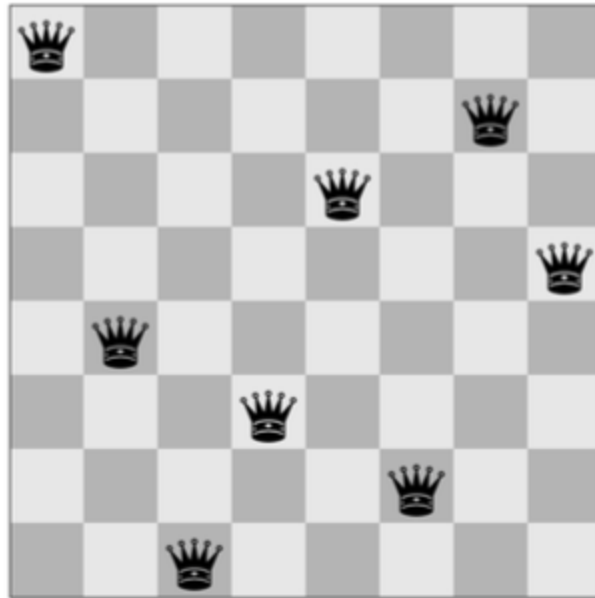


Langage Python A3 TP3\_2 (ce TP se déroule sur 1h30, la séance TD3\_1 est à faire sur schooding)

## Le problème des 8 reines



Le but est de réussir à placer 8 reines sur un échiquier de 8 cases sur 8 cases de telle façon à ce qu'aucune reine ne puisse en attaquer directement une autre. Une reine est attaquée si sur sa ligne, sa colonne ou ses diagonales une autre reine est présente.

### Classe individu:

Pour chacune des huit colonnes, on doit reporter la ligne où se trouve la reine. La configuration de reines de l'échiquier ci-dessus s'écrit alors: [0, 4, 7, 5, 2, 6, 1, 3]

**Définir la classe individu**, ayant comme attributs : une liste de 8 valeurs (ou plus précisément `echec_dim` valeurs selon la dimension de l'échiquier) et un entier `nbconflict` représentant le nombre de conflit associé à cet individu (ce n'est pas un attribut à affecter par l'utilisateur au moment de l'instanciation mais à calculer selon la fonction `fitness` à définir ultérieurement).

*Astuce, constructeur avec une liste en paramètre, sans ce paramètre ca génère aléatoirement une liste*

```

9 import random
10 echec_dim=8
11
12 class individu:
13     def __init__(self, val=None):
14         if val==None:
15             self.val=random.sample(.....)
16         else:
17             self.val=val
18         #self.nbconflict=self.fitness()#à définir ultérieurement
19

```

Dans la classe individu, **surcharger une fonction spéciale** de façon à pouvoir écrire `print(ind)` #ind est une instance de individu et qui imprime l'individu. Exemple : 0 4 7 5 2 6 1 3

**Définir la méthode conflict** qui retourne true si la reine à la position p1 est en conflit avec la reine en position p2

```

20     def conflict(p1,p2):
21         """ retourne true si la reine à la position p1 est
22         en conflit avec la reine en position p2"""
23         return .....|

```

p1 et p2 deux séquences à deux éléments (position sur l'échiquier), exemple p1=[0,1] p2=[1,2]  
`individu.conflict(p1,p2)` doit retourner True.

**Définir la méthode fitness** qui permet de retourner le nombre de conflit d'un individu

```

26     def fitness(self):
27         """ evaluer l'individu c'est connaitre le nombre de conflit"""
28         self.nbconflict=0
29         for i in .....:
30             for j in .....:
31                 if(individu.conflict ([i,.....],[j,.....])):
32                     self.nbconflict=self.nbconflict+1
33         return self.nbconflict

```

## Population

Définir une méthode `create_rand_pop(count)` qui génère une liste de "count" individus.

Attention : il faut définir cette méthode hors la classe. il ne s'agit pas ni d'une méthode de classe ni d'instance.

## Evaluation

**Définir une méthode evaluate(pop)** qui évalue la population, en gros retourne une liste des individus triés selon le nombre de conflit de chacun.

*Astuce: sorted, lambda (faites votre recherche)*

## Selection

Définir une méthode `selection(pop, hcount, lcount)` qui retourne une sous population avec les “hcount” premiers éléments et les “lcount” derniers éléments de la liste `pop`.

## Croisement

Définir une méthode `croisement(ind1,ind2)` qui retourne une liste de deux individus à partir de deux individus `ind1` et `ind2` (4 premières données de `ind1` suivies des 4 dernières de `ind2` puis 4 premières données de `ind2` suivies des 4 dernières de `ind1`)

## Mutation

Définir une méthode `mutation(ind)` qui retourne un individu suite à la mutation de `ind`. Il s’agit de prendre un indice aléatoire de l’individu et de remplacer la donnée correspondante par une nouvelle valeur aléatoire (entre 0 et 7).

Attention, il faut que ça soit un nouvel individu.

## Boucle finale

Maintenant que vous avez défini la création d’une population, l’évaluation, la sélection, le croisement et la mutation, il faut mettre en place la boucle permettant de trouver une solution à notre problème.

```
56 def algoLoopSimple():
57     pop=create_rand_pop(25) #je commence par créer une population aléatoire de 25 individus
58     solutiontrouvee=False
59     nbriteration=0
60     while not solutiontrouvee: #j'entre dans une boucle jusqu'à ce que je tombe sur une solution
61         print("iteration numéro : ", nbriteration)
62         nbriteration+=1
63         evaluation=evaluate(pop) #j'évalue la population, le retour est une liste triée selon le nbre de conflit
64         if evaluation[0].fitness()==0: # c'est à dire j'ai une solution
65             solutiontrouvee=True
66         else: #j'ai pas de solution
67             select=selection(evaluation,10,4) # je sélectionne les 10 meilleurs et les 4 pire
68             croises=[]
69             for i in range (0,len(select),2 ): # je fais le croisement deux par deux
70                 croises+=croisement(select[i],select[i+1])
71             mutes=[]
72             for i in select: #j'opère la mutation sur chacun des sélectionnés
73                 mutes.append(mutation(i))
74             newalea=create_rand_pop(5) # j'ajoute 5 nouveaux individus aléatoires
75             pop=select[:]+croises[:]+mutes[:]+newalea[:] #je recrée ma population :la selection,la mutation,le croisement et les nouveaux
76     print(evaluation[0])
```

En gros, on crée une population aléatoire d’un certain nombre d’individus, on l’évalue, si on tombe sur un individu à fitness nulle, tant mieux c’est la solution, sinon on sélectionne 10 meilleurs et 4 pires, on croise la sélection deux par deux, on mute chaque individu de la sélection

et on crée 5 nouveaux individus, on regroupe le tout pour former la nouvelle population et la boucle est bouclée.

## Discussion et échange:

À chaque fois que vous lancez `algoloopSimple`, est ce que c'est le même nombre d'itération est nécessaire pour trouver une solution?

Si on veut appliquer un algorithme de force brute, quel est l'espace de recherche?

Pourquoi ne nous contentons pas de sélectionner les meilleurs? À quoi sert `newalea`?

## Optionnel : Boucle finale, Toutes les solutions possible

Réécrire le code précédent de façon à récupérer toutes les solutions possibles, commencer par initialiser une liste vide `allsolutions`, chaque fois que vous tombez sur une solution vous l'ajoutez à la cette liste (si elle n'existe pas déjà) et vous la supprimer de la liste `evaluation`.

Faites une boucle infinie et imprimez à chaque itération le nombre de solutions atteintes.