ACADÉMIE DE NORMANDIE
Liberté
Égalité
Fraternité

ACADÉMIE DE NICE
Liberté
Égalité
Fraternité

ACADÉMIE DE GRENOBLE
Liberté
Égalité
Fraternité

ACADÉMIE DE CORSE
Liberté
Égalité
Fraternité

aefe
Agence pour l'enseignement français à l'étranger

# Olympiades inter académique d'informatique 2025

## - Classe de seconde –

### Durée de l'épreuve : 2 heures

L'épreuve se déroule par équipe, de deux ou trois candidats, et sur ordinateur.

Le sujet est constitué de parties indépendantes qui peuvent être traitées dans l'ordre souhaité. Un dossier est mis à disposition des équipes avec les fichiers requis pour certaines questions.

Le dossier intitulé **SUJET** contenant les fichiers requis pour traiter les questions sera à mettre à disposition des équipes.

Les fichiers complétés par les candidats seront à joindre dans un unique dossier compressé et qui seront nommés de la façon suivante (sans espace ni accents) :

- NomEtablissement_VilleEtablissement_EquipeXX_Tour.svg
- NomEtablissement_VilleEtablissement_EquipeXX_Pion.svg
- NomEtablissement_VilleEtablissement_EquipeXX_Notre_piece.svg

- NomEtablissement_VilleEtablissement_EquipeXX_Dame_Roi.odt

- NomEtablissement_VilleEtablissement_EquipeXX_Parcours_cavalier.py

- NomEtablissement_VilleEtablissement_EquipeXX_equipe.odt

où le numéro de l'équipe sera attribué par vos soins.

Le dernier fichier (equipe.odt) recense les noms des membres de l'équipe.

Les calculatrices sont autorisées selon la réglementation en vigueur.

**Les démarches, même incomplètes seront prises en compte dans l'évaluation.**

Inria
inventeurs du monde numérique

# De toute pièce

Un fichier SVG (*Scalable Vector Graphics*) est un fichier qui décrit des dessins avec des formes géométriques (rectangles, cercles, lignes, etc.). Il permet de créer des images dites vectorielles qui ont l'avantage de pouvoir être agrandies autant que l'on souhaite sans perte d'information. C'est un format d'image également utilisé pour les images sur le web et qui peut être ouvert avec un navigateur web (pour visualiser), ou un éditeur de texte (pour éditer).

Dans cette première partie, nous allons créer la vue de face de deux pièces d'échiquier (une tour et un pion) au format SVG.

### La Tour

Ouvrir avec un éditeur de texte (Bloc-notes, Notepad ou autre) le fichier `Tour.svg`. Celui-ci contient la ligne de code suivante qui définit le tracé d'un rectangle :

```
<rect x="20" y="180" width="140" height="20" fill="gray"
stroke="black"/>
```

où :

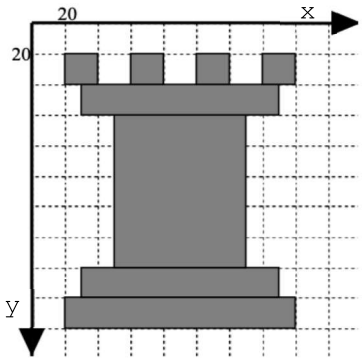
- les valeurs attribuées à `x` et `y` définissent la position du coin supérieur gauche du rectangle ;
- les valeurs attribuées à `width` et `height` définissent la largeur et la hauteur du rectangle ;
- `fill="gray"` définit la couleur de remplissage en gris ;
- `stroke="black"` définit la couleur du contour en noir.

Ouvrir le fichier `Tour.svg` avec un navigateur Web pour visualiser le rectangle obtenu.

### Tâche à accomplir

Compléter avec l'éditeur de textes les lignes du fichier `Tour.svg` entre les lignes de commentaires `<!-- Début -->` et `<!-- Fin -->` afin d'obtenir le dessin d'une tour d'échiquier.

On pourra s'aider du schéma suivant :



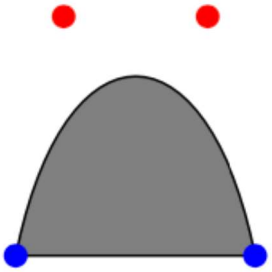

*Attention, pour le format SVG, l'axe des ordonnées est orienté vers le bas, et l'origine du repère est dans le coin supérieur gauche de l'image.*

**Le Pion**

Ouvrir avec un éditeur de textes le fichier `Exemple_courbe.svg`.
Celui-ci contient la ligne de code :

```
<path d="M 50 150 C 70 50, 130 50, 150 150 Z" stroke="black" fill="gray"/>
```

Cette ligne décrit une courbe (appelée *courbe de Bézier*) qui est tracée entre deux points (représentés ci-contre en bleu), dont la forme est influencée par un ou plusieurs points de contrôle (représentés ci-contre en rouge) qui « tirent » la courbe vers eux. Plus les points de contrôle sont éloignés, plus la courbe est tendue, ce qui permet de dessiner une grande variété de formes arrondies.



Détail de la ligne de code :

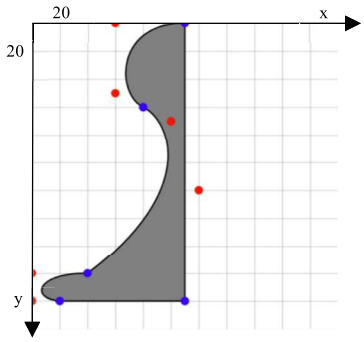
- `path` indique que l'on définit un chemin
- `M 50 150` : c'est le point de départ du path, M veut dire *moove* et 50 150 sont ses coordonnées (en bleu)
- `C 70 50, 130 50, 150 150` : c'est la courbe en noir sur le dessin
- `C` indique qu'il s'agit d'une courbe de Bézier
- `70 50` et `130 50` sont les coordonnées des points de contrôle de la courbe (en rouge)
- `150 150` sont les coordonnées du point d'arrivée (en bleu).
- `Z` veut dire que l'on revient au premier point du path (en ligne droite)
- `stroke="black"` : couleur du contour (noir).
- `fill="gray"` : couleur de remplissage (gris).

Remarques :

- une figure peut contenir plusieurs morceaux de courbe de Bézier qui se suivent ; il n'est pas nécessaire de répéter la lettre C à chaque fois dans ce cas.
- si l'on trouve la lettre L avant des coordonnées c'est qu'il s'agit d'un segment (L pour line).

**Tâche à accomplir**

Le fichier `Pion.svg` contient les lignes de codes d'une moitié de la représentation d'un pion, donnée ci-dessous.




Le modifier et le compléter à l'aide de l'éditeur de textes afin d'obtenir par symétrie l'image complète de ce pion.

**À vous de choisir !**

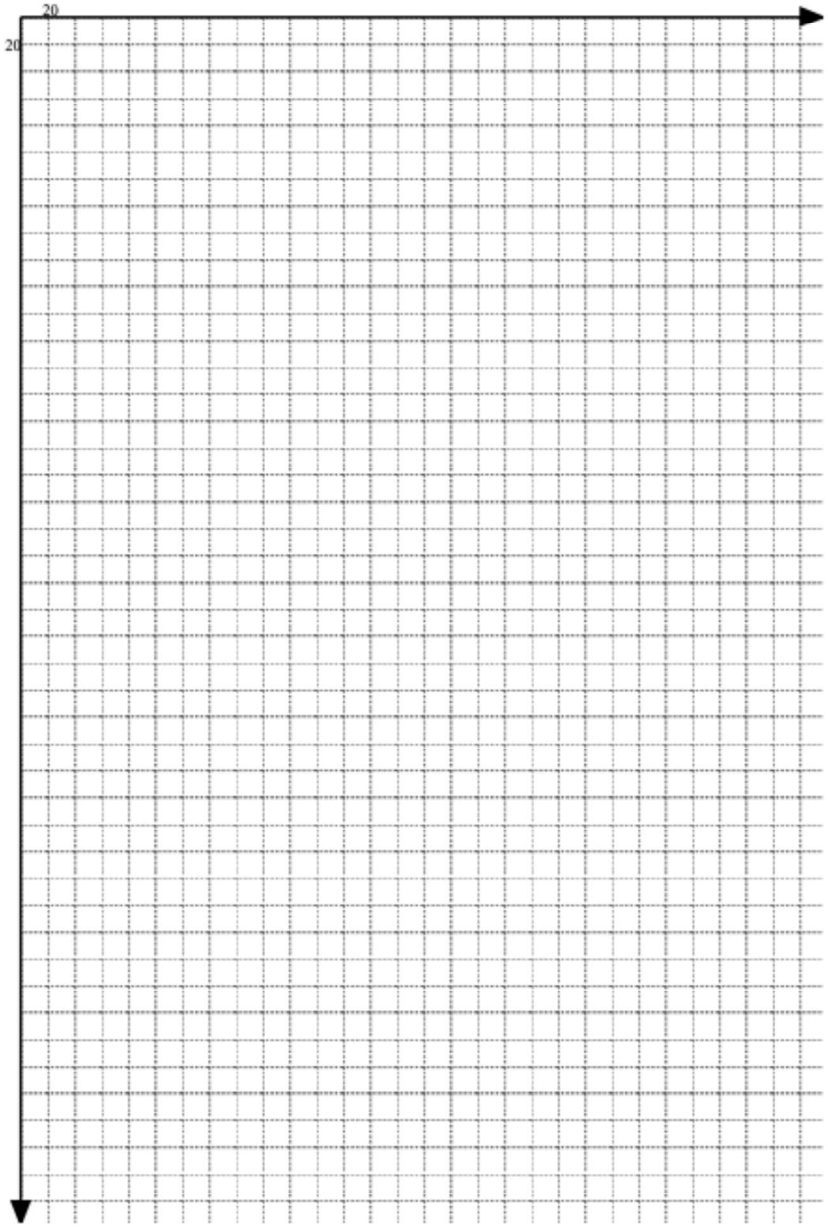
Créer en reprenant la structure de l'un des fichiers précédents pour créer un fichier `Notre_piece.svg` permettant de dessiner une pièce d'échiquier d'un autre type.

On pourra s'inspirer de l'une des images ci-dessous, ou laisser libre cours à son imagination et se servir du quadrillage donné pour élaborer un croquis.

Le roi

Le fou

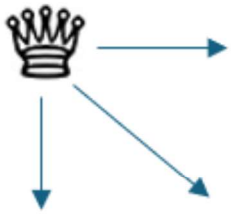
Le cavalier

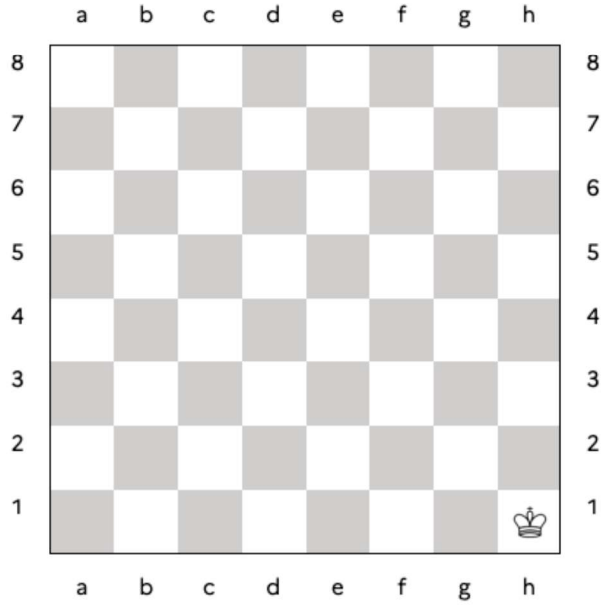


# Jeu de la Dame contre le Roi

C'est un jeu à deux joueurs sur un échiquier 8x8 dont l'objectif est de capturer le roi à l'aide d'une dame. Le roi est placé en *h*1 (comme illustré sur l'image ci-dessous) et reste immobile tout au long de la partie. La dame commence sur une case de la ligne 8, choisie librement par le joueur. Elle peut se déplacer d'une ou plusieurs cases, mais contrairement au jeu d'échecs classique, ses déplacements sont limités à trois directions :

- vers la droite,
- vers le bas,
- en diagonale vers le bas à droite.






Par exemple, si la dame débute en c8, elle peut se déplacer uniquement vers l'une des cases suivantes :

- sur la ligne 8 (*d*8, *e*8, *f*8, *g*8 ou *h*8),
- sur la colonne *c* (*c*7, *c*6, *c*5, *c*4, *c*3, *c*2 ou *c*1),
- sur la diagonale descendante vers la droite (*d*7, *e*6, *f*5, *g*4 ou *h*3).

Le jeu commence lorsque le premier joueur place la dame sur une case de la ligne 8. Ensuite, le deuxième joueur effectue le premier déplacement de la dame. Par la suite, les joueurs jouent à tour de rôle en déplaçant la dame en respectant les trois directions autorisées. Le premier joueur qui capture le roi, c'est-à-dire qui arrive en h1, remporte la partie.

***Les réponses aux questions suivantes seront à inscrire dans le fichier* Dame_Roi.odt.**

**Question 1 :** Effectuer plusieurs parties sur l'échiquier 8x8. Pour chaque partie, remplir le tableau du fichier fourni dans le fichier Dame_Roi.odt, selon le modèle à la page suivante, en notant :

- Les positions successives de la dame après chaque déplacement.
- Le résultat de la partie (victoire du joueur 1 ou du joueur 2).

Analyser ces parties et décrire vos observations. Y a-t-il une stratégie gagnante ?

Partie n° _____

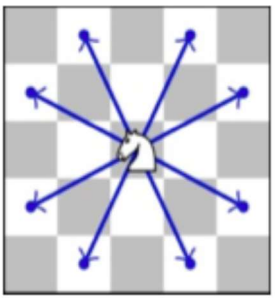
| N° de déplacement | La case où la dame a été placée/déplacée par chaque joueur | |
|---|---|---|
| | Joueur 1 | Joueur 2 |
| 1. | | |
| 2. | | |
| … | | |
| | | |
| Le résultat de la partie | | |

**Question 2 :** Pour gagner à coup sûr, est-il préférable de commencer ou de jouer en deuxième dans ce jeu en conservant les mêmes règles sur un échiquier 5×5 avec une dame qui démarre sur la ligne 5 avec le Roi en e1 ? Si vous pensez qu'il est avantageux de jouer en premier, indiquez la case de départ sur la ligne 5 où vous placeriez la dame et justifiez votre choix.

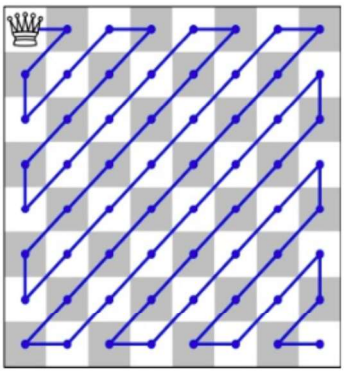
**Question 3 :** Comment garantir une victoire dans ce jeu sur un échiquier 8×8, quel que soit le jeu de l'adversaire ? Proposez une stratégie gagnante et justifiez-la. Dans votre réponse, précisez s'il est préférable de jouer en premier ou en deuxième. Si jouer en premier est avantageux, indiquez la case de départ optimale pour la dame. Expliquez ensuite le principe du déplacement de la dame selon votre stratégie afin d'assurer la victoire, quelle que soit la stratégie adoptée par l'adversaire.

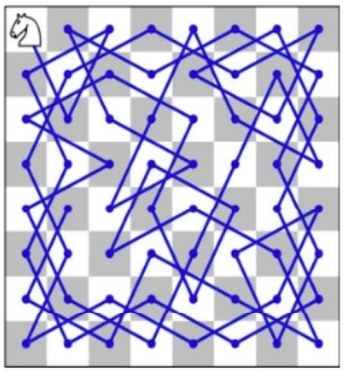
# Le parcours du Cavalier

Dans un jeu d'échecs, il y a plusieurs pièces ayant chacune sa façon de se déplacer. Le cavalier est la pièce qui a le déplacement le plus original : il peut se déplacer de 2 cases dans une direction (horizontale ou verticale) puis d'une case dans la direction perpendiculaire. Dans la figure ci-dessous, les cases avec des points représentent toutes celles où le Cavalier peut se déplacer, à partir de sa position :



Un parcours de l'échiquier consiste à trouver un chemin permettant de passer par toutes les cases d'un échiquier. La seule contrainte, c'est qu'on ne peut pas passer deux fois sur une même case pendant le parcours. Avec une Reine, un Roi ou une Tour, le problème est très simple. Voici deux exemples de parcours, l'un pour une Reine et le second pour un Cavalier :





Nous allons programmer en Python des fonctions permettant de chercher un tel parcours pour le Cavalier. En effet, lorsqu'on est dans un échiquier de taille 8×8, la tâche devient très complexe puisqu'il existe 19 591 828 170 979 904 ≈ $2^{16}$ parcours de Cavalier possibles. Même pour un ordinateur, cela fait trop de combinaisons pour chercher de tels parcours sans méthode particulière.

Pour pouvoir simplifier la recherche, on utilise des techniques, appelées **heuristiques**. Ces techniques permettent de simplifier la recherche, au risque de ne pas forcément trouver de solution.

La **règle de Warnsdorf** est probablement l'heuristique la plus célèbre pour le parcours du Cavalier. Elle sera décrite et programmée dans la dernière partie de l'exercice.

Le fichier `Parcours_cavalier.py` contient des fonctions déjà écrites et d'autres à compléter tout au long de cet exercice.

Des rappels et compléments sur le langage Python sont donnés en page 13 et 14.

**Notations**

Dans les questions suivantes, nous utiliserons les définitions suivantes :

- **voisin :** un voisin d'une case est une autre case que l'on peut atteindre en faisant un déplacement du cavalier. Par exemple les cases a1 et b3 sont des cases voisines.
- **case libre :** une case est libre si elle n'a pas encore été visitée dans le parcours actuel.
- **score :** le score d'une case est le nombre de ses voisins qui sont libres.

Les cases seront désignées par des textes correspondant aux notations classiques aux échecs : 'a1', 'c5' ou encore 'b3'.

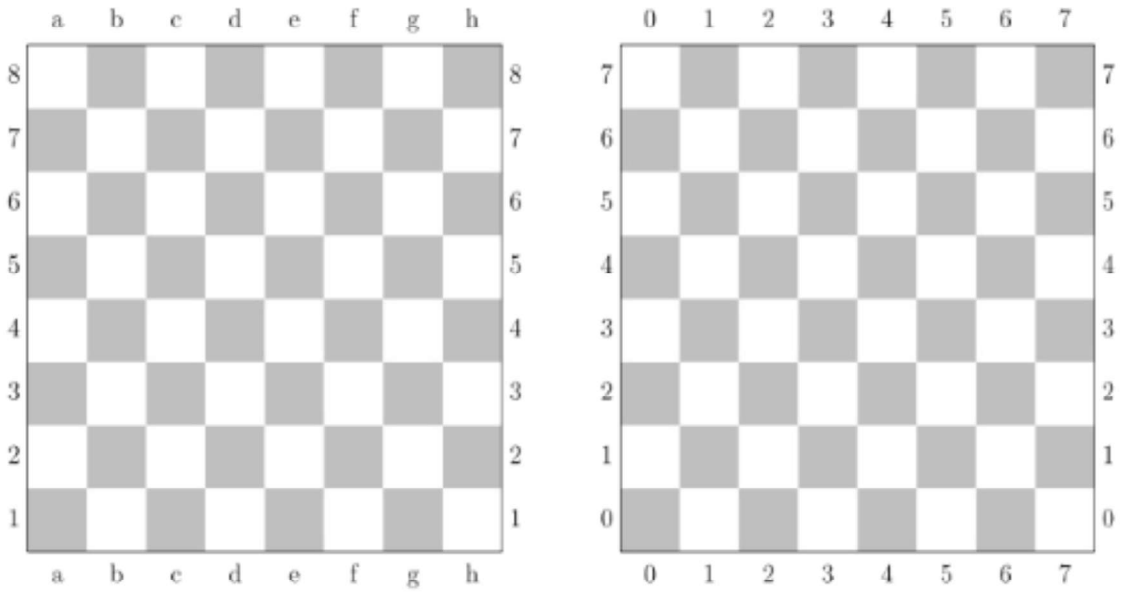
**Premières fonctions**

Tout d'abord, nous devons compléter la liste `déplacements` qui indique les déplacements qui sont possibles de faire avec un cavalier, sans tenir compte des bords de l'échiquier ou les cases déjà visitées.

Pour l'instant, dans le fichier, la liste contient cela :

```
deplacements = [(1, 2), (-1, 2), ...]
```

Afin de déterminer les coordonnées des cases sur lesquelles le cavalier peut aller, on utilise un système de coordonnées permettant de faire des calculs. Les coordonnées sont données sous forme de couple (colonne, ligne). La figure ci-dessous montre à gauche les notations classiques et à droite les coordonnées utilisées :




Dès lors, la case c1 a pour coordonnées (2, 0) et la case g3 a pour coordonnées (6, 2).

Ainsi les deux premiers couples de valeurs que contient la liste `deplacements` signifient qu'un cavalier peut se déplacer d'une colonne vers la droite (1) et de 2 lignes vers le haut (2), et qu'il peut se déplacer d'une colonne vers la gauche (-1) et de 2 lignes vers le haut (2).

**Question 1 :** Compléter dans le fichier la liste `deplacements` pour indiquer les 8 déplacements possibles par le cavalier.

Les fonctions prédéfinies `noms_case` et `coord_case` permettent respectivement d'obtenir le nom d'une case à partir de ses coordonnées et d'obtenir les coordonnées à partir du nom d'une case.

```
>>> nom_case((2, 0))
'c1'
>>> coord_case('g3')
(6, 2)
```

La fonction `dans_echiquier` permet de déterminer si une case se trouve dans l'échiquier ou pas. Plus précisément, elle vérifie si le numéro de ligne de la case se trouve bien dans l'échiquier et que le numéro de colonne aussi.

Pour cela, on utilise la fonction `taille` qui prend une grille et renvoie le nombre de colonnes et de lignes que contient cet échiquier. On peut également utiliser la fonction `echiquier` qui renvoie une grille correspondant à un échiquier dont le côté correspond à l'entier donné en paramètre. Par exemple, `echiquier(8)` correspond à un échiquier de taille 8x8.

On remarque que pour un échiquier 8x8, les coordonnées des lignes et des colonnes vont de 0 à 7. De même, dans un échiquier 5x5, les coordonnées des lignes et des colonnes vont de 0 à 4.

**Question 2 :** Compléter la fonction `dans_echiquier` qui prend en paramètres une grille et un nom de case et qui renvoie un booléen valant `True` si la case est dans la grille et `False` sinon.

On pourra tester cette fonction avec les exemples suivants :

```
>>> grille8 = echiquier(8)
>>> dans_echiquier(grille8, 'a6')
True
>>> dans_echiquier(grille8, 'a9')
False
>>> dans_echiquier(grille8, 'j3')
False
>>> dans_echiquier(grille8, 'h8')
True
>>> grille5 = echiquier(5)
>>> dans_echiquier(grille5, 'h8')
False
>>> dans_echiquier(grille5, 'c5')
True
```

Nous allons maintenant traiter les fonctions qui permettent de déterminer les prochaines cases sur lesquelles pourra aller le cavalier. Il faut donc déterminer tous les voisins de la case actuelle et ne garder que ceux qui sont libres.

Afin de distinguer les cases libres des cases déjà visitées dans la grille, nous allons mettre 0 dans les cases libres et mettre 1 lorsqu'elles sont visitées. Une case libre est donc une case dont la valeur est 0 dans la grille. La fonction `valeur`, qui prend en paramètres une grille et un nom de case, renvoie la valeur de cette case dans la grille.

```
>>> grille5 = echiquier(5)
>>> valeur(grille5, 'a1')
0
>>> valeur(grille5, 'c4')
0
```

Pour l'instant toutes les valeurs sont à 0. La fonction `modification` permet de changer la valeur d'une case dans la grille.

```
>>> grille5 = echiquier(5)
>>> modification(grille5, 'a1', 1)
>>> valeur(grille5, 'a1')
1
>>> valeur(grille5, 'b1')
0
```

Nous disposons aussi d'une fonction `voisins`, qui prend en paramètres une grille et un nom de case, qui renvoie la liste des noms des voisins de la case dans la grille, que ces cases soient libres ou non.

```
>>> grille5 = echiquier(5)
>>> voisins(grille5, 'a1')
['b3', 'c2'] # l'ordre ne sera pas forcément le même
>>> voisins(grille5, 'c3')
['d5', 'b5', 'd1', 'b1', 'e4', 'e2', 'a4', 'a2'] # l'ordre ne sera
pas forcément le même
```

Attention, en fonction de l'ordre dans lequel vous avez défini les déplacements dans la question 1, les résultats ne sont pas forcément dans le même ordre, mais il faut trouver exactement les mêmes cases. Si ce n'est pas le cas, il faut vérifier les valeurs de la liste `déplacements`.

La fonction `voisins_libres`, qui prend en paramètres une grille et un nom de case, renvoie la liste des voisins **libres** de la case dans la grille. Pour cela, on parcourt chaque voisin de la case et, s'il est libre (c'est-à-dire que sa valeur est à 0 dans la grille), on rajoute ce voisin dans la liste `liste_voisins`.

**Question 3 :** Compléter le code de la fonction `voisins_libres` dans le fichier.

```
>>> grille5 = echiquier(5)
>>> voisins_libres(grille5, 'b5')
['c3', 'a3', 'd4']
>>> modification(grille5, 'a3', 1)
>>> modification(grille5, 'd4', 1)
>>> voisins_libres(grille5, 'b5')
['c3']
```

Avec les fonctions que nous avons déjà vues, nous pouvons chercher manuellement un parcours. Pour cela, il faut utiliser la fonction `interactif` qui prend en paramètres une grille et un nom de case, et qui permet de choisir étape par étape la prochaine

case à visiter. Cette fonction utilise une autre fonction, `affichage`, qui affiche la grille. Dans cette grille, la position actuelle est représentée par un `C`. Les voisins sont notés `V` et les cases visitées `O`.

**La règle de Warnsdorf**

Nous allons maintenant implémenter la règle de Warnsdorf, qui consiste à choisir systématiquement le voisin avec le plus faible score. En cas d'égalité, on en déterminera un de façon arbitraire parmi ceux qui ont le score minimal. Pour cela, on peut soit compter le nombre de cases renvoyées par la fonction `voisins_libres`, soit compter les cases renvoyées par la fonction `voisins` qui sont libres.

**Question 4 :** Compléter le code de la fonction `score` qui prend en paramètres une grille et un nom de case et qui renvoie le score de la case dans la grille.

```
>>> grille5 = echiquier(5)
>>> voisins_libres(grille5, 'b5')
['c3', 'a3', 'd4']
>>> score(grille5, 'b5')
3
>>> modification(grille5, 'c3', 1) # on marque comme visité un des
voisins
>>> score(grille5, 'b5')
2
>>> modification(grille5, 'a3', 1) # on marque un autre voisin
>>> score(grille5, 'b5')
1
```

Une fois la fonction `score` complétée, le score des voisins s'affichera dans la fonction `interactif`, afin de pouvoir appliquer la règle de Warnsdorf à la main.

Afin de permettre à l'ordinateur d'appliquer la règle automatiquement, il faut déterminer la liste des voisins libres triée par score. Ainsi, la première case de la liste aura un score minimal, même si ce n'est pas la seule.

Pour cela, nous avons déjà défini la fonction `tri` qui prend en paramètres une grille et une liste de noms de cases, et qui trie les cases dans l'ordre croissant de score. Cette fonction ne renvoie pas une nouvelle liste, mais modifie la liste donnée en paramètre.

```
>>> grille5 = echiquier(5)
>>> liste_voisins = voisins_libres(grille5, 'b5')
>>> liste_voisins
['c3', 'a3', 'd4']
>>> score(grille5, 'c3')
8
>>> score(grille5, 'a3')
4
>>> score(grille5, 'd4')
4
>>> tri(grille5, liste_voisins)
>>> liste_voisins
['a3', 'd4', 'c3']
>>> modification(grille5, 'b3', 1) # On enlève un voisin libre à d4
>>> tri(grille5, liste_voisins)
>>> liste_voisins
['d4', 'a3', 'c3']
```

**Question 5 :** Compléter le code de la fonction `voisins_tries` qui prend en paramètres une grille et un nom de case, et qui renvoie la liste des voisins libres, triée dans l'ordre croissant des scores des voisins.

```
>>> grille5 = echiquier(5)
>>> voisins_tries(grille5, 'b5')
['a3', 'd4', 'c3']
>>> modification(grille5, 'b3', 1)
>>> voisins_tries(grille5, 'b5')
['d4', 'a3', 'c3']
```

Nous avons maintenant toutes les fonctions nécessaires pour appliquer automatiquement la règle de Warnsdorf. La fonction `warnsdorf` prend en paramètres une grille et le nom de la case de départ, et cherche le plus long chemin possible en appliquant systématiquement cette règle. Pour simplifier, on prendra systématiquement le premier voisin libre dans la liste triée par score. À la fin, la fonction renvoie la liste des noms de cases du parcours obtenu. Ce parcours peut ne pas passer par toutes les cases.

**Question 6 :** Compléter le code de la fonction `warnsdorf` pour qu'elle applique la règle et renvoie le parcours obtenu. Il y a des indications dans le code de la fonction pour expliciter ce qui est attendu à chaque ligne à compléter.

Pour vous aider à tester votre fonction, vous pouvez aussi utiliser la fonction `applique_chemin` qui prend en paramètres une grille et une liste de noms de cases, et qui modifie la grille en mettant à 1 la valeur des cases faisant partie de la liste. Si la liste ne correspond pas à des déplacements valides du cavalier, la fonction s'arrête au premier mouvement invalide et renvoie `False`. Si le chemin est valide, la fonction renvoie `True`. Cette fonction vous permettra de donner le début d'un chemin et ensuite d'appliquer la règle de Warnsdorf.

**Question 7 :** Affecter à la variable `solution26` le chemin trouvé par la fonction `warnsdorf` en partant de a1 dans une grille 26x26.

On doit obtenir :

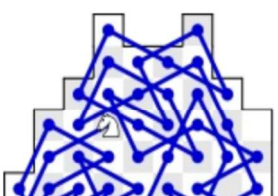
```
>>>  solution26 = warnsdorf(echiquier(26), 'a1')
>>>  len(solution26) # Il doit y avoir 26*26=676 cases 676
>>>  applique_chemin(echiquier(26), solution26) # On vérifie que le
chemin est valide True
```

**D'autres types de grilles**

Afin de chercher des parcours dans des échiquiers qui ne soient pas carrés, vous pouvez utiliser la fonction `importation` qui prend en paramètre une liste de texte et qui renvoie la grille correspondante. Chacun des textes de la liste correspond à une ligne de la grille. Les espaces correspondent à des cases vides et les autres symboles à des cases bloquées. Dans la grille, la valeur des cases bloquées est mise à 2.

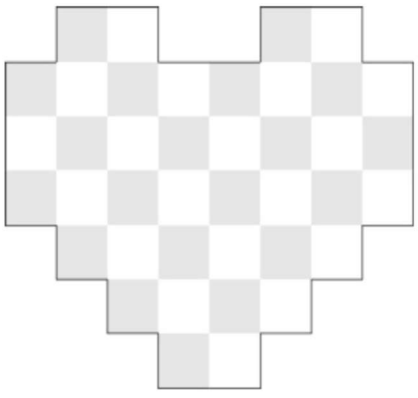
Par exemple, la grille suivante en forme de cheval (avec un peu d'imagination) peut être obtenue ainsi :





```
>>> cheval = ['XXX XX XXX',
              'XXX    XXX',
              'XX      XX',
              'X        X',
              'X        X',
              '         X',
              '         X',
              '    X    X',
              'XXXXX    X',
              'XXXX     X',
              'XXXX     X',
              'XXX      X',
              'XXX      X',
              'XXX      X',
              'XX        ',
              'XX        ']
>>> grille_cheval = importation(cheval)
>>> affichage(grille_cheval)
   abcdefghij
  +----------+
16|XXX.XX.XXX|16
15|XXX....XXX|15
14|XX......XX|14
13|X........X|13
12|X........X|12
11|.........X|11
10|.........X|10
 9|....X....X|9
 8|XXXXX....X|8
 7|XXXX.....X|7
 6|XXXX.....X|6
 5|XXX......X|5
 4|XXX......X|4
 3|XXX......X|3
 2|XX........|2
 1|XX........|1
  +----------+
   abcdefghij
```

**Question 8 :** Affecter à la variable `coeur` à la fin de votre programme une liste de chaînes de caractères permettant d'obtenir la grille en forme de cœur ci-dessous :



Vous pourrez utiliser votre code ainsi :

```
>>> solution_coeur =
warnsdorf(importation(coeur), 'b7')
>>> len(solution_coeur) # Il y a 40 cases
à parcourir 40
>>> applique_chemin(importation(coeur),
solution_coeur)
True
```

## Rappels et compléments sur les listes en langage Python

Une liste en Python est un objet permettant de regrouper plusieurs valeurs. Elle se note à l'aide de crochets, avec les éléments séparés par des virgules :

```
>>> une_liste = [6, 3, 2]
>>> une_liste_vide = []
>>> animaux = ['chien', 'chat', 'poisson', 'oiseau']
```

La liste `une_liste` contient les nombres 6, 3 et 2. La liste `une_liste_vide` ne contient rien. On dit que c'est la **liste vide**. Enfin, la liste `animaux` contient les noms de 4 types d'animaux. En effet, les listes peuvent contenir de nombreux types d'objets, comme des nombres (entiers ou décimaux) ou des textes.

Afin d'accéder à une valeur contenue dans une liste, on utilise son **indice**. Les indices commencent à 0, pour la valeur au début de la liste (à gauche). Ainsi, dans la liste animaux, l'indice de `'chien'` est 0, celui de `'chat'` est 1, celui de `'poisson'` est 2 et celui de `'oiseau'` est 3. Plus concrètement, on obtient la valeur à l'aide d'un indice avec la notation `Nom_de_liste[INDICE]` :

```
>>> animaux[0]
'chien'
>>> animaux[1]
'chat'
>>> animaux[2]
'poisson'
>>> animaux[3]
'oiseau'
```

On peut également accéder au dernier élément d'une liste à l'aide de l'indice -1 :

```
>>> animaux[-1]
'oiseau'
>>> une_liste[-1]
2
```

On peut ajouter une valeur à la fin de la liste par `Nom_de_liste.append(valeur)` :

```
>>> animaux.append('cheval') # ajout d'un élément
>>> animaux
['chien', 'chat', 'poisson', 'oiseau', 'cheval']
>>> une_liste.append(15)
>>> une_liste
[6, 3, 2, 15]
```

Pour connaître le nombre d'éléments dans une liste, il y a la fonction `len` :

```
>>> len([])
0
>>> len([4, 2, 1])
3
>>> jours = ['a1', 'b3', 'c5', 'a4']
>>> len(jours)
4
```

Les listes peuvent se parcourir de deux façons avec une boucle `for` :

```
>>> for val in animaux: # parcours par valeur
...     print(val)

chien
chat
poisson
oiseau
cheval
>>> for i in range(len(animaux)): # parcours par indice
...     print(i, animaux[i])

0 chien
1 chat
2 poisson
3 oiseau
4 cheval
```

Dans le premier cas, la variable `val` reçoit successivement les valeurs de la liste. Dans le deuxième cas, la variable `i` reçoit successivement les indices de la liste et `Nom_de_liste[i]` est la valeur se trouvant à cet indice.