurs éléments ; la fonction `Fusion(A,B)` retourne alors comme valeur la liste fusionnée de tous les éléments de `A` et `B` triée par ordre croissant de valuation de ses éléments.

7.3 Écrire une fonction `TriFusion` qui prend comme argument une liste `L` d'entiers compris entre 0 et 63, a priori non supposée triée par valuation croissante de ses éléments, et qui retourne comme valeur la liste de tous les éléments de `L` triée par valuation croissante de ses éléments.

7.4 Modifier la fonction `TestePosition` pour qu'elle agisse ainsi que l'on a décidé en début de question.