

LISTES ET STRUCTURES MULTIDIMENSIONNELLES

Lundi 10 novembre

Option Informatique
Ecole Alsacienne

PLAN

1. Les listes en Python
2. Structures multidimensionnelles
3. Les grilles en Python
4. Les huit dames
5. Labyrinthes
6. Mini-TD

LES LISTES EN PYTHON

LE TYPE `LIST`

- En Python, le type `list` recouvre deux notions vues l'année dernière
 - Les vecteurs (`array` en Caml)
 - Les listes (`list` en Caml)
- Selon les circonstances, on manipulera les objets de type `list` plutôt comme des vecteurs, ou plutôt comme des listes.
- Plusieurs fonctions sont ainsi à votre disposition pour ces deux usages.

LA LISTE VIDE

- Une liste vide est une liste qui ne contient aucun élément
- Deux syntaxes sont possibles pour déclarer une liste vide
 - `nom = []`
 - `nom = list()`

- Exemple

```
l0 = []  
print(l0)
```

```
[]
```

LONGUEUR D'UNE LISTE

- La longueur d'une liste est le nombre d'éléments contenus dans cette liste
- Pour obtenir cette longueur, on utilise la fonction `len`

- Exemple :

```
l1 = [2, 4, 5]
longueur = len(l1)
print(longueur)
```

3

- Remarque : La liste vide a pour longueur 0

AJOUTER UN ÉLÉMENT DANS UNE LISTE

- La fonction `append` permet d'ajouter un élément à la fin d'une liste

- Exemple :

```
l2 = [2, 4, 5]
```

```
l2.append(3)
```

```
print(l2)
```

```
[2, 4, 5, 3]
```

- Remarque 1 : Le nouvel élément est ajouté à la fin et non en tête de liste
- Remarque 2 : La liste est modifiée sur place (on ne crée pas une nouvelle liste avec un élément en plus)
 - En effet, on utilise la "méthode" `append` de l'objet `l2`
 - Ce sera le cas de la majorité des fonctions sur les listes

CONCATÉNATION DE LISTES

- La fonction `extend` permet de concaténer une liste à une autre

- Exemple :

```
l3 = [2, 4, 5]
```

```
l4 = [0, 7]
```

```
l3.extend(l4)
```

```
print(l3)
```

```
[2, 4, 5, 0, 7]
```

RETIRER ET RENVOYER LE DERNIER ÉLÉMENT D'UNE LISTE

- La fonction `pop` permet de
 - retirer le dernier élément d'une liste
 - renvoyer cet élément

- Exemple :

```
l5 = [2, 4, 5]
```

```
x = l5.pop()
```

```
print(l5)
```

```
print(x)
```

```
[2, 4]
```

```
5
```

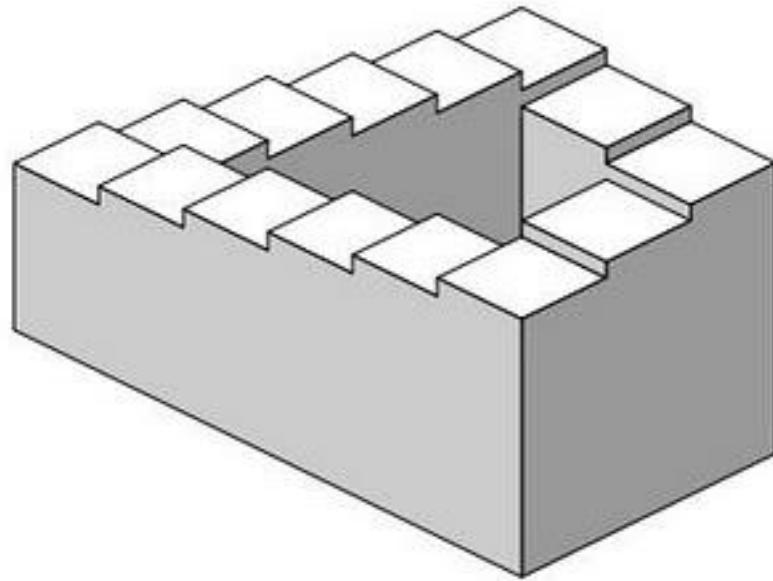
FONCTION AVANCÉE SUR LES LISTES

- Python met à votre disposition un certain nombre de fonctions plus avancées sur les listes
 - `in` : détermine si un élément appartient à une liste
 - `index` : détermine la position de la première occurrence d'un élément dans une liste
 - `count` : détermine le nombre d'occurrence d'un élément dans une liste
 - `remove` : retire la première occurrence d'un élément dans une liste
 - `insert` : insère un élément à une position donnée dans une liste
 - `reverse` : "renverse" une liste
 - `sort` : trie une liste

STRUCTURES MULTIDIMENSIONNELLES

POUR COMMENCER...

Qu'est-ce que c'est que cette histoire de multidimension ?



DIMENSION 1

Une **structure à une dimension**, c'est tout simplement :

- Une droite
- Un tableau
- Informellement, un truc "rectiligne"

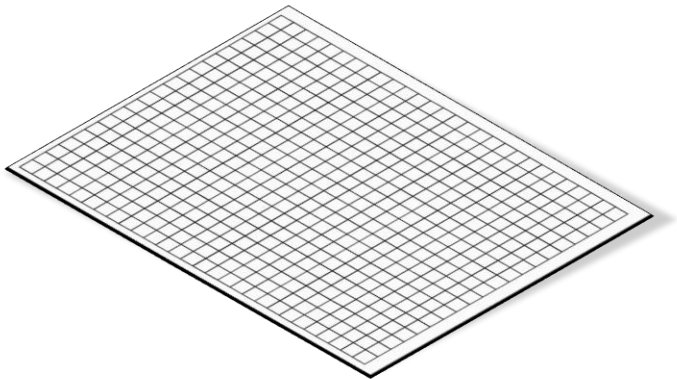


0	1	2	3	4	5	6	7	8	9
51	23	78	64	89	53	90	12	3	75

DIMENSION 2

Une **structure à deux dimension**, c'est par exemple :

- Un plan
- Un tableau à double entrée

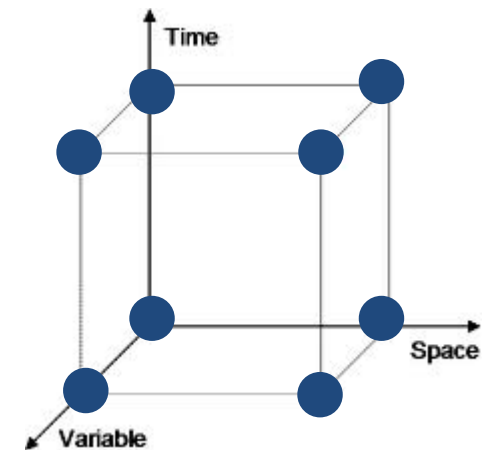


	USA	Chine	Russie
Prénom	Barack	Xi	Vladimir
Nom	Obama	Jinping	Poutine

DIMENSION 3

Une **structure à trois dimensions**, c'est par exemple :

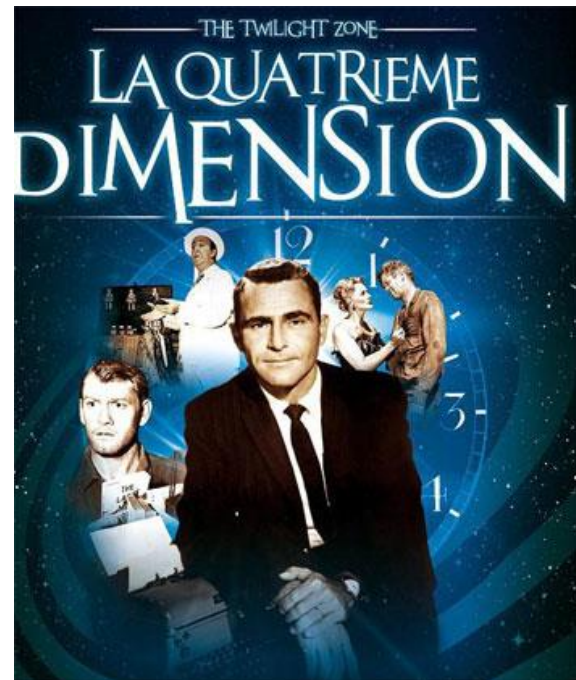
- Un objet en relief
 - N'importe quel objet en relief
 - Enfin presque...
- Un tableau à 3 entrées



DIMENSION 4 (ET PLUS)

Au-delà de la dimension 3, les choses se compliquent...

- La représentation mentale et physique est ardue
- Mais on pourra créer et manipuler de tels objets virtuels



DIMENSION 0

Une structure de dimension 0, c'est :

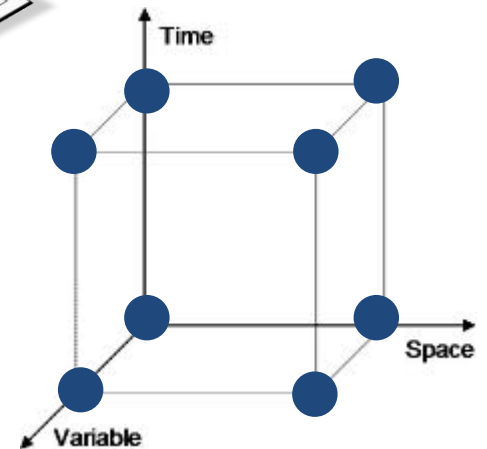
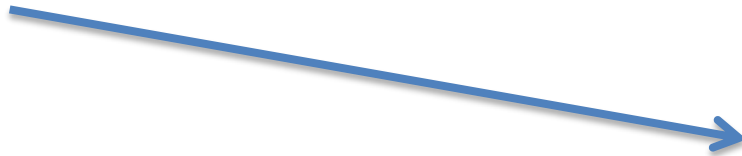
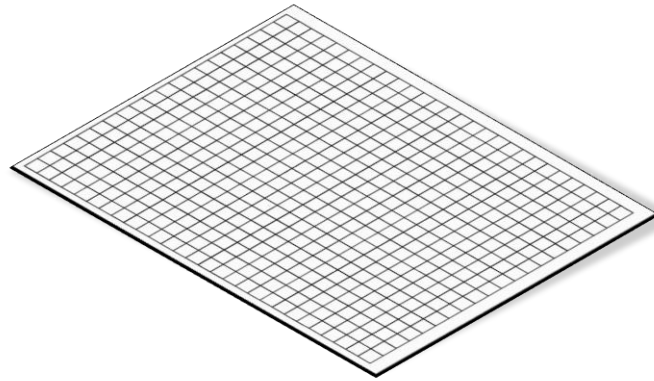
- Un point
- Une variable simple (ex : `int`, `float`, `bool`)



FORMELLEMENT

La **dimension** d'un objet mathématique est

- Le nombre de **degrés de liberté** qu'on a avec cet objet
- Le nombre de **paramètres à fixer** pour se situer sur cet objet



COMMENT PASSER EN DIMENSIONS SUPÉRIEURES ?

- *Question : Comment utiliser les structures de données que nous avons déjà vues pour obtenir des structures à plusieurs dimensions ?*
- Réponse : en les **combinant** !
 - Tableau de tableaux
 - Tableau de tableaux de tableaux
 - Etc.

EXEMPLE 1 : LE DAMIER

- *Question : Quelle structure vous semblerait adaptée pour représenter l'objet suivant ?*



Tableau de tableaux d'entiers

EXEMPLE 2 : LE RÉPERTOIRE ALPHABÉTIQUE

- Question : *Quelle structure vous semblerait adaptée pour représenter l'objet suivant ?*



Tableau de tableaux de chaînes de caractères

EXEMPLE 3 : LE PUISSANCE 4 3D

- *Question : Quelle structure vous semblerait adaptée pour représenter l'objet suivant ?*

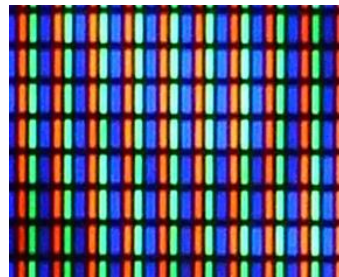


Tableau de tableaux de tableaux d'entiers

LES GRILLES EN PYTHON

LA GRILLE

- Un tableau à deux dimensions est souvent appelé une **grille**
- En général, une grille est de taille fixe
- Cette structure permet de représenter de nombreuses situations
 - Damier ou échiquier
 - Pixels sur un écran
 - Interactions
 - Etc.



COMMENT CRÉER UNE GRILLE EN PYTHON ?

- Pour créer une grille en Python, on peut utiliser la fonction suivante :

```
def creer_grille(largeur, hauteur, val_init) :  
    grille = [[]] * hauteur  
    for ligne in range(hauteur) :  
        grille[ligne] = [val_init] * largeur  
    return grille
```

- Exemple :

```
g0 = creer_grille(4, 2, 0)  
print(g0)
```

```
[[0, 0, 0, 0], [0, 0, 0, 0]]
```

UN TABLEAU DE LIGNES

- Une grille est un tableau de tableaux :

	0	1	2	3	4
0	14	4	72	31	38
1	25	38	44	1	58
2	94	86	51	13	37
3	7	39	3	14	2

- Si `t` représente tout le tableau (de type `list list`)
 - `t[3][1]` représente une case contenant 39 (de type `int`)
 - `t[1]` représente la deuxième ligne du tableau (de type `list`)
 - `t` est en quelque sort un « tableau de ligne »

LARGEUR ET HAUTEUR D'UN TABLEAU

- **Question** : *Comment trouver la largeur et la hauteur d'une grille (représenté par un tableau de tableaux) ?*

```
def hauteur t :  
    len(t)
```

```
def largeur t :  
    len(t[0])
```

	0	1	2	3	4
0	14	4	72	31	38
1	25	38	44	1	58
2	94	86	51	13	37
3	7	39	3	14	2

ATTENTION AUX RÉFÉRENCES

- Proposition
 - Pour créer un tableau, la syntaxe est `t = [valeur] * taille`
 - Pour créer une grille, on pourrait donc écrire
`g = [[valeur] * dimension1] * dimension2`

- Exemple

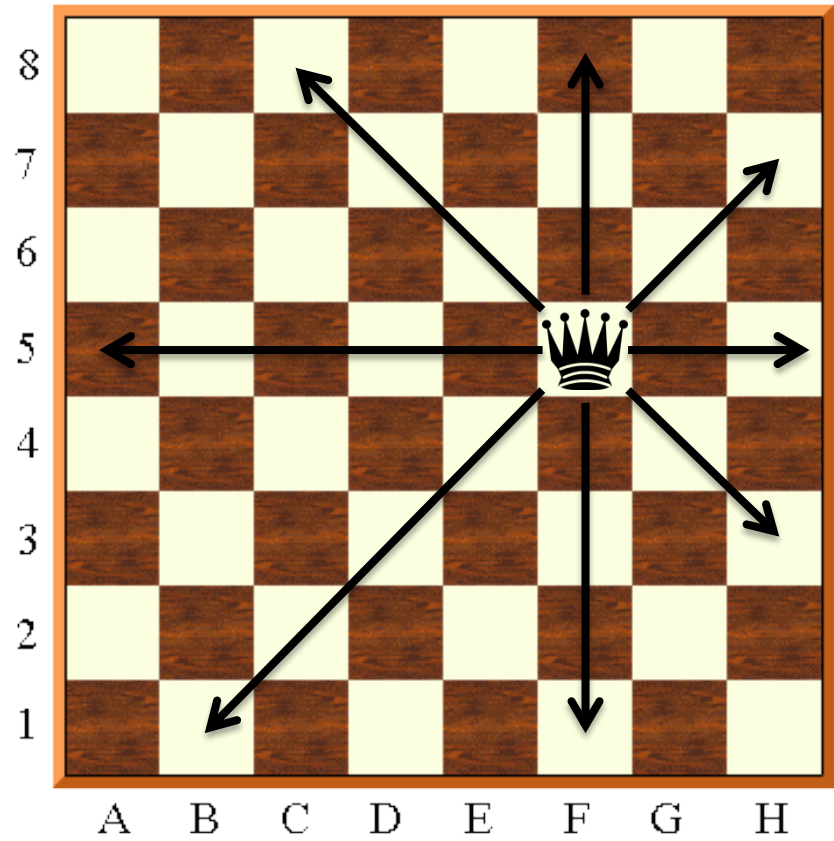
```
g0 = ([[0] * 2]) * 3
print(g0)
g0[0][0] = 1
print(g0)
```

```
[[0, 0], [0, 0], [0, 0]]
[[1, 0], [1, 0], [1, 0]]
```

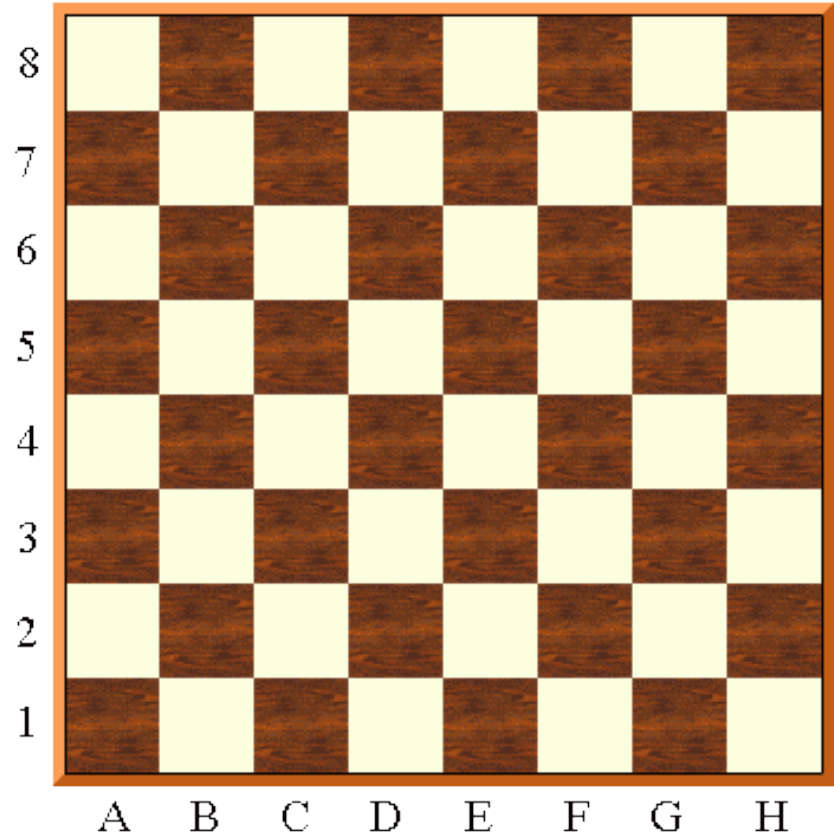
- Attention aux références !

LES HUIT DAMES

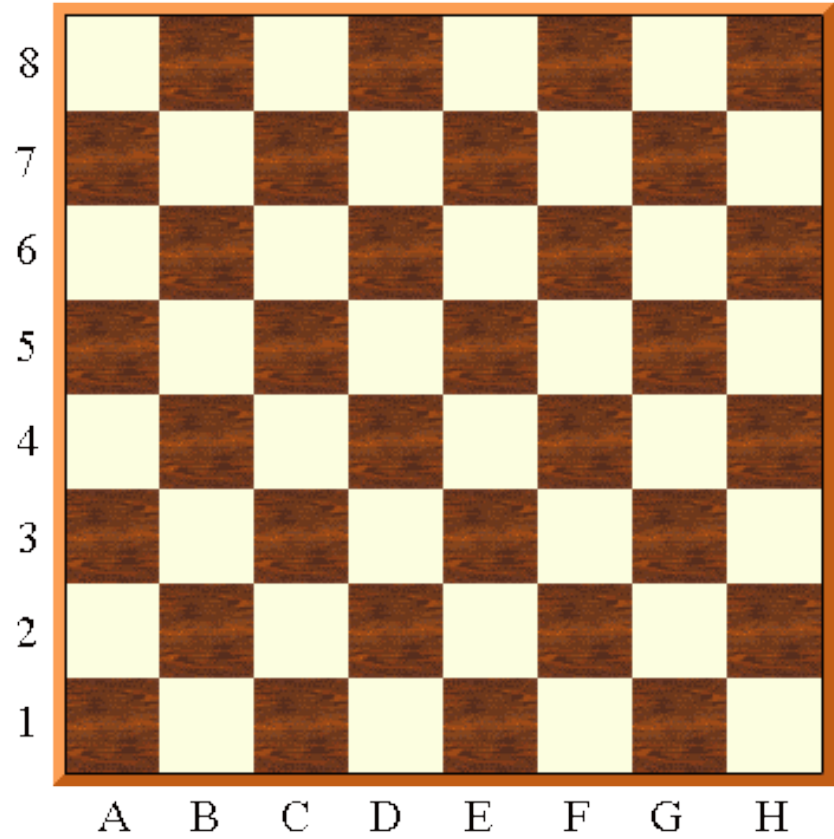
LA DAME AUX ÉCHECS



AVEC DEUX DAMES

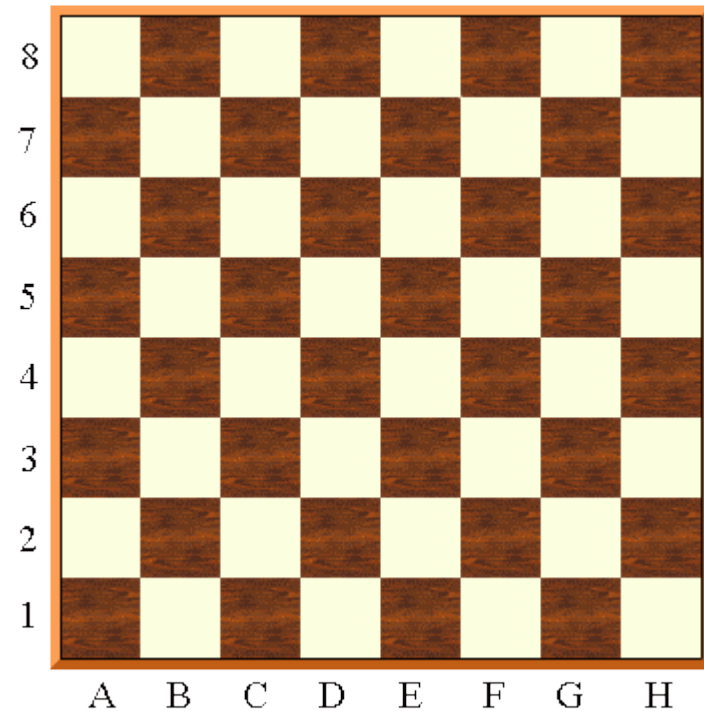


AVEC HUIT DAMES



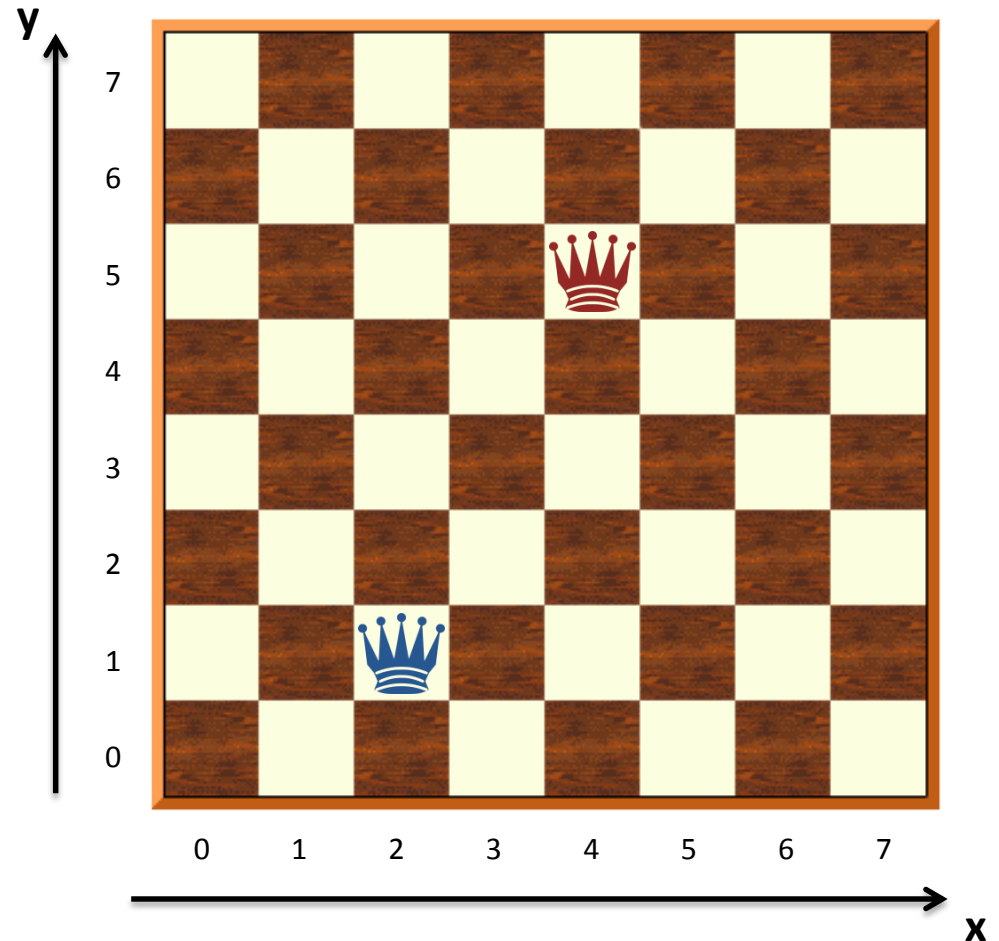
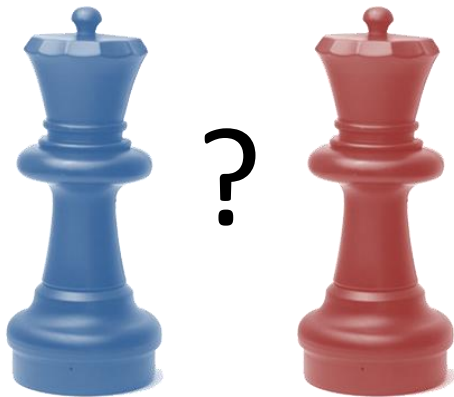
LE PROBLÈME DES HUIT DAMES

- Question : Combien de façons existe-t-il de poser huit dames sur un échiquier sans qu'aucune ne soit en prise avec une autre ?



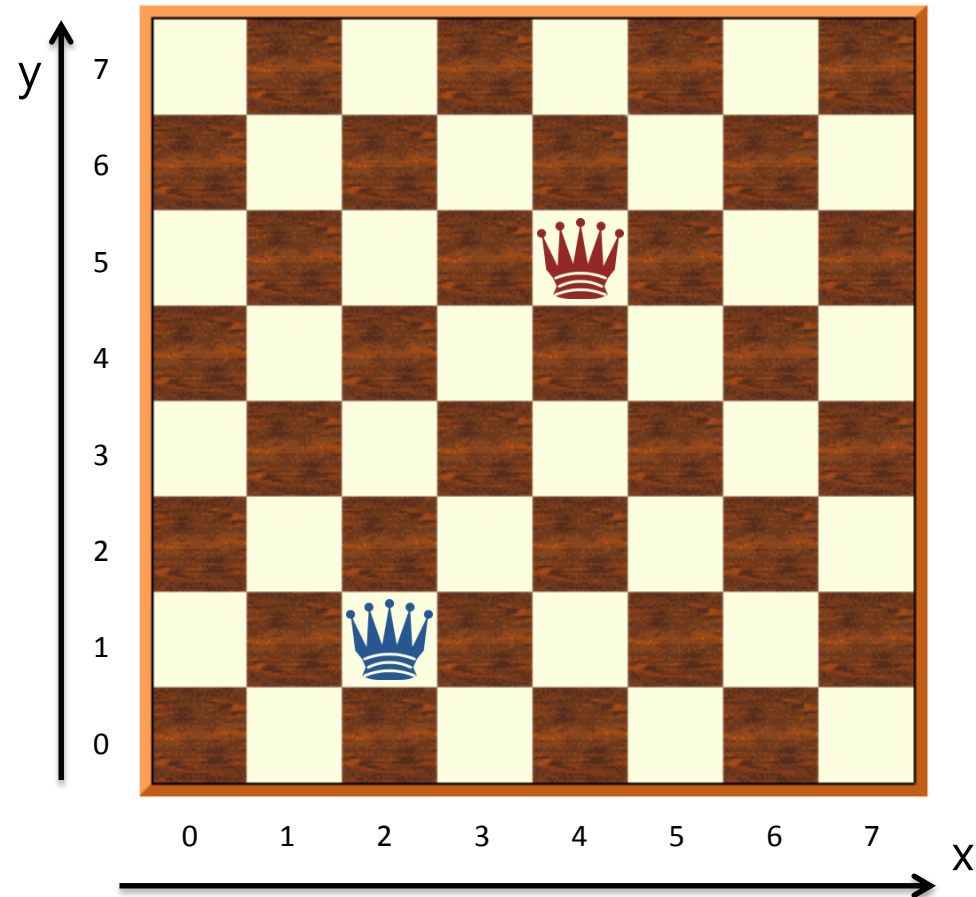
COMMENT SAVOIR SI DEUX DAMES SONT EN PRISE ?

- Question : Comment déterminer si les dames situées aux positions (x_1, y_1) et (x_2, y_2) sont en prise ?



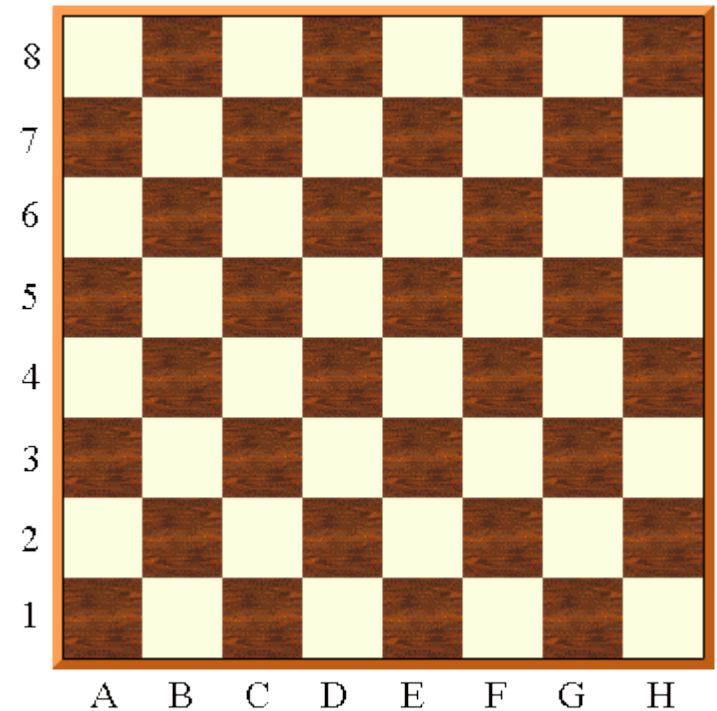
REPRÉSENTATION DU DAMIER

- L'échiquier est représenté par un tableau de tableaux
- Pour chaque case, on utilise un booléen pour indiquer la présence d'une dame :
 - Vrai si une dame est posée
 - Faux sinon
- Exemple
 - `echiquier[2][2] = False`
 - `echiquier[4][5] = True`



PARLONS CHIFFRES...

- Approche naïve : *Combien y a-t-il de façons de poser ces huit dames sur l'échiquier ?*
 - $64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57 = 178\,462\,987\,637\,760$
 - *S'il faut une microseconde (10^{-6} sec) pour tester chaque disposition, il faudrait plus de cinq ans avec cette méthode*

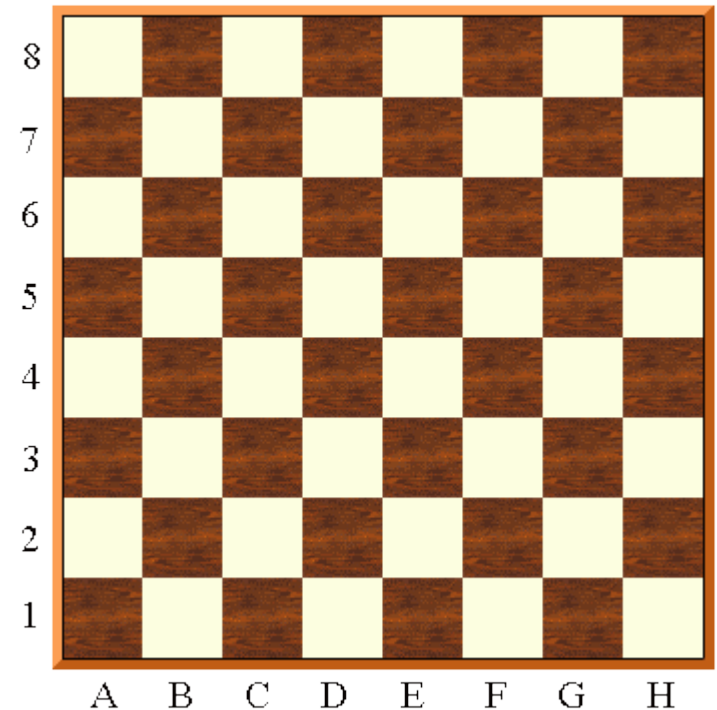


PARLONS CHIFFRES...

- Approche plus efficace : *Combien y a-t-il de façons de poser ces huit dames sur l'échiquier, sans considération d'ordre ?*

$$\rightarrow \binom{64}{8} = \frac{64!}{8! \times (64-8)!} = \frac{64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57}{8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1} = 4\,426\,165\,368$$

→ *S'il faut une microseconde (10^{-6} sec) pour tester chaque disposition, il faudra un peu plus d'une heure avec cette méthode*

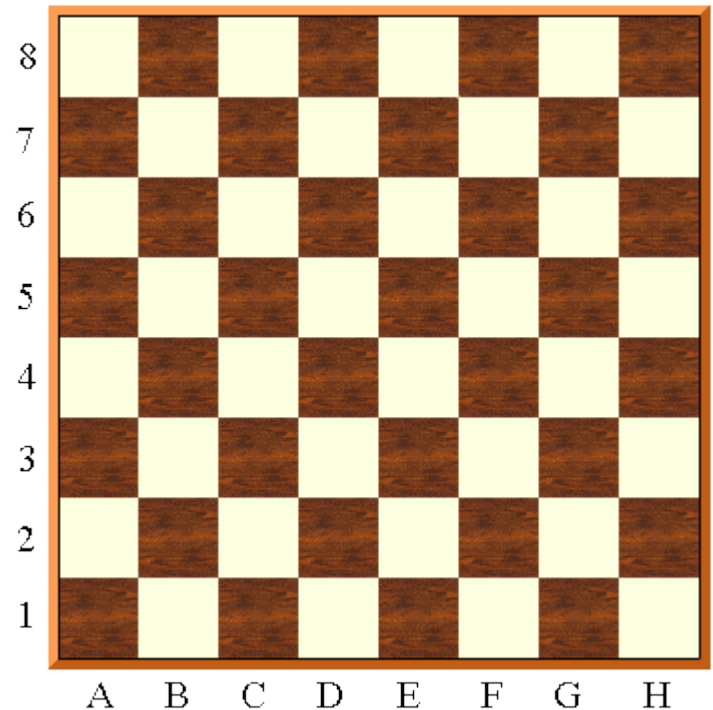


PARLONS CHIFFRES...

- Approche encore plus efficace : *Combien y a-t-il de façons de poser ces huit dames sur l'échiquier, sans considération d'ordre, sachant qu'il y en a exactement une par colonne ?*

→ $8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 = 167\,777\,216$

→ *S'il faut une microseconde (10^{-6} sec) pour tester chaque disposition, il faudra environ 17 secondes avec cette méthode*



RÉSULTATS POUR UN ÉCHIQUIER CLASSIQUE

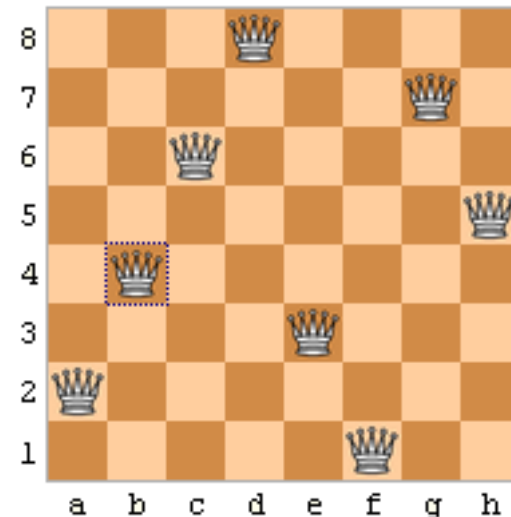
- Nombre de solutions pour un échiquier classique (8 cases) :
→ 92 solutions

- Arguments de symétries :
 - Symétrie centrale : avec une rotation de 90° , on obtient une autre solution
 - Symétrie axiale : en inversant haut et bas ou gauche et droite, on obtient une autre solution

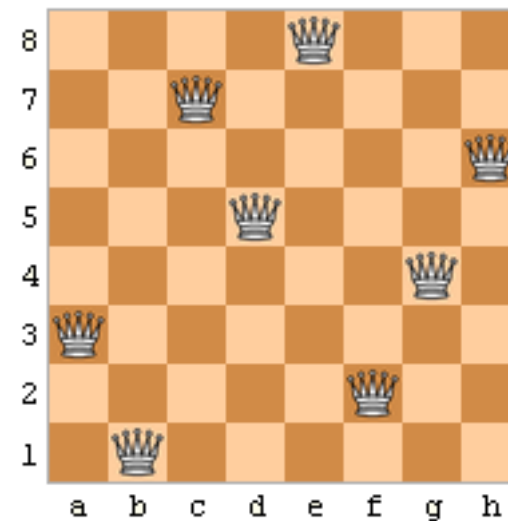
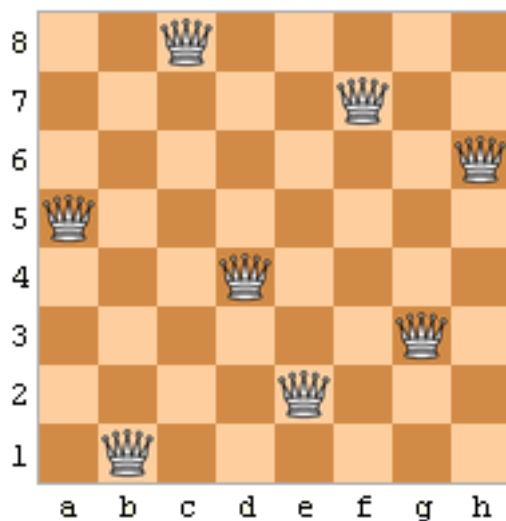
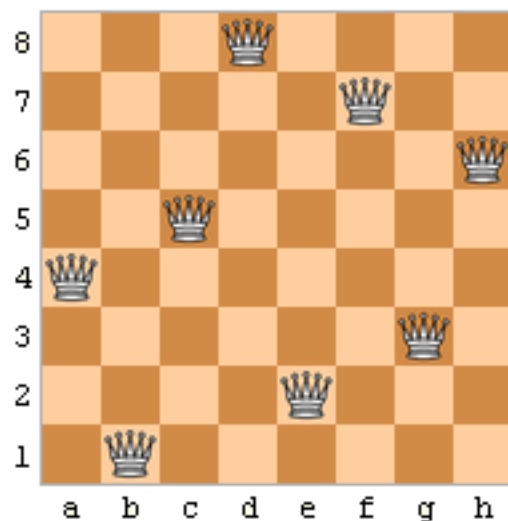
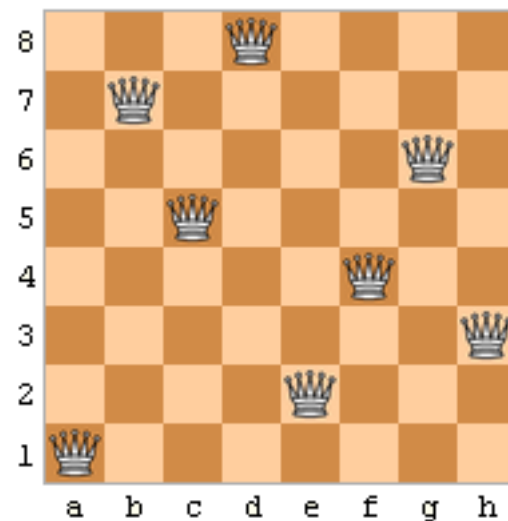
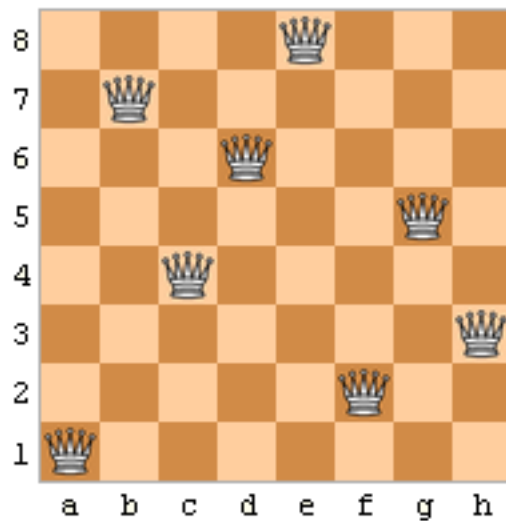
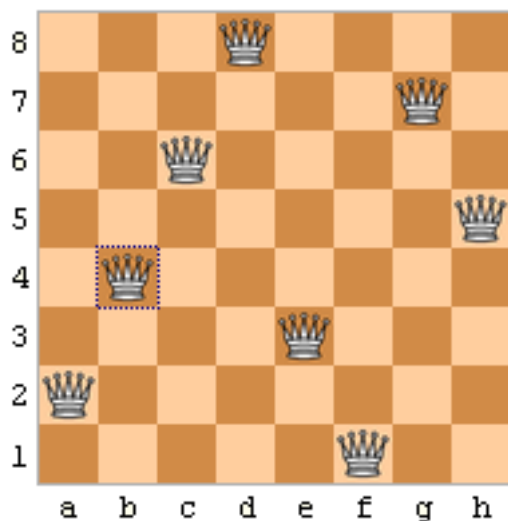
- Nombre de solutions « aux symétries » près :
→ 12 solutions

- Génération du problème :

*Combien de façons existe-t-il de poser **n dames** sur un échiquier à **n cases** sans qu'aucune ne soit en prise avec une autre ?*

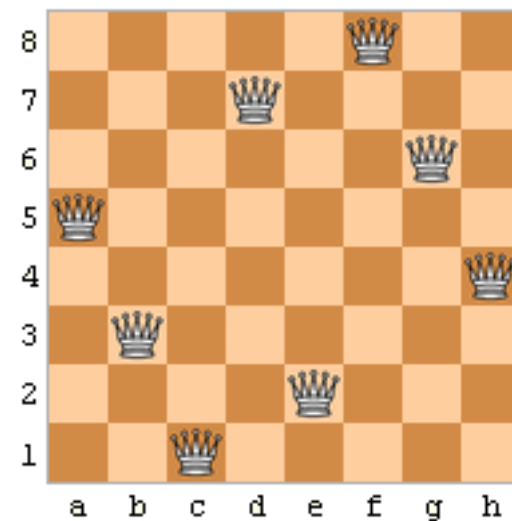
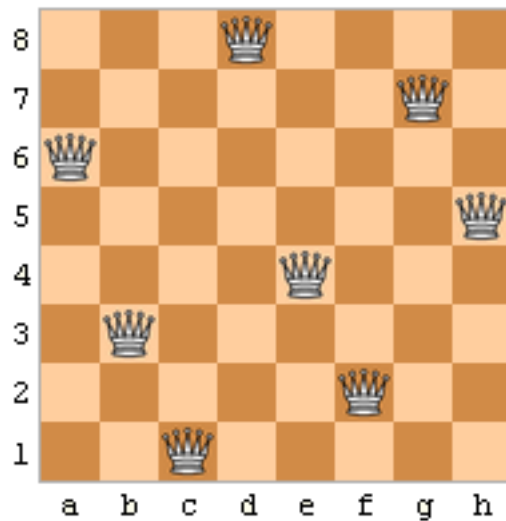
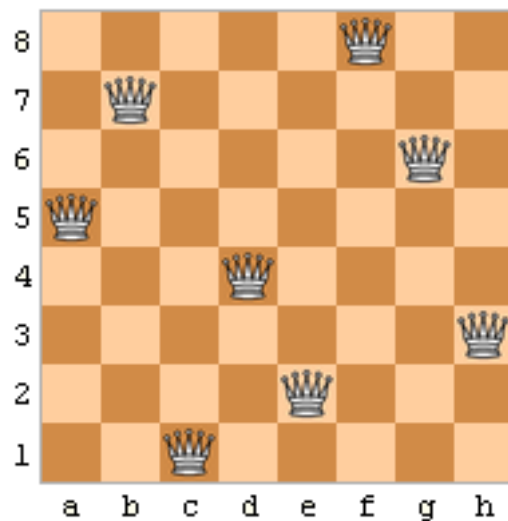
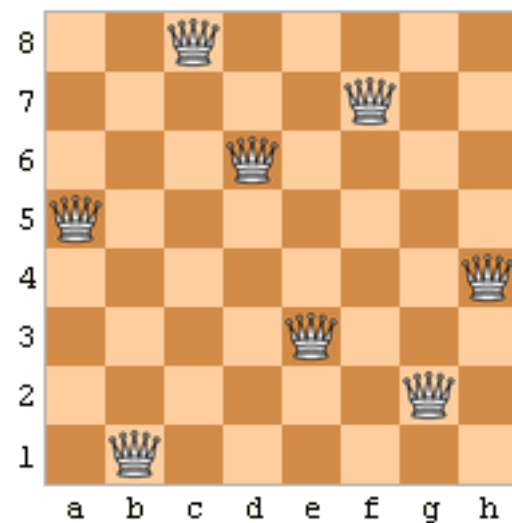
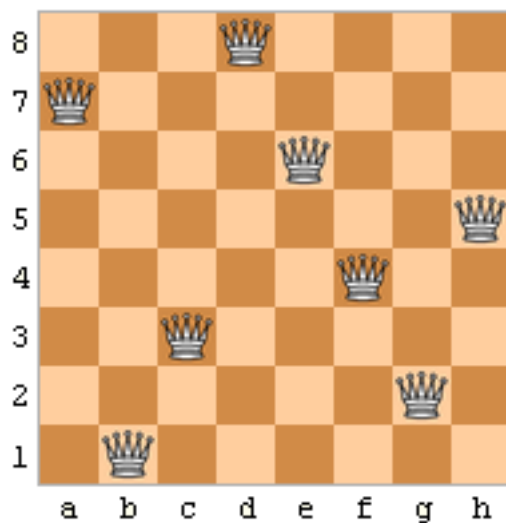
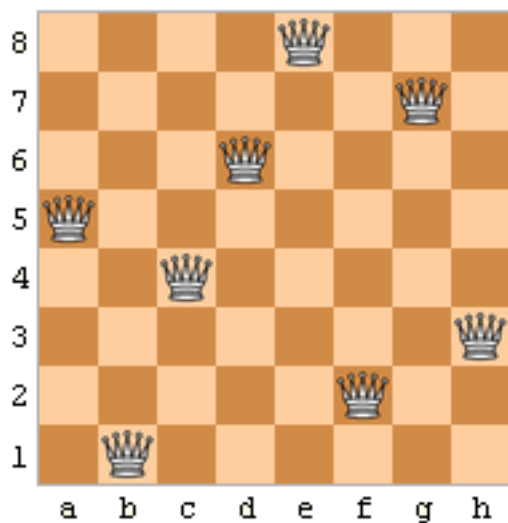


LES 12 SOLUTIONS



Les huit dames

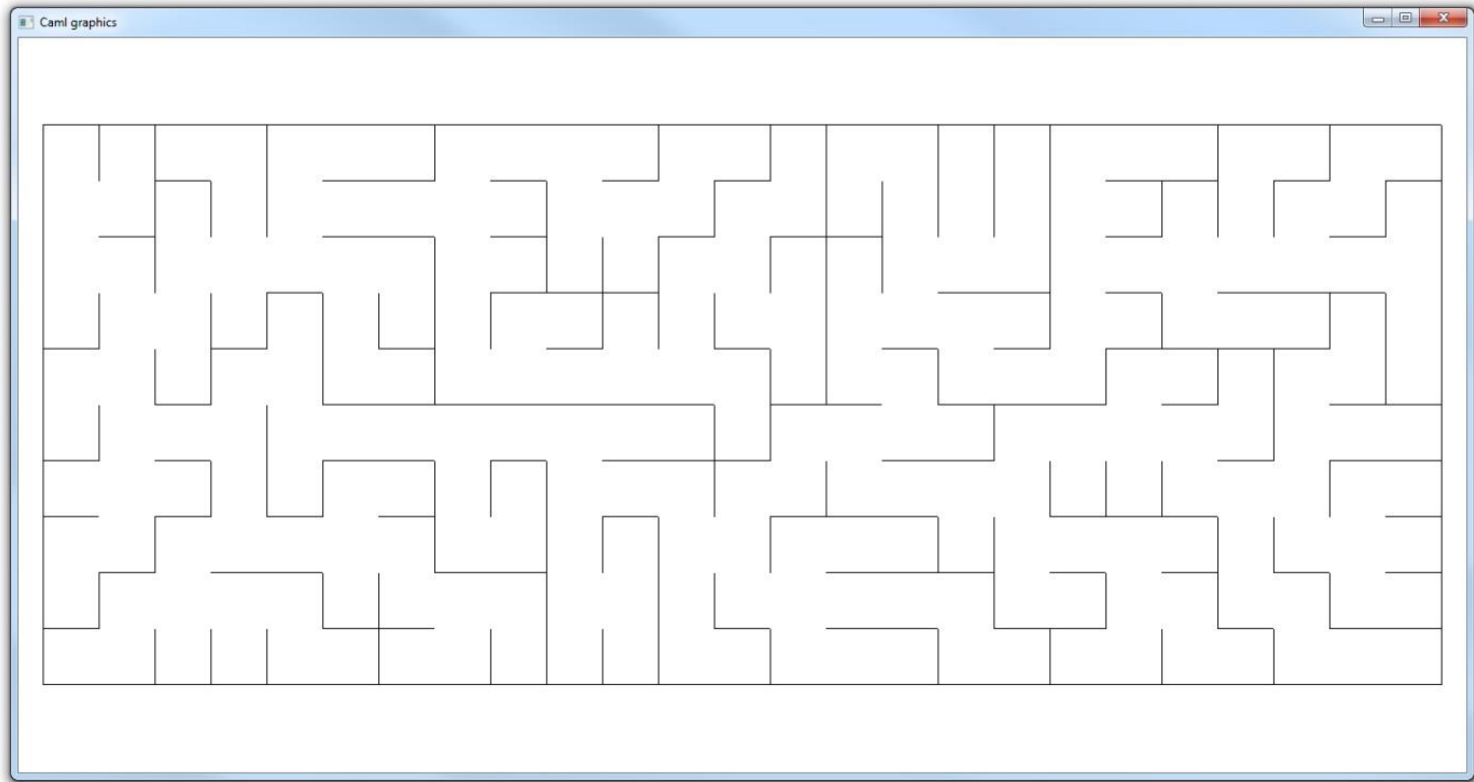
LES 12 SOLUTIONS



Les huit dames

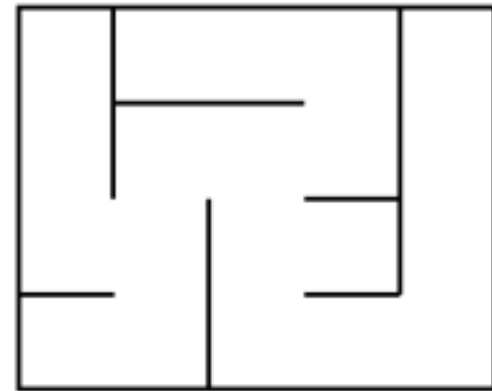
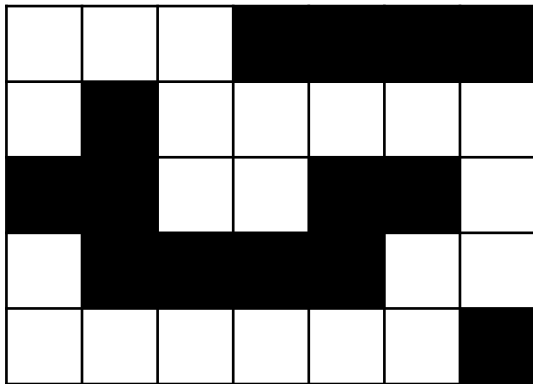
LABYRINTHES

NOTRE OBJECTIF



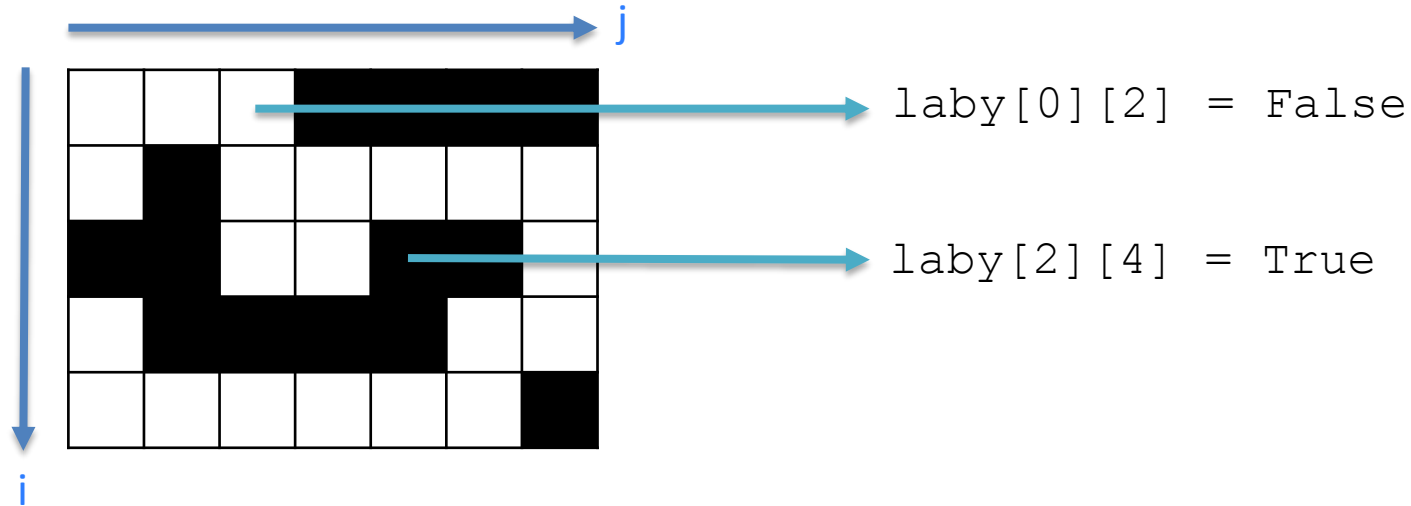
CHOIX DE LA REPRÉSENTATION

- **Question** : *Quelle structure de données pourrait-on utiliser pour représenter un labyrinthe ?*
 - Réponse : un tableau de tableaux
- **Question subsidiaire** : *Comment représenter chaque case ?*
 - Première réponse possible : avec un booléen
 - Deuxième réponse possible : avec plusieurs booléen



LABYRINTHES SIMPLES : AVEC UN SEUL BOOLÉEN

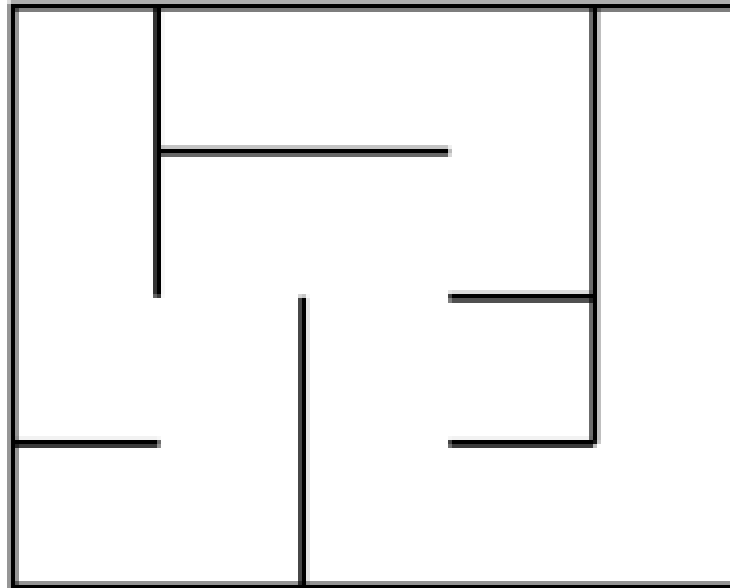
- **Question** : *Comment représenter un labyrinthe avec un seul booléen pour chaque case ?*



- Convention proposée :
 $\text{laby}[i][j] = \text{True}$ si la case contient un mur, False sinon
- Type correspondant en Python :
 - Une grille (de type `list list`)
 - Dans chaque case, un booléen (de type `bool`)

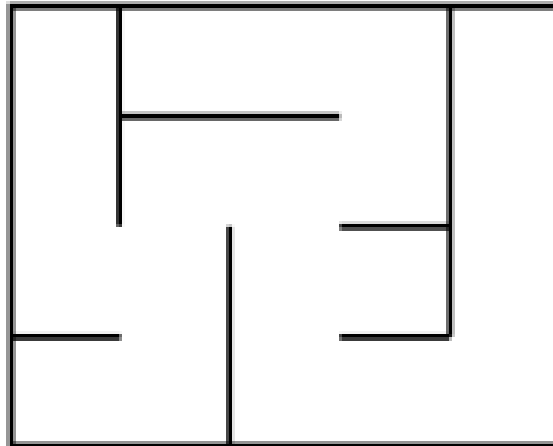
LABYRINTHE AVANCÉS

- Principe : *Les murs sont situés entre les cases*



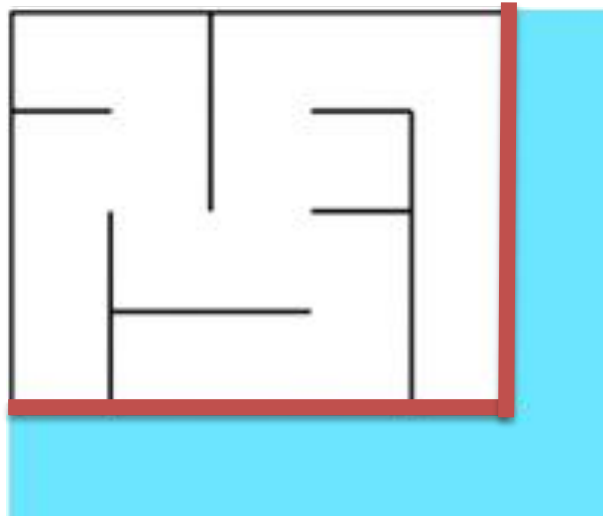
LABYRINTHE AVANCÉS : CHOIX DE LA REPRÉSENTATION

- Question : *Comment représenter un tel labyrinthe ?*
- Réponse :
 - On utilise une grille (tableau de tableaux)
 - Pour chaque case, on stocke deux informations :
 - *Est-ce qu'il y a un mur à gauche de cette case ?*
 - *Est-ce qu'il y a un mur au-dessus de cette case ?*



LABYRINTHE AVANCÉS : CHOIX DE LA REPRÉSENTATION

- **Question** : *Est-ce qu'on va pouvoir gérer toutes les cases avec cette représentation ?*



- **Réponse** : *Oui, en ajoutant une ligne et une colonne*
 - *Des cases qui servent juste à fermer le labyrinthe*
 - *Des cases auxquelles on n'accèdera pas par la suite*

MINI-TD

MINI-TD

- **Question 1** : Ecrire une fonction qui prend en argument un nombre entier positif et qui renvoie la liste des diviseurs de ce nombre
- **Question 2.** Ecrire une fonction qui prend en argument deux nombres entiers positifs et qui renvoie le plus grand diviseur commun de ces deux nombres
- **Question 3.** Ecrire une fonction qui prend en argument une liste et qui renvoie une version "miroir" de cette liste, sans modifier la liste originale.
- **Question 4.** Résoudre le problème des huit dames

PROCHAINE SÉANCE

Lundi 17 novembre

[TD] LABYRINTHES

