Structures multidimensionnelles

Jeudi 1er octobre

Option Informatique

Ecole Alsacienne

Plan

1. Pile et récursivité
2. Structures multidimensionnelles
3. Les grilles en Python
4. Les huit dames
5. Labyrinthes
6. Mini-TD : les huit dames

Pile et récursivité

Petite parenthèse récursive

- Rappels
  - Une fonction récursive est une fonction qui s'appelle elle-même
  - Une telle fonction doit comporter
    - Un (ou plusieurs) cas de base
    - Un (ou plusieurs) cas récursif(s)

- Exemple classique :

```
def factorielle(n):
    if (n == 0):
        return 1
    else:
        return n * (factorielle (n-1))
```

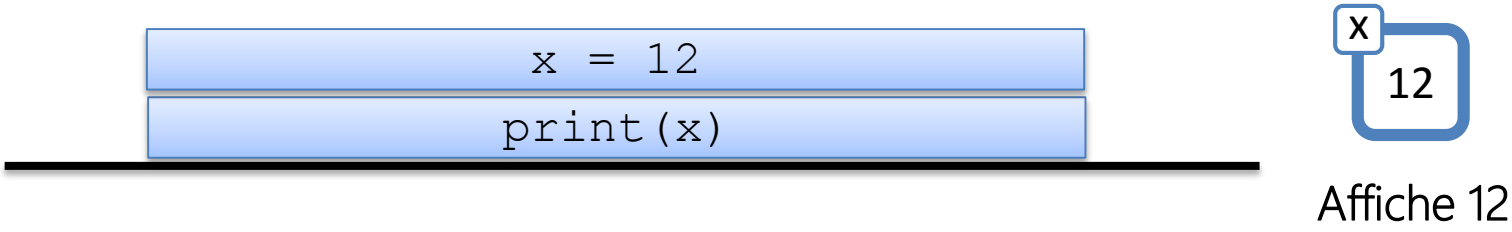
- Mais au final, comment ça marche ?

UNE PILE DE CHOSE À FAIRE

- Commençons par un exemple simple :

```
x = 12
print(x)
```

- On peut voir cette séquence d'instructions comme une pile de "choses à faire" :

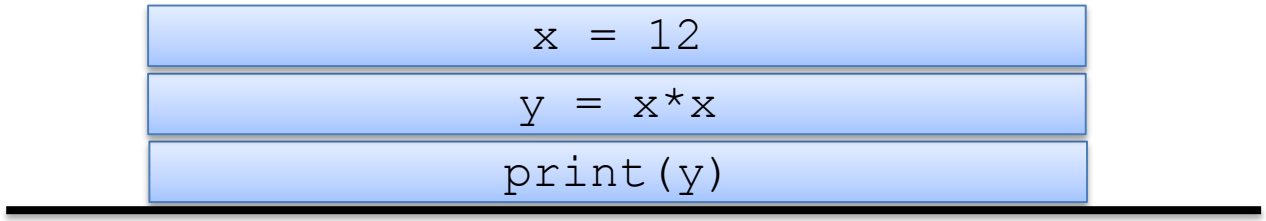


- L'ordinateur va traiter cette pile dans l'ordre : tant que l'instruction sur le dessus de la pile n'a pas été traitée, il ne passe pas à la suite.

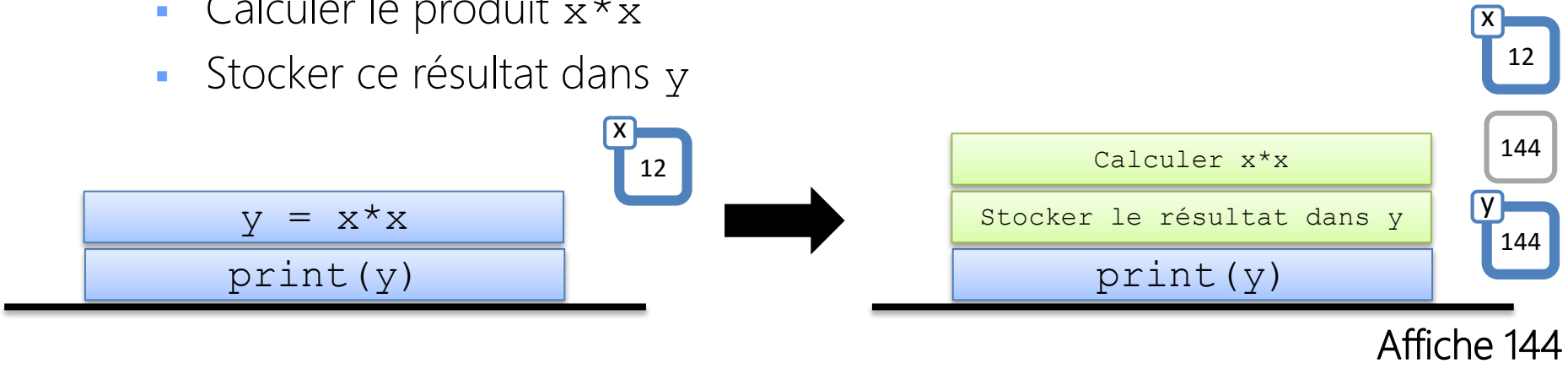
Une pile de chose à faire

- Il peut arriver qu'une même ligne contienne plusieurs instructions :

```
x = 12
y = x*x
print(y)
```



- Une fois la première ligne traitée, l'ordinateur passe à la deuxième… qui contient en fait deux instructions :
  - Calculer le produit `x*x`
  - Stocker ce résultat dans `y`



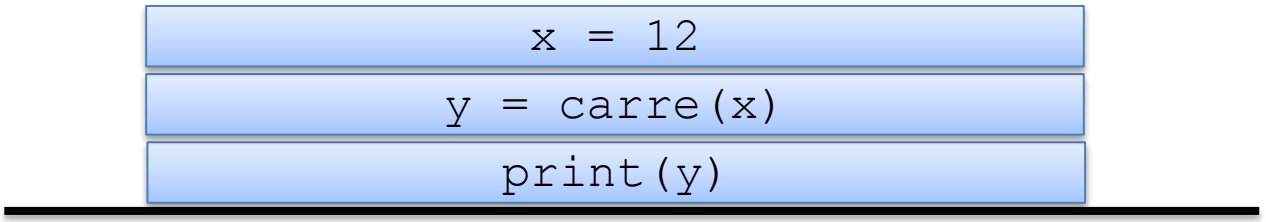
Affiche 144

UNE PILE DE CHOSE À FAIRE

- Une séquence d'instructions peut faire appel à une fonction

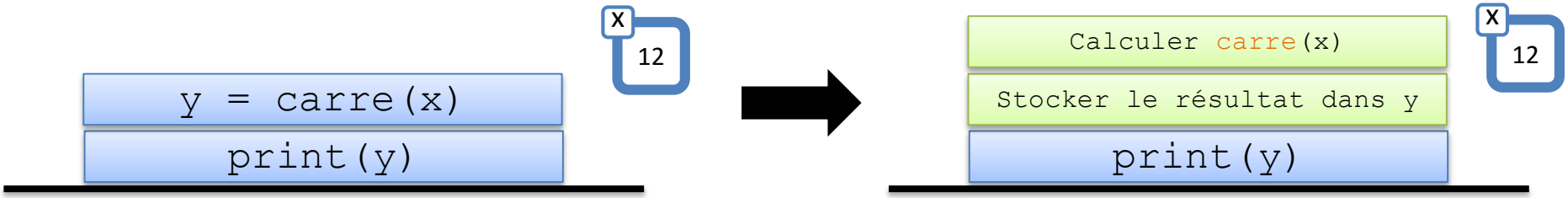
```
def carre(n):
    c = n*n
    return c

x = 12
y = carre(x)
print(y)
```

- La déclaration de fonction n'apparaît pas directement dans cette pile de choses à faire :



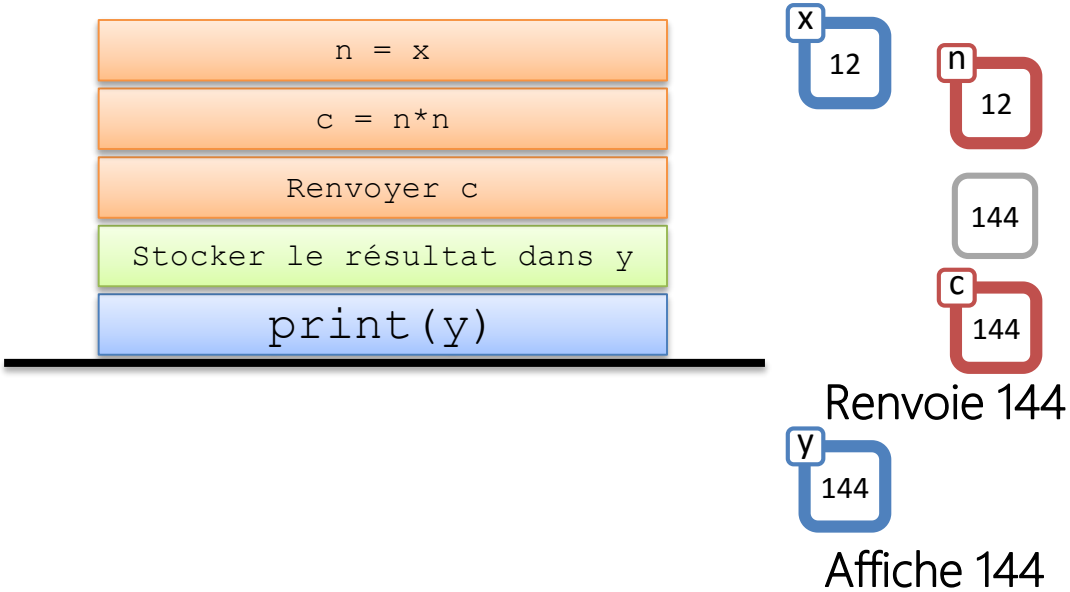
UNE PILE DE CHOSE À FAIRE

- Après avoir traité le premier élément de la pile, on a :



- Comme on fait appel à la fonction `carre`, on ajoute le contenu de cette fonction à la pile des "choses à faire" :
  - Il faut d'ailleurs définir la valeur des arguments de la fonction

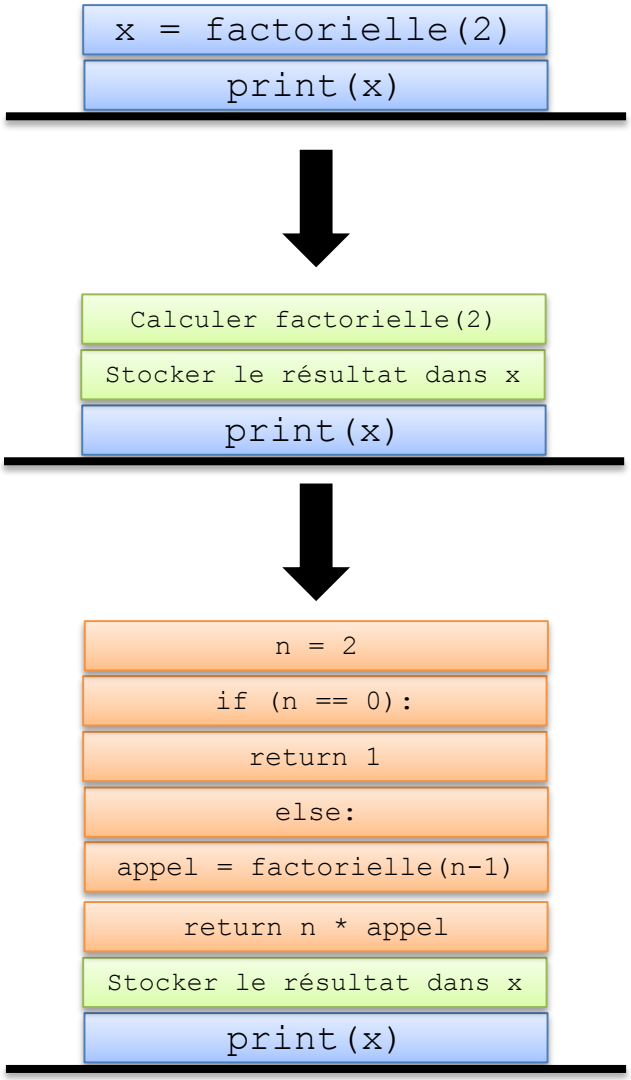
```
def carre(n):
    c = n*n
    return c

x = 12
y = carre(x)
print(y)
```



Petite parenthèse récursive

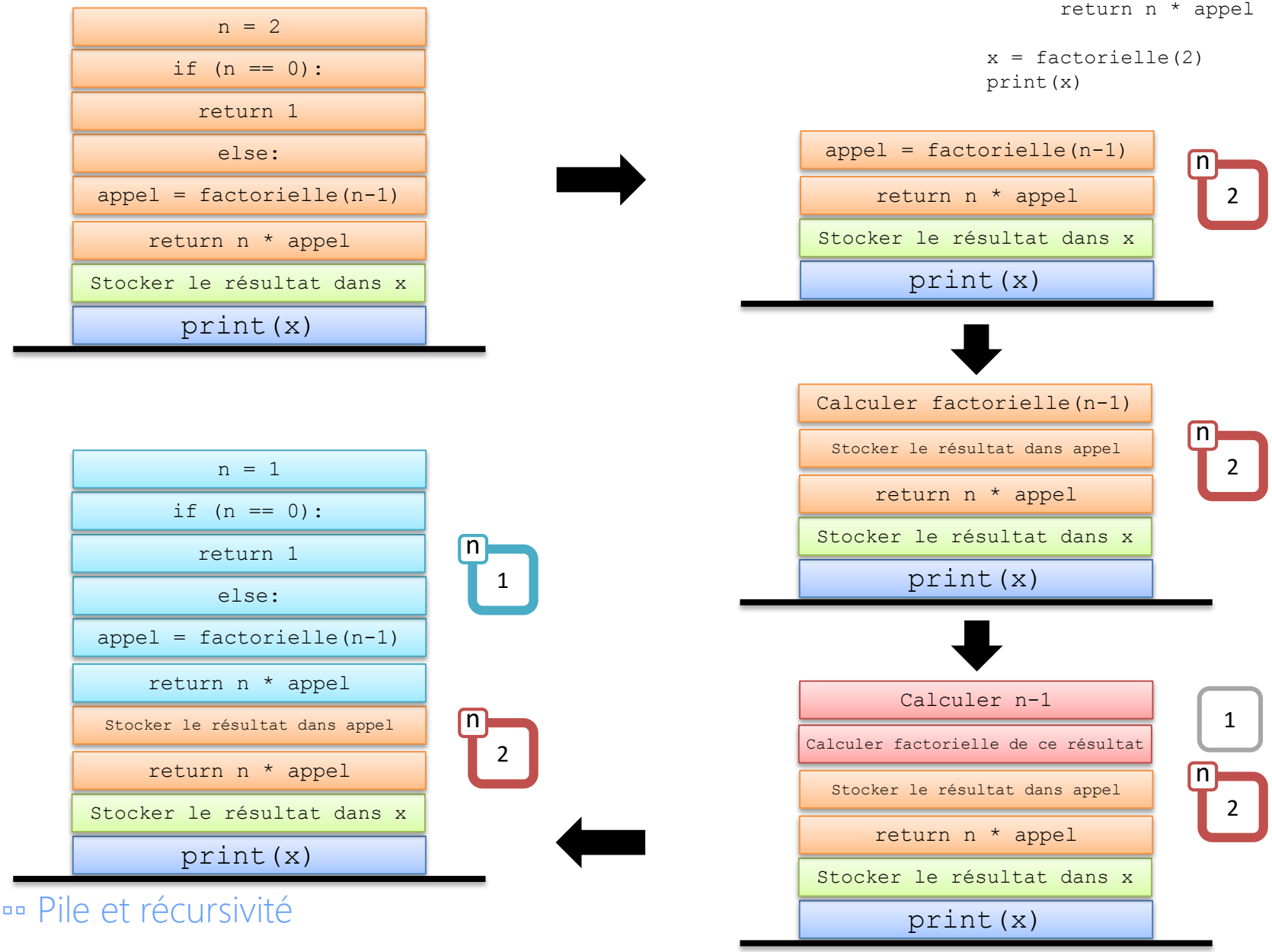
- C'est cette idée de pile de "choses à faire" qui s'applique dans le cas d'une fonction récursive :

```
def factorielle(n):
    if (n == 0):
        return 1
    else:
        appel = factorielle(n-1)
        return n * appel

x = factorielle(2)
print(x)
```



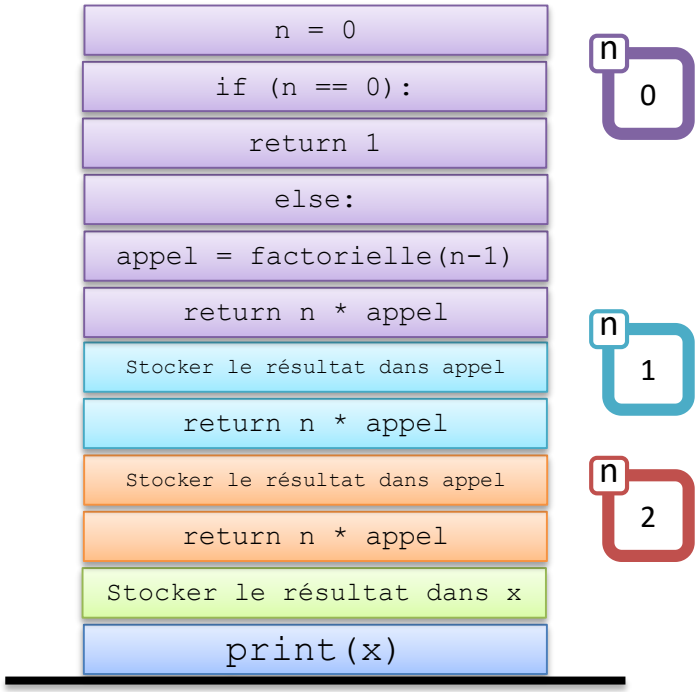
Petite parenthèse récursive

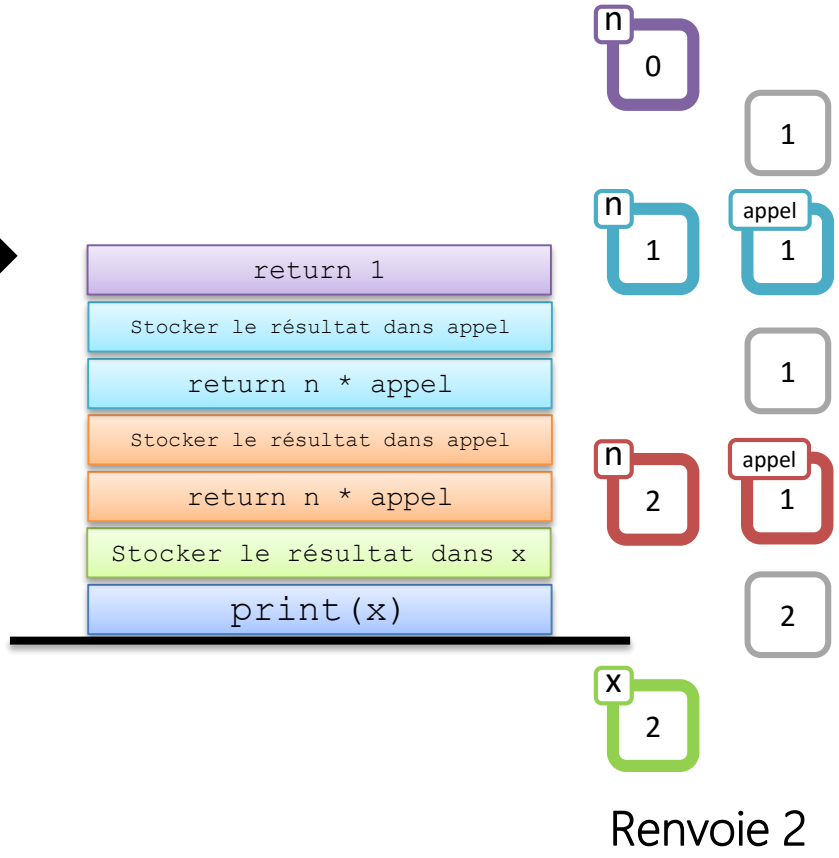
```
def factorielle(n):
    if (n == 0):
        return 1
    else:
        appel = factorielle(n-1)
        return n * appel

x = factorielle(2)
print(x)
```



Petite parenthèse récursive

```
def factorielle(n):
    if (n == 0):
        return 1
    else:
        appel = factorielle(n-1)
        return n * appel

x = factorielle(2)
print(x)
```





Petite parenthèse récursive

- Cette notion de "pile de choses à faire" est présente dans la plupart des langages informatiques

- On lui donne différents noms :
  - Pile (en anglais *stack*)
  - Pile d'appels (en anglais *call stack*)
  - Pile d'exécution (en anglais *execution stack*)

- Le mécanisme est en pratique un peu plus compliqué que ça (variables locales, gestion de la mémoire, etc.)

- Remarque : On parle de récursivité terminale lorsque l'appel récursif est la dernière instruction d'une fonction (sans calcul supplémentaire)

TRACEBACK

- En vérité, vous vous êtes déjà intéressés à cette pile…

```
def fonction_3(z):
    c = 3 // z
    return c

def fonction_2(y):
    b = fonction_3(y)
    return b

def fonction_1(x):
    a = fonction_2(x)
    return a

resultat_A = fonction_1(2)
Resultat_B = fonction_2(0)
```

- En effet, dans le cas d'une erreur, on retrouve la pile des appels en cours

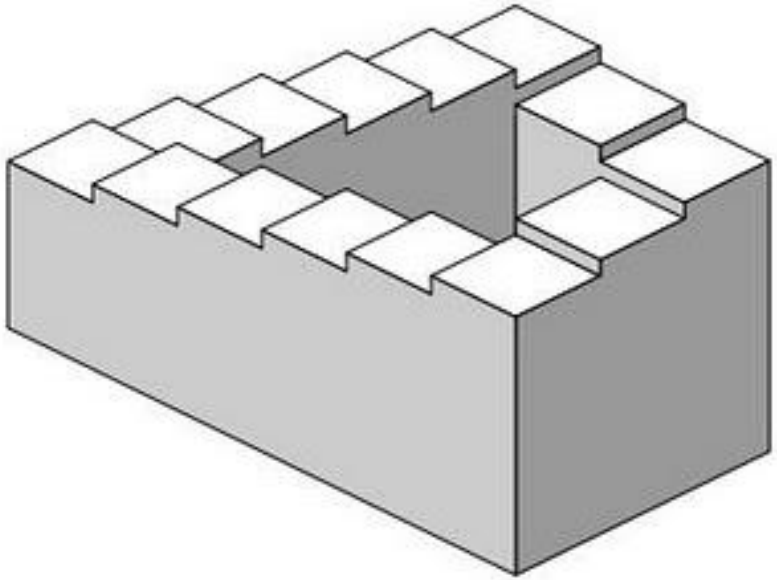
```
Traceback (most recent call last):
  File "C:\Users\Romain André-Lovichi\Documents\Teaching\2015-2016\Cours\Terminales\01 -
Introduction\tp1.py", line 14, in <module>
    Resultat_B = fonction_2(0)
  File "C:\Users\Romain André-Lovichi\Documents\Teaching\2015-2016\Cours\Terminales\01 -
Introduction\tp1.py", line 6, in fonction_2
    b = fonction_3(y)
  File "C:\Users\Romain André-Lovichi\Documents\Teaching\2015-2016\Cours\Terminales\01 -
Introduction\tp1.py", line 2, in fonction_3
    c = 3 // z
ZeroDivisionError: integer division or modulo by zero
```

- On parle aussi parfois de *stacktrace*

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 :

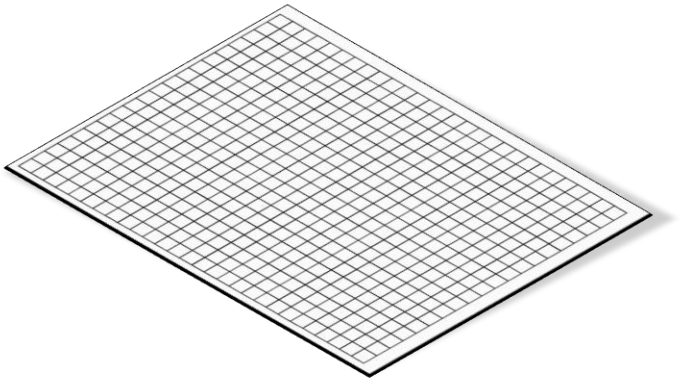
- Une droite
- Un tableau
- Informellement, un truc "rectiligne"

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 dimensions**, c'est par exemple :

- Un plan
- Un tableau à double entrée

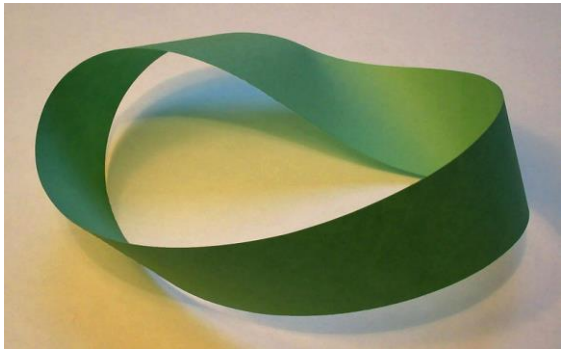


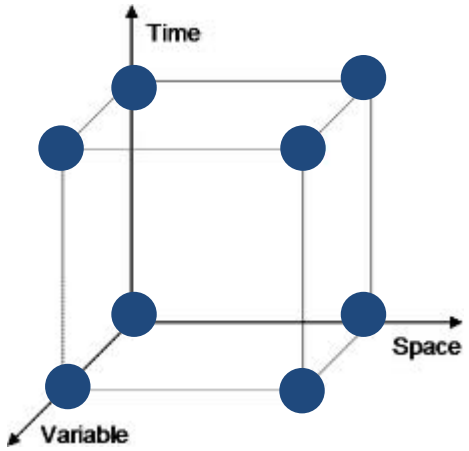
	USA	Chine	Russie
Prénom	Barack	Xi	Vladimir
Nom	Obama	Jinping	Poutine

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

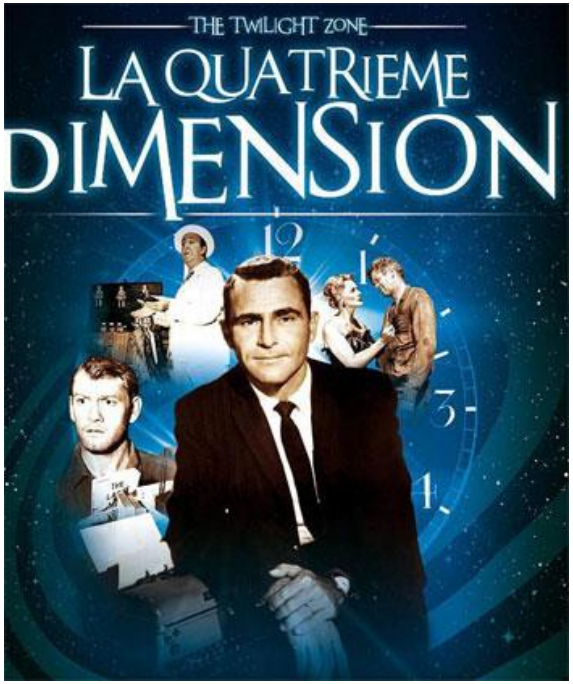




DIMENSION 4 (ET PLUS)

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

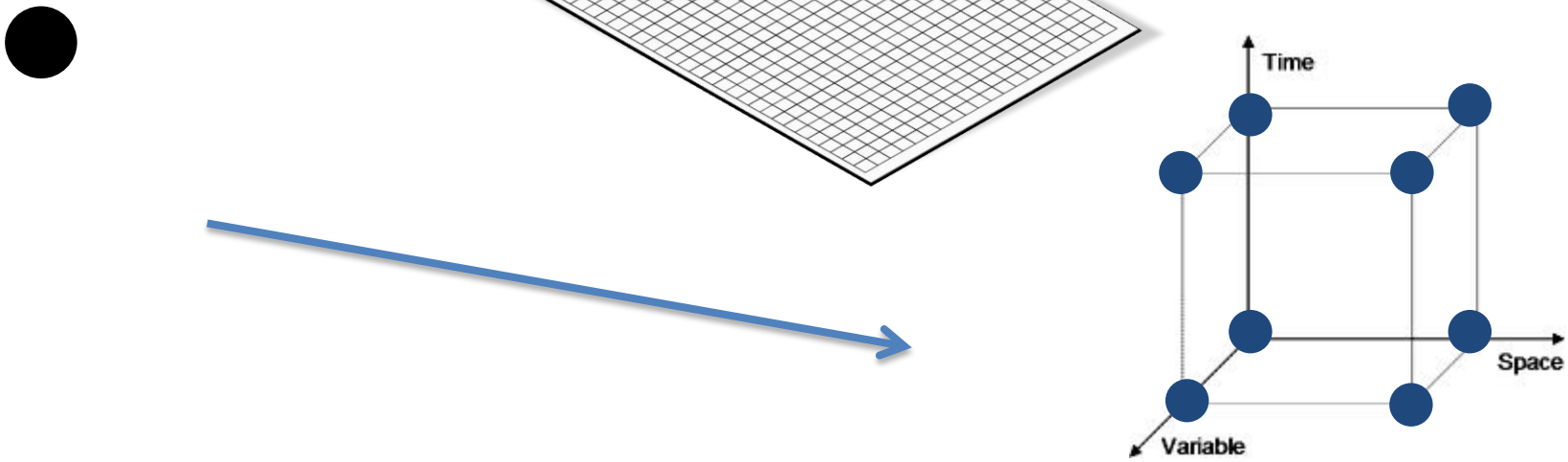
Une structure de dimension 0, c'est :

- 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

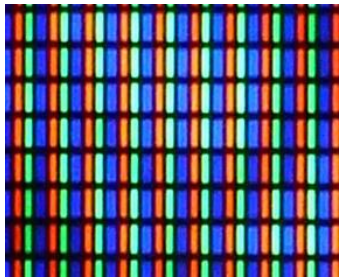
- *Question : Quelle structure vous semblerait adaptée pour représenter l'objet suivant ?*

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
```

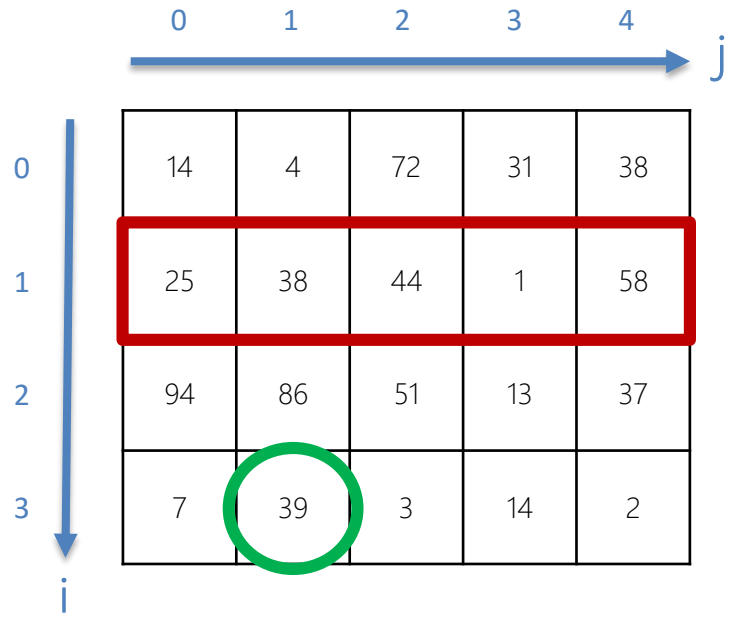
- Exemple :

```
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 :

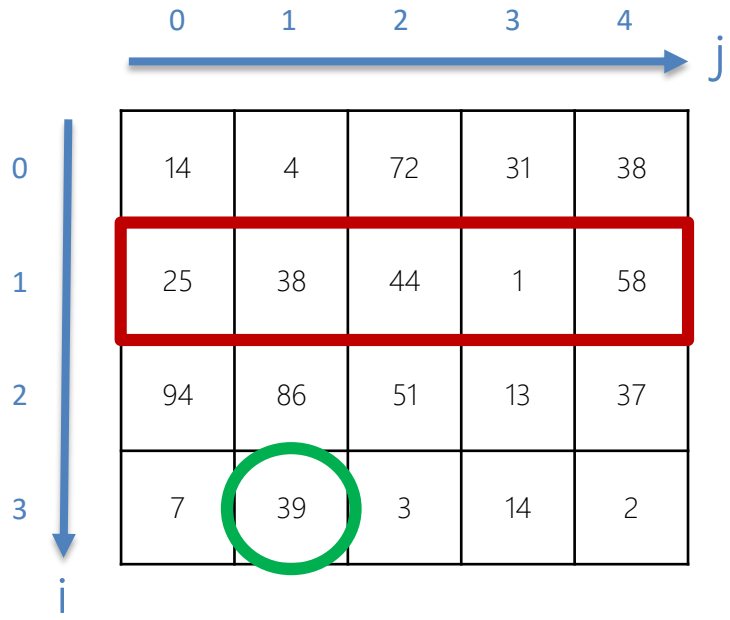


- Si `t` représente tout le tableau (de type `list list`)
  - `t[3][1]` représente une case contenant 39 (de type `int`)
  - `t[1]` représente la deuxième ligne du tableau (de type `list`)
  - `t` est en quelque sort un « tableau de lignes »

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):
    return len(t)

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



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

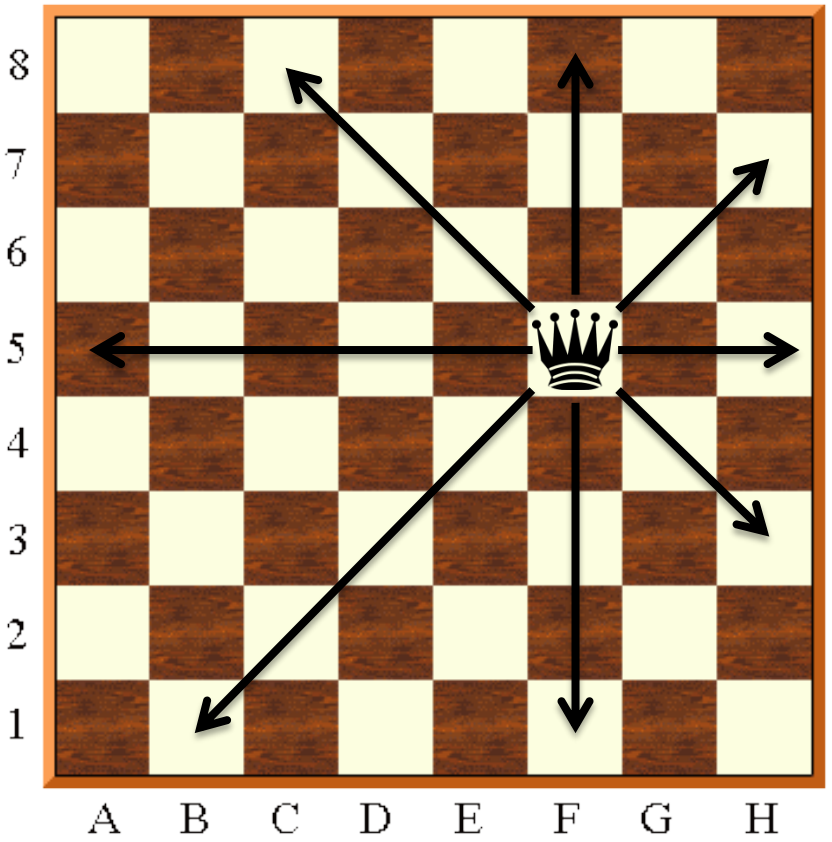
```
g0 = [([0] * 2)] * 3
print(g0)
g0[0][0] = 1
print(g0)
```

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

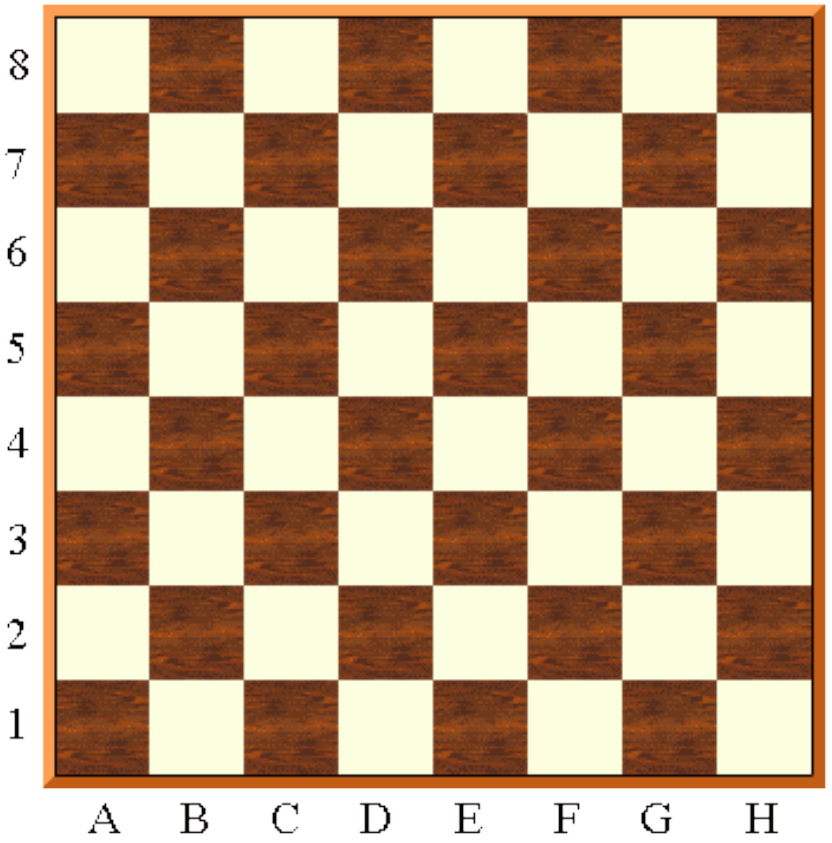
- Attention aux références !

Les huit dames

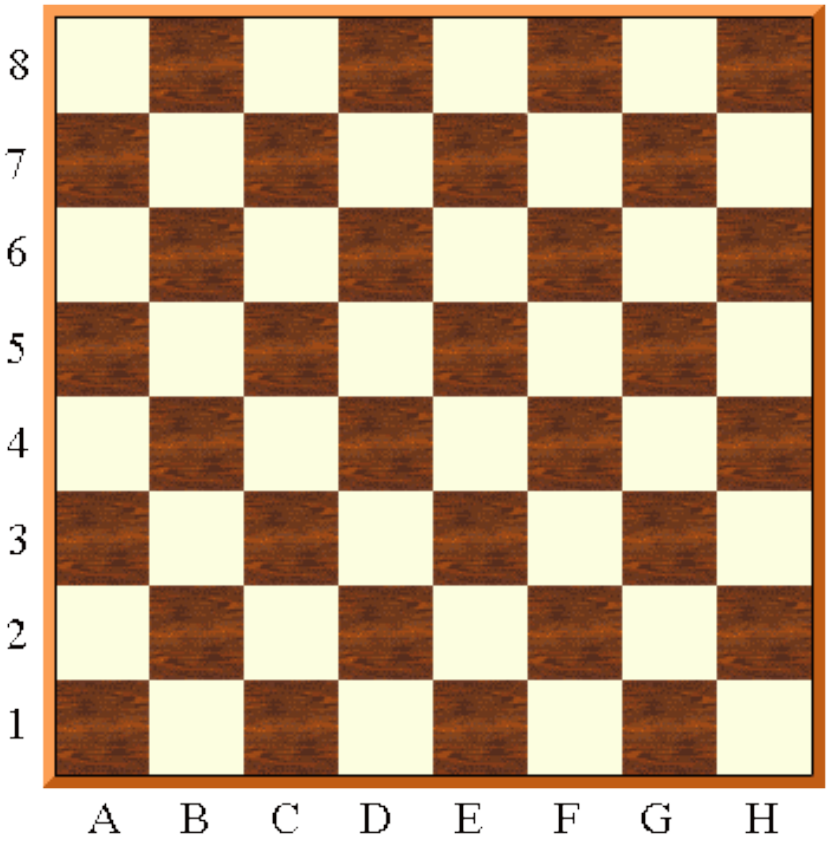
La dame aux échecs



Avec deux dames

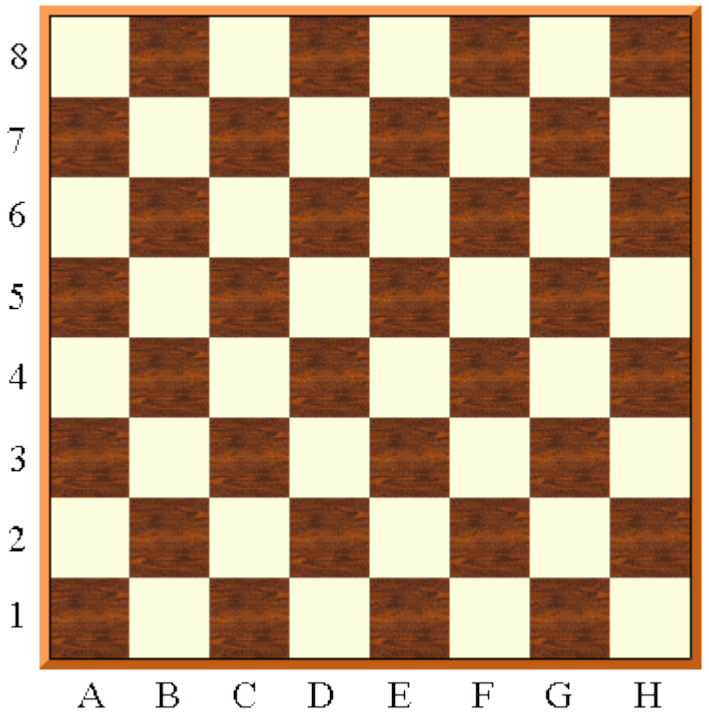


AVEC HUIT DAMES



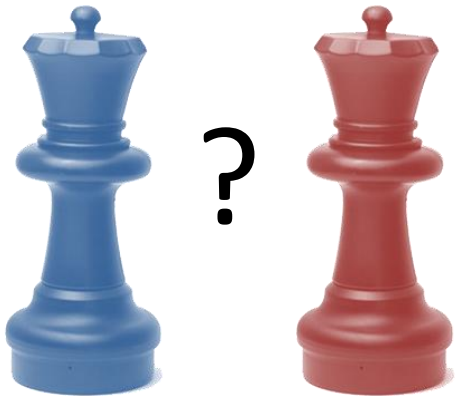
Le problème des huit dames

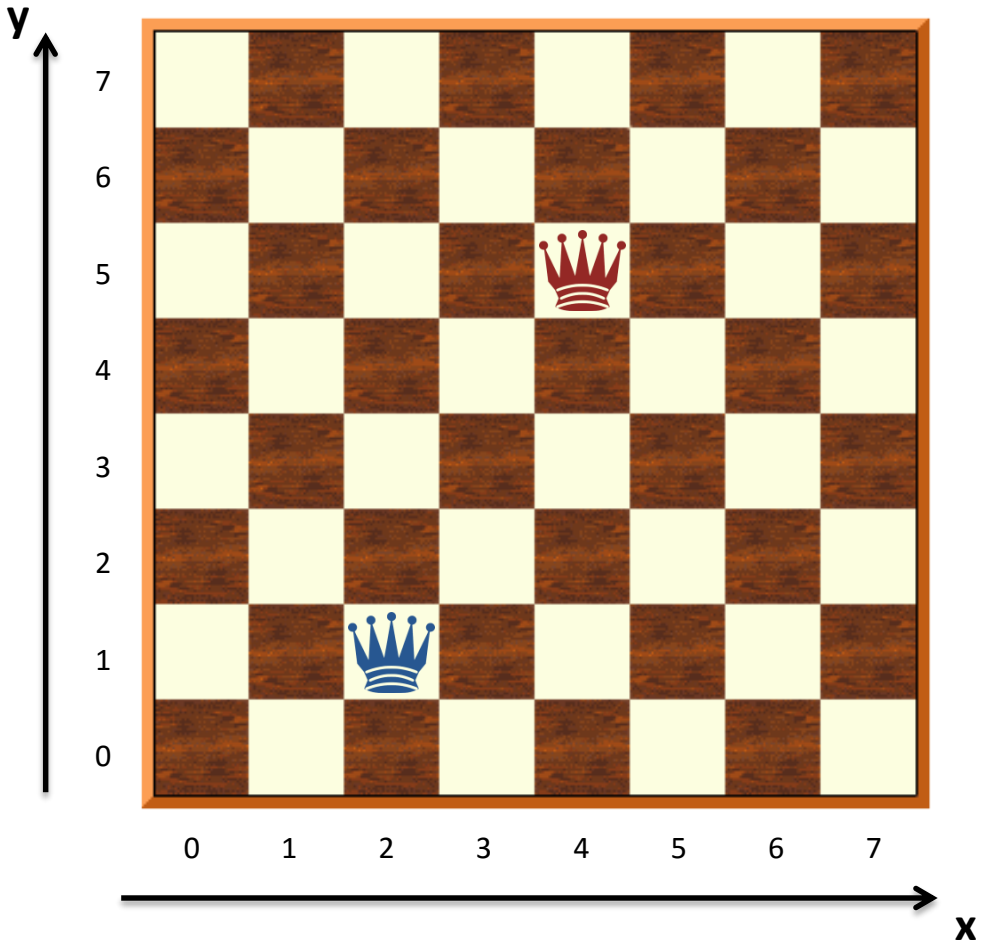
- *Question : Combien de façons existe-t-il de poser huit dames sur un échiquier sans qu'aucune ne soit en prise avec une autre ?*



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 ?*



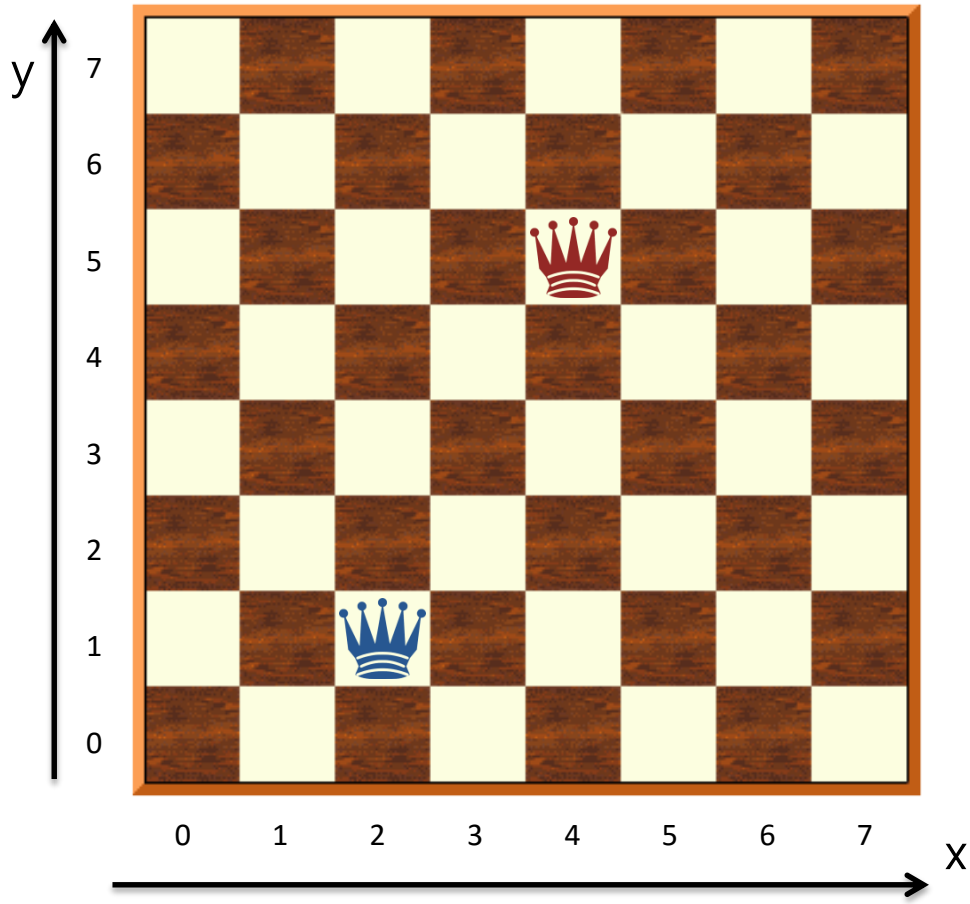


Représentation du damier

- L'échiquier est représenté par un tableau de tableaux

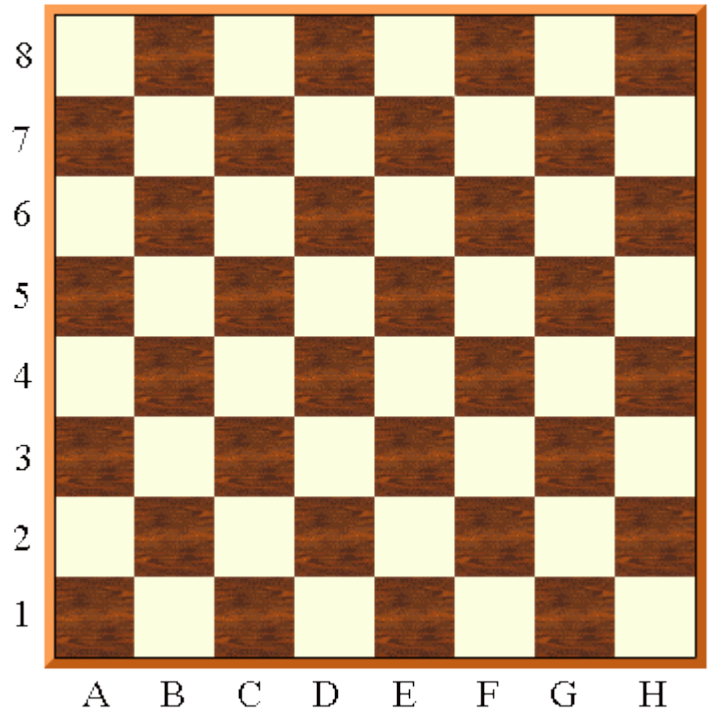
- Pour chaque case, on utilise un booléen pour indiquer la présence d'une dame :
  - Vrai si une dame est posée
  - Faux sinon

- Exemple
  - `echiquier[2][2] = False`
  - `echiquier[4][5] = True`



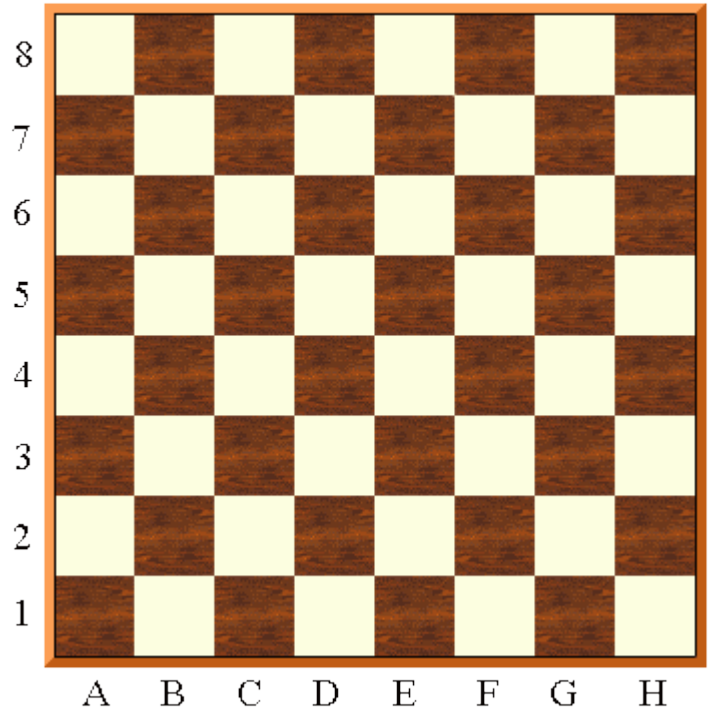
PARLONS CHIFFRES…

- 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*



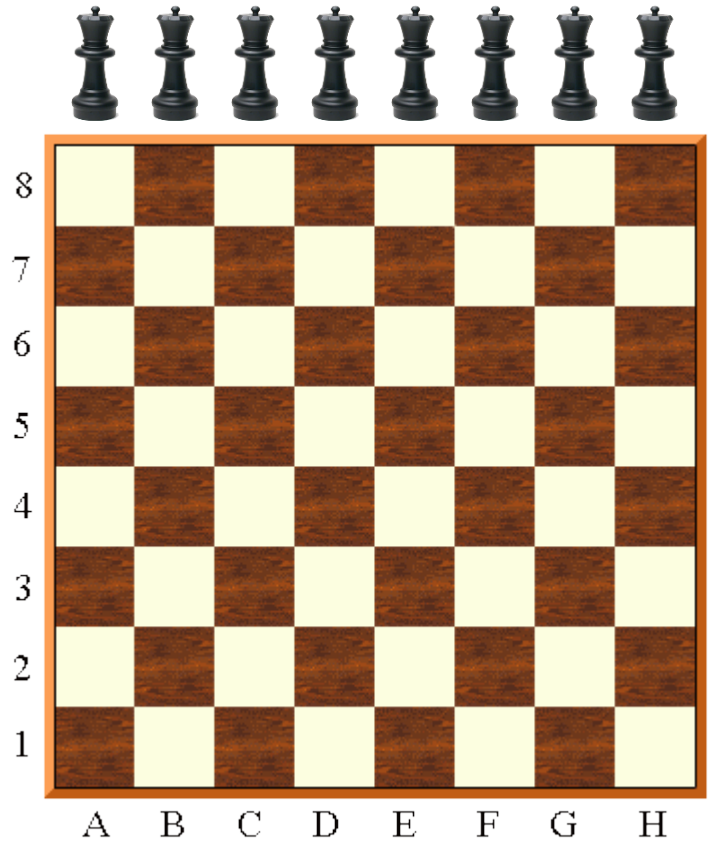
PARLONS CHIFFRES…

- Approche plus efficace : *Combien y a-t-il de façons de poser ces huit dames sur l'échiquier, sans considération d'ordre ?*
  - ➔ $\binom{64}{8} = \frac{64!}{8! \times (64-8)!} = \frac{64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57}{8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1} = 4\,426\,165\,368$
  - ➔ *S'il faut une microseconde (10^{-6} sec) pour tester chaque disposition, il faudra un peu plus d'une heure avec cette méthode*



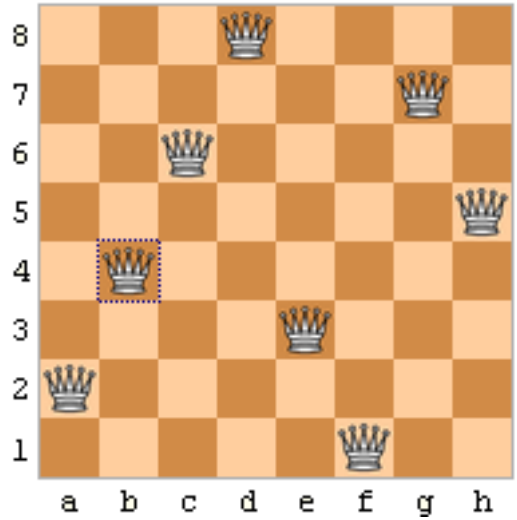
PARLONS CHIFFRES…

- 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



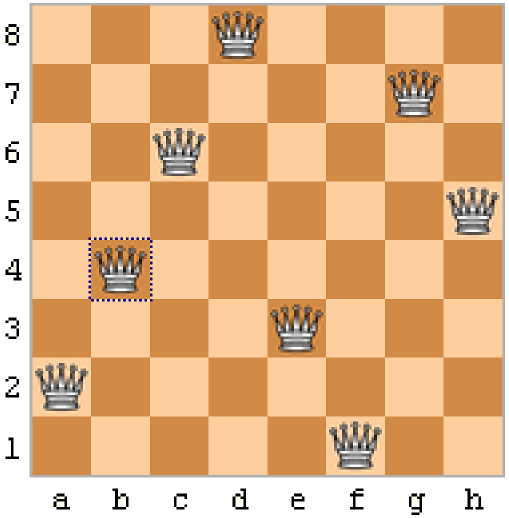
- Arguments de symétries :
  - Symétrie centrale : avec une rotation de 90°, on obtient une autre solution
  - Symétrie axiale : en inversant haut et bas ou gauche et droite, on obtient une autre solution

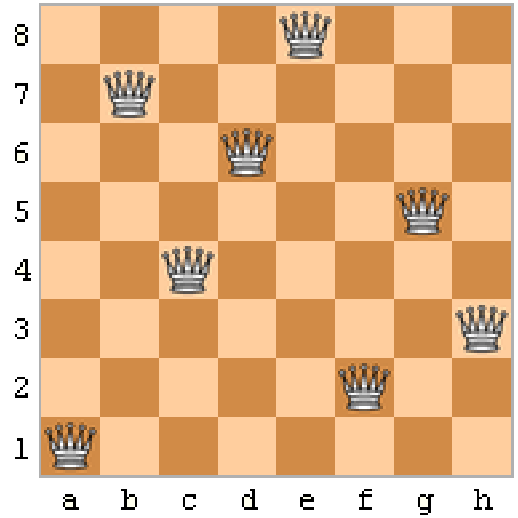
- Nombre de solutions « aux symétries » près :
  - ➔ 12 solutions

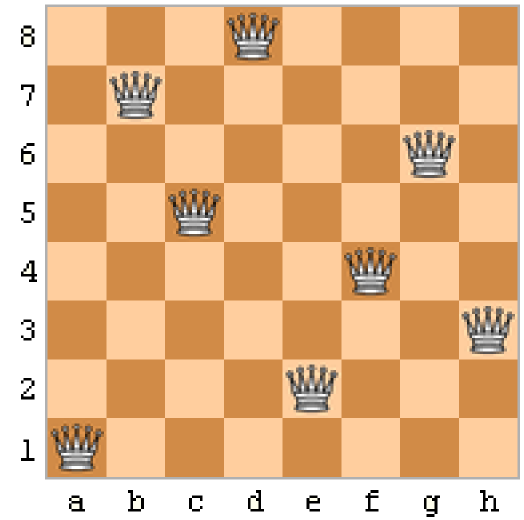
- Génération du problème :

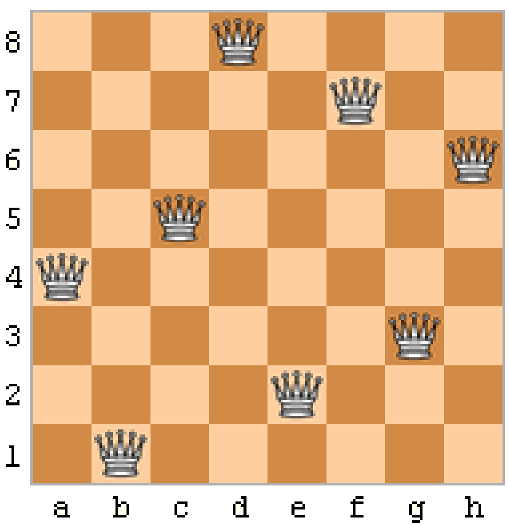
  *Combien de façons existe-t-il de poser **n dames** sur un échiquier à **n cases** sans qu'aucune ne soit en prise avec une autre ?*

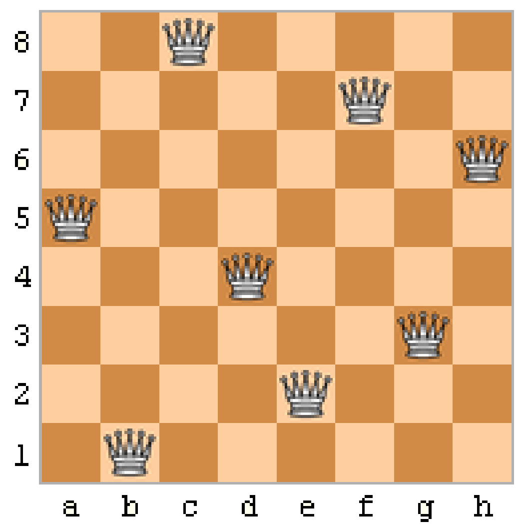
Les 12 solutions

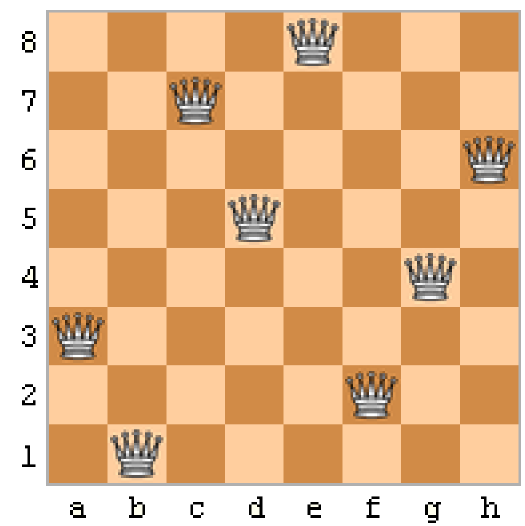




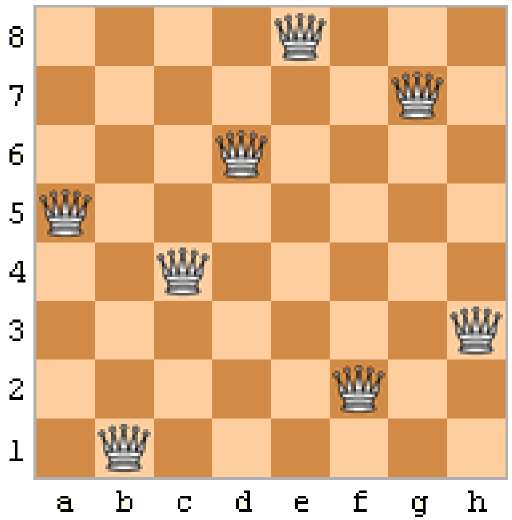


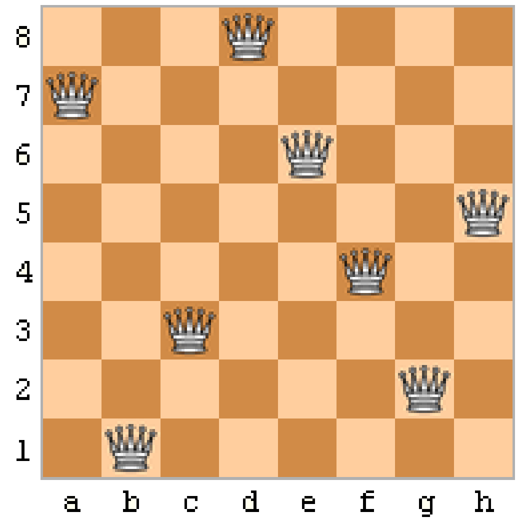


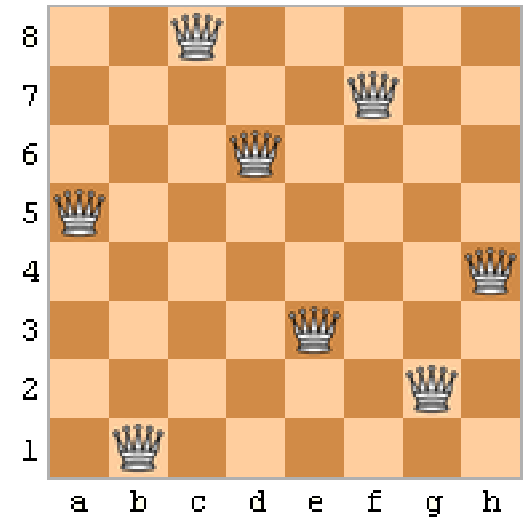


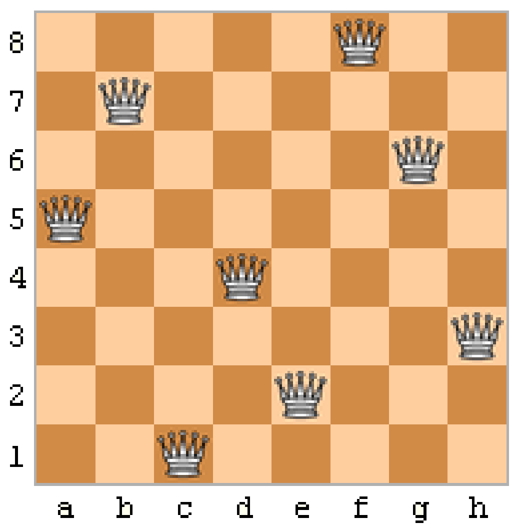


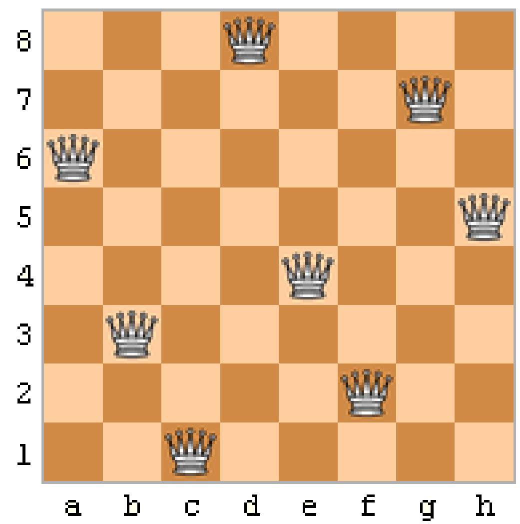
Les 12 solutions

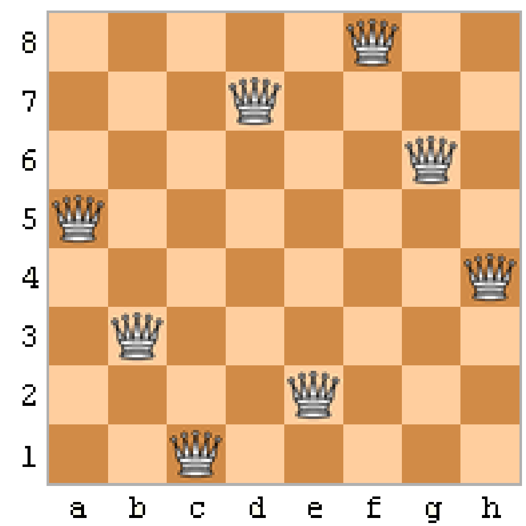






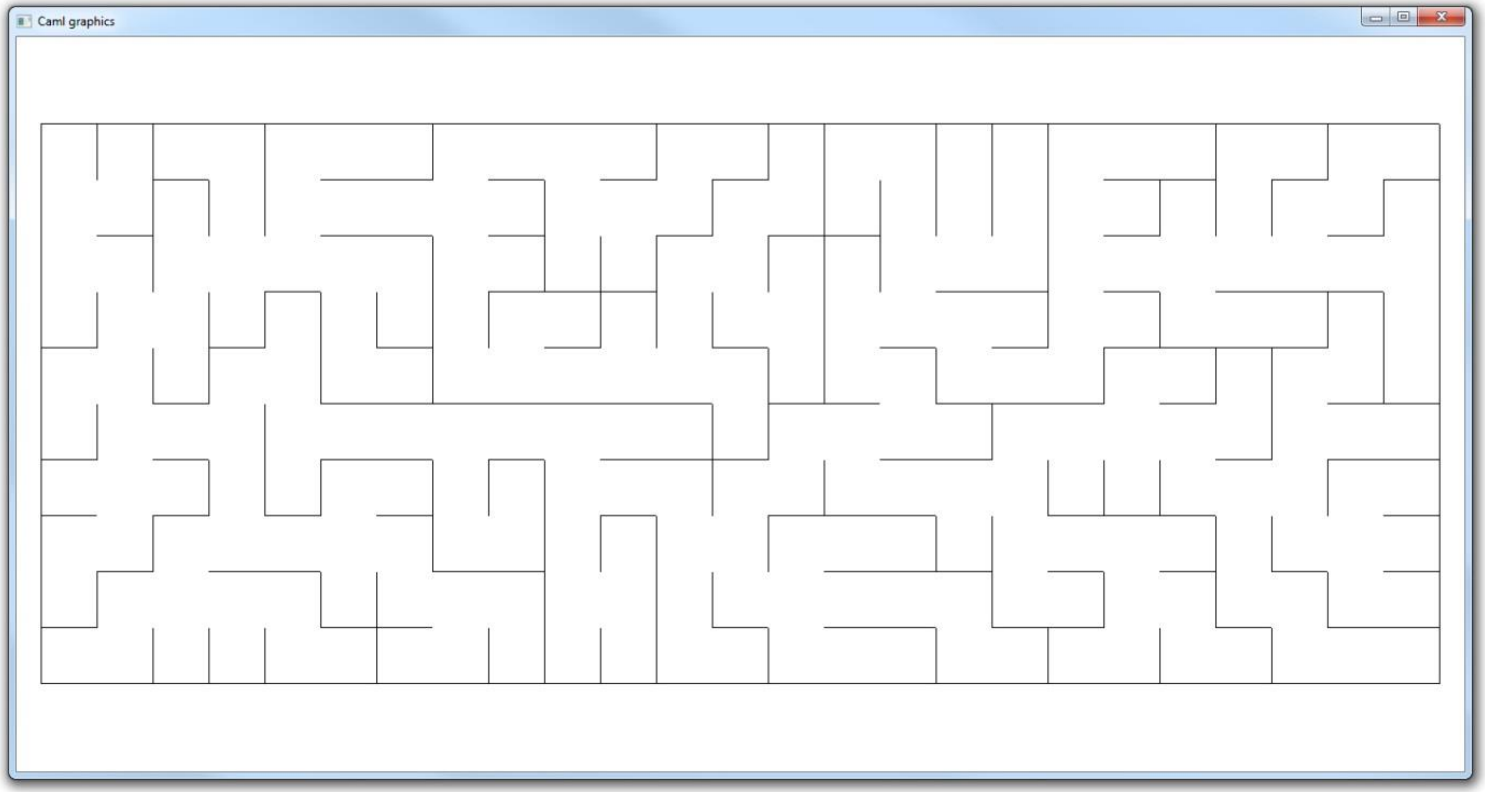






Labyrinthes

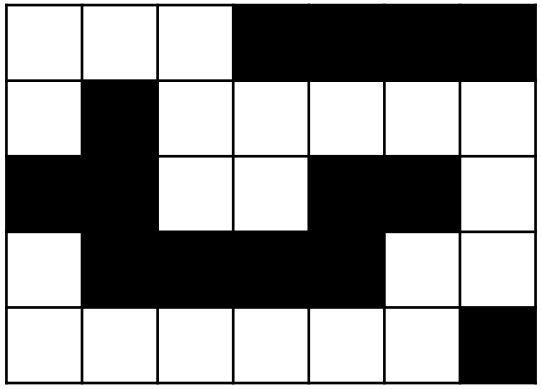
NOTRE OBJECTIF

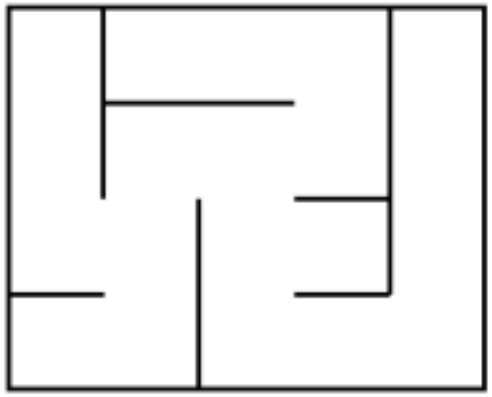


Choix de la représentation

- **Question** : *Quelle structure de données pourrait-on utiliser pour représenter un labyrinthe ?*
  - ➔ **Réponse** : un tableau de tableaux

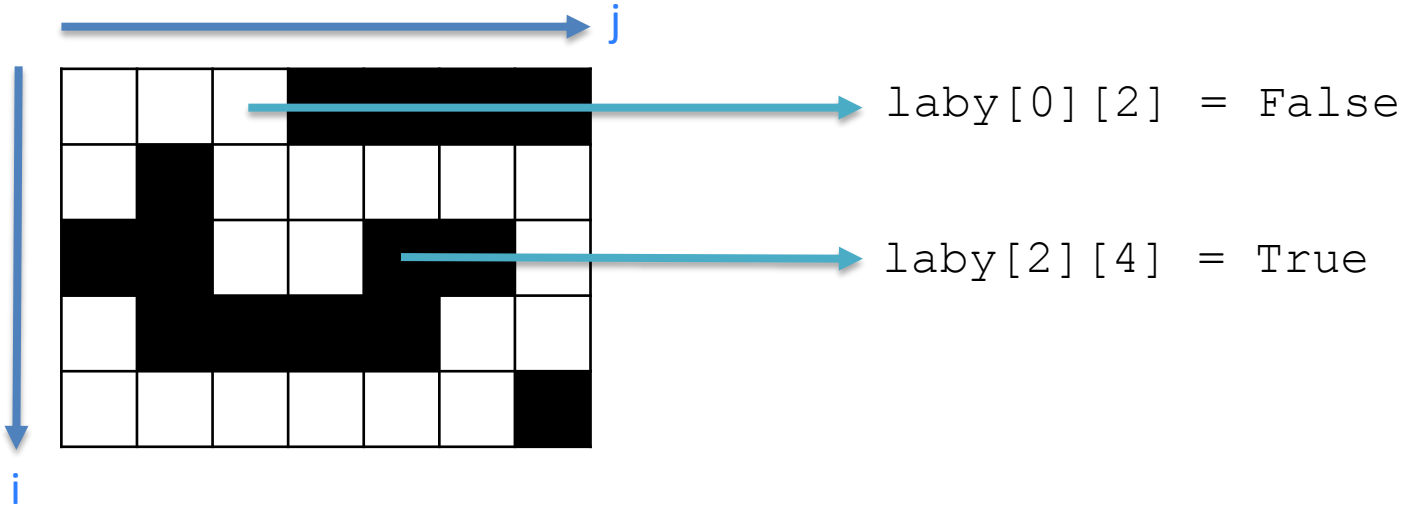
- **Question subsidiaire** : *Comment représenter chaque case ?*
  - ➔ **Première réponse possible** : avec un booléen
  - ➔ **Deuxième réponse possible** : avec plusieurs booléen





Labyrinthes simples : avec un seul booléen

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



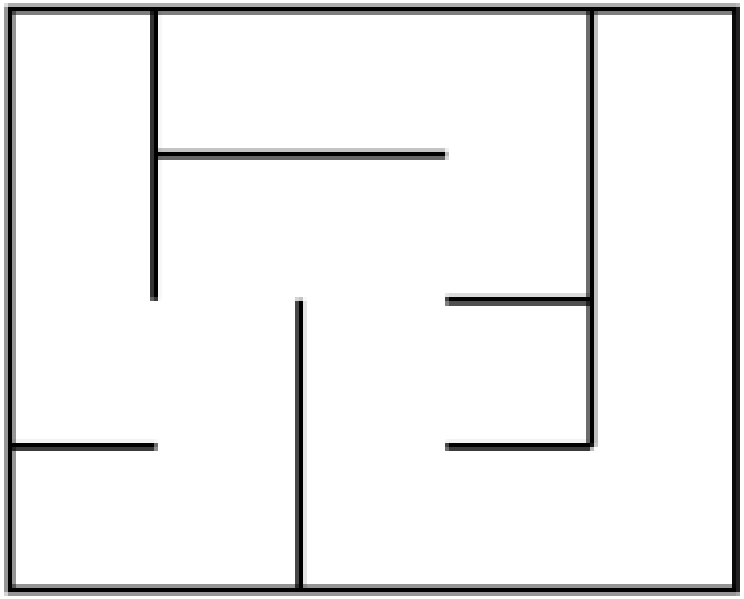
- Convention proposée :

  `laby[i][j] = 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

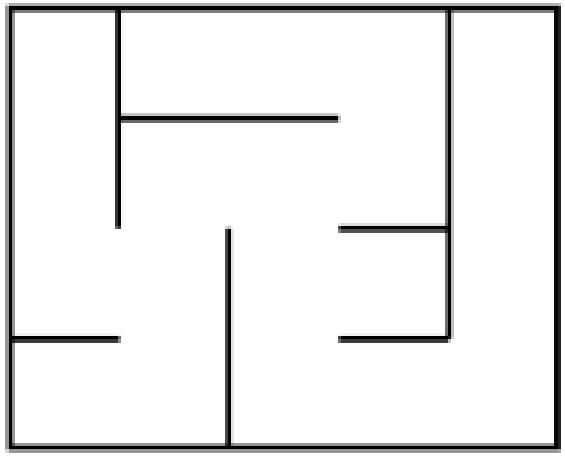
- Principe : *Les murs sont situés entre les cases*



Labyrinthe avancés : Choix de la représentation

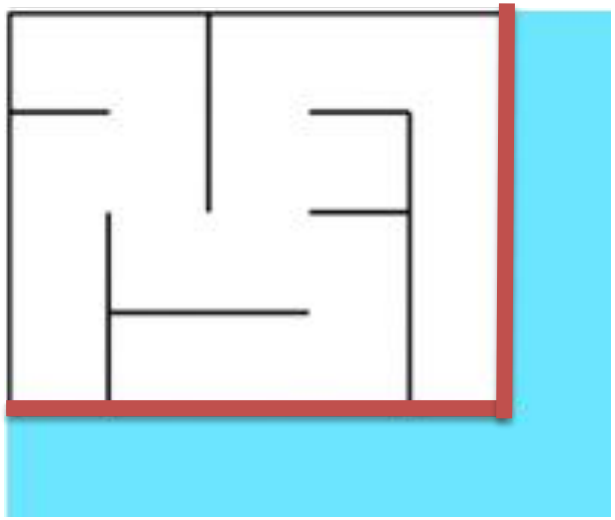
- Question : *Comment représenter un tel labyrinthe ?*

- 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 : Les huit dames

Mini-TD

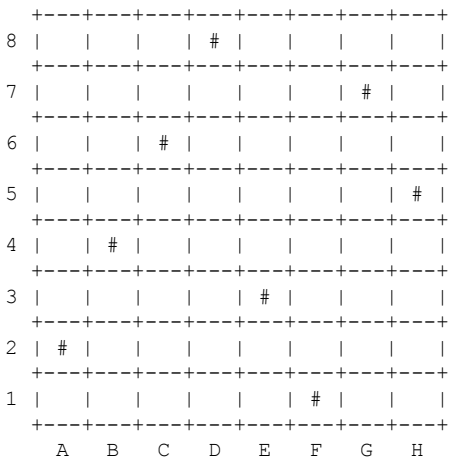
- **Question 1.a.** Ecrire une fonction qui renvoie un échiquier vide, sur lequel on pourra poser jusqu'à huit dames (à vous de déterminer la représentation la plus adaptée).

- **Question 1.b.** Ajouter manuellement 8 dames sur cet échiquier.

- **Question 2.** Ecrire une fonction qui prend en argument un échiquier sur lequel on a posé huit dames, et le trace sous forme textuelle.

- **Question 3.** Ecrire une fonction qui prend en argument deux couples de coordonnées et qui détermine si ces deux positions sont "en prise"

- **Question 4.** Ecrire une fonction qui détermine les solutions du problème des huit dames



PROCHAINE SÉANCE

Jeudi 8 octobre

[TD] LABYRINTHES