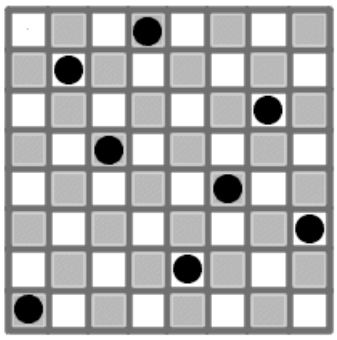
Das Acht-Damenproblem

Es handelt sich hier um ein Schachproblem, das erstmals von M. Bezzel 1845 in einer Schachzeitung veröffentlicht wurde, aber vorerst unbeachtet blieb. Erst als die Aufgabe 1850 vom blinden Schachexperten Dr. Nauk erneut zur Diskussion gestellt wurde, fand sie breites Echo. Als Nauk 3 Monate später alle 92 Lösungen publizierte, hatte der berühmte C.F. Gauss erst 72 Lösungen gefunden!

Die Aufgabe:
Man finde alle Möglichkeiten, 8 Damen auf einem Schachbrett so zu platzieren, dass keine eine andere bedroht.
(Eine Dame kann auf allen Feldern waagrecht, senkrecht und diagonal schlagen)

Eine Lösung ist etwa:



Wir werden die Aufgabe aber verallgemeinern:

Es sind alle Lösungen zu suchen, wie man n Damen auf einem n mal n großen Schachbrett platzieren kann, ohne dass sie einander bedrohen.

Als Lösungsstrategie bietet sich hier das **Backtracking** an.
1.) Wir beginnen mit dem leeren Brett
2.) Die Platzierung wird schrittweise immer um eine Dame erweitert, es entsteht eine hierarchische Baumstruktur
3.) Wird bei der Positionierung einer weiteren Dame festgestellt, dass sie auf einem bedrohten Feld steht, werden mit dieser Platzierung keine weiteren Schritte mehr unternommen (Backtracking zur letzten gültigen Platzierung)
4.) Wenn n Damen zu platzieren versucht wurden, sind wir fertig

Wie beschreiben wir das Brett:
Da n Damen auf dem nxn Brett sein müssen, ist klar, dass in jeder Zeile und in jeder Spalte genau eine Dame stehen muss. (Zwei würden einander schon bedrohen)
Arbeiten wir zeilenweise. Dann müssen wir angeben, an welcher Position die Dame in dieser Zeile stehen soll. Dazu ist eine Numerierung der waagrechten Positionen mit 0,1,2,3,4,...n-1 geeignet, die die Koordinate der Dame angibt. Und diese Werte stecken wir in eine Liste, die die Zeilen des Bretts von unten nach oben darstellt.

Obige Lösung entspräche somit der Liste [0,4,7,5,2,7,1,3]

Nebenbei: Das n-Damenproblem scheint (es gibt aber noch keinen Beweis) ein NP-schwieriges Problem (not polynomial time solvable) zu sein. Das heißt, dass es keinen Lösungsalgorithmus gibt, der die Aufgabe in polynomialer Zeit lösen kann. Die Rechenzeit steigt also exponentiell mit der Zahl n, und dieses Verhalten dürfte wahrscheinlich nicht verbesserbar sein. Beachte: Bereits auf dem n=8 – Schachbrett gibt es $8^8 = 16777216$ mögliche Stellungen von 8 Damen! Ein geniales Programm, das alles in einem Schlag löst, können wir nicht erwarten.

Welche Funktionen benötigen wir?

- eine Funktion, die feststellt, ob ein gewisses Feld von einer der bisher aufgestellten Damen bedroht wird. Wir nennen sie **`bedroht`** und übergeben ihr die zu testende Position der Dame und die Liste **`teilloesung`** der bereits aufgestellten Damen.
- eine Funktion die für jede Zeile des Bretts versuchsweise eine Dame aufstellt. Ist dies nach Spielregel möglich, wird die nächste Dame nach den selben Regeln (Rekursion!) zu setzen versucht. Ist es nicht möglich, werden dieser Ort und damit auch alle mit dieser Positionierung folgenden Platzierungen sofort verworfen. Wir nennen diese Funktion **`solve`** und übergeben ihr die Zahl der zu setzenden Damen. Damit ist dann ja auch die Brettgröße klar. Weiters nehmen wir für die Rekursion die Liste der bereits gesetzten Damen mit. Diese soll **`loesung`** heißen, weil sie sich dieser immer mehr annähert. Bange Frage: wird die Rekursion enden? Ja, denn bei jedem Aufruf ist eine Dame weniger übrig. Sind alle positioniert, sind wir fertig.

Die Funktion '`bedroht`':
Wann ist die Position *pos* bedroht? *pos* selbst ist ja eine Zahl von 0 bis n-1 und gibt die waagrechte Koordinate einer Position an. Die vertikale Koordinate ist die über der letzten bearbeiteten Zeile, deren Nummer wir aus der Länge der übergebenen Teilliste erfahren können.
Eine Test-Dame ist bedroht, wenn eine bereits gesetzte Dame

1.) senkrecht drunter steht. (Drüber kann nicht passieren, da wir zeilenweise nach oben aufbauen.) Was heißt senkrecht drunter: *pos* hat den selben Zahlenwert wie eine bereits gesetzte Dame.
2.) in der selben Zeile steht. Das kann aber nie passieren, da wir immer genau eine Dame pro Zeile setzen.
3.) auf der selben Diagonale links oder rechts hinunter steht. Das wird schwierig und verlangt Vektorrechnung:
Die Koordinaten der Testdame sind xtest = pos und ytest = len(teilloesung), also die neue aktuelle y-Koordinate, da die Länge der Liste immer um eins über dem Index des letzten Elements steht,. (Eine Liste der Länge N hat die Bestandteile L[0], L[1],...L[N-1])
Die bereits gesetzten Damen haben die Koordinaten xqueen = teilloesung[queen] und y=queen, wenn wir mit queen den Index in der Liste bezeichnen. Wann liegen nun beide auf der selben Diagonale? Der Unterschied ihrer x-Werte ist bis aufs Vorzeichen gleich dem Unterschied ihrer y-Werte (Mathematiker sprechen von Anstieg 1 oder -1).

```
def bedroht(pos,teilloesung):
    # falls das Brett noch leer war
    if teilloesung==[]: return 0

    # vertikale Koordinate ist bereits vergeben
    if pos in teilloesung: return 1

    # diagonal
    xtest = pos
    ytest = len(teilloesung)
    # teste alle bereits gesetzten Damen
    for queen in range(len(teilloesung)):
        xqueen = teilloesung[queen]
        yqueen = queen
        if abs(ytest-yqueen)==abs(xtest-xqueen):
            return 1

    return 0
```

eine einzelne Dame ist nie bedroht

eine andere Dame steht genau darunter und bedroht

Übereinstimmung bedeutet Bedrohung

keine Bedrohung vorgefunden

Testen wir die Funktion sofort!
Zuerst eine senkrechte Bedrohung, dann eine diagonale, dann keine.

```
>>> bedroht(2,[3,1,2,0])
1
>>> bedroht(4,[3,1,2,0])
1
>>> bedroht(8,[3,1,2,0])
0
```

Die Funktion 'solve':
Sie probiert alle Möglichkeiten durch und verfolgt nur richtige Teillösungen weiter. Da wir mehrere Lösungen erwarten und die Funktion rekursiv sein wird, nehmen wir eine globale Variable antwort, in der wir alle gefundenen Lösungen protokollieren.
Aufrufparameter: Die Anzahl `n` der zu setzenden Damen, was der Brettgröße gleichkommt. Die (bisherige Teil-)Lösung `solution` müssen wir auch übergeben.

```
def solve(n,solution):
    global antwort

    # wenn alle n Damen in der Liste sind
    if len(solution)==n:
        antwort.append(solution)
        return

    # jede moegliche x-Position
    for x in range(n):
        if not bedroht(x,solution):
            solve(n,solution+[x])

    return
```

wenn wir fertig sind

sonst:
jede waagrechte Koordinate testen, nur eine unbedrohte Position wird weiter bearbeitet

Jetzt noch eine freundliche Startfunktion, und wir können die Lösung unserer gestellten Aufgabe angehen!

```
def queens(anzahl):
    global antwort
    antwort = []

    solve(anzahl,[])

    for i in range(len(antwort)):
        print '%3d: '%(i+1),antwort[i]
```

anfangs ist die Lösung leer

die Ausgabe der Zählung soll mit 1 beginnen

Ein Beispiellauf:

```
>>> queens(5)
  1:  [0, 2, 4, 1, 3]
  2:  [0, 3, 1, 4, 2]
  3:  [1, 3, 0, 2, 4]
  4:  [1, 4, 2, 0, 3]
  5:  [2, 0, 3, 1, 4]
  6:  [2, 4, 1, 3, 0]
  7:  [3, 0, 2, 4, 1]
  8:  [3, 1, 4, 2, 0]
  9:  [4, 1, 3, 0, 2]
 10:  [4, 2, 0, 3, 1]
```

Das 5-Damenproblem hat also 10 unterschiedliche Lösungen.

Für Neugierige, die nicht warten wollen: Die Lösungsanzahl des n-Damenproblems:

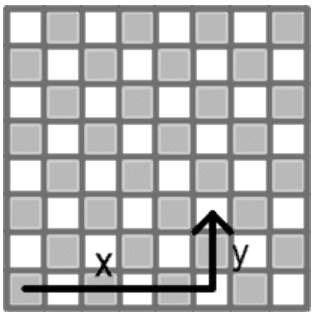
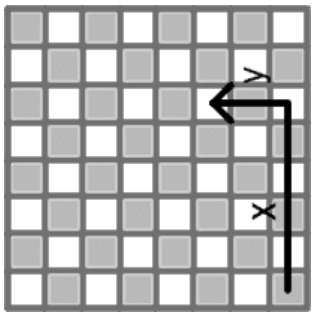
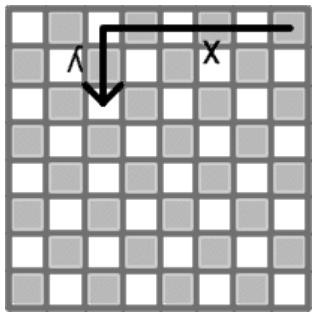
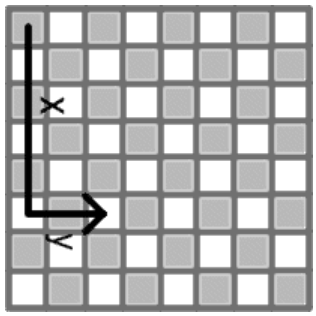
n	Anzahl
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200

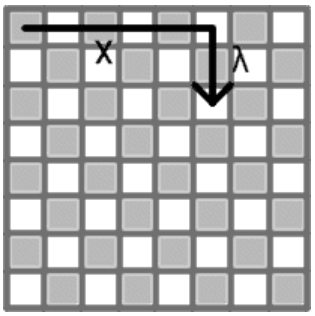
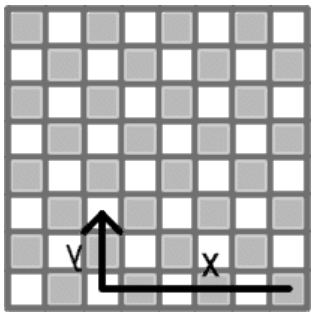
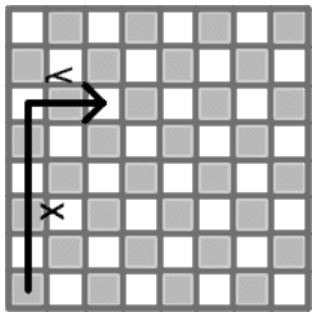
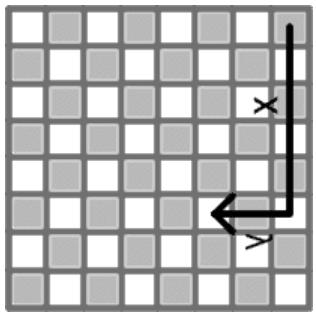
(Für diese Experimente solltest Du die print-Anweisung in queens aber besser entfernen, und nur die Länge der Antwort zurückgeben. Und beachte: eine Dame mehr bedeutet ab n=8 nicht 'etwas' mehr Rechenzeit, sondern 'deutlich viel' mehr!)

Perfektion:

Wenn Du die beiden Lösungen von queens(4) berechnest und vergleichst wirst Du feststellen, dass sie zwar unterschiedlich sind, aber nicht wirklich anders – die eine ist das Spiegelbild der anderen! Wir können nun fragen, wie viele Lösungen das n-Damenproblem besitzt, wenn wir alle durch Symmetrieoperationen entstandenen Lösungen weglassen.

Etwas Mathe-Nachhilfe: Das Brettquadrat hat für uns 8 gleichwertige Ansichten. (Wenn das Brett richtig liegen soll – mit dem weißen Eckfeld rechts - sind es nur 4). Wie können wir die Transformation des 'nullten' Zustandes in die anderen beschreiben? Wir sehen uns das in Koordinaten an.

Original	**Drehung 90°**	**Drehung 180°**	**Drehung 270°**
			
x / y	n-y / x	n-x / n-y	y / n-x

Spiegelung x-Achse	**Spiegelung y-Achse**	**Spiegel 1. Diagonale**	**Spiegel 2. Diagonale**
			
x / n-y	n-x / y	y / x	n-y / n-x

Nun braucht man Funktionen, die eine entsprechend transformierte Lösung zurückliefern. Ist `l` die zu bearbeitende Liste, so brauchen wir nur `x/y` durch `x/l[x]` zu ersetzen, und haben die passenden Koordinaten. Die umgerechneten Werte tragen wir in die vorbereitete Rückgabeliste ein.

```
def mirrorx(l):
    lnew = l[:]
    lnew.reverse()
    return lnew

def mirrory(l):
    n = len(l)-1
    lnew = [0]*len(l)
    for x in range(len(l)):
        lnew[x]=n-l[x]
    return lnew

def turn180(l):
    n = len(l)-1
    lnew = [0]*len(l)
    for x in range(len(l)):
        lnew[n-x]=n-l[x]
    return lnew

def mirrord(l):
    n = len(l)-1
    lnew = [0]*len(l)
    for x in range(len(l)):
        lnew[l[x]]=x
    return lnew

def mirrord1(l):
    n = len(l)-1
    lnew = [0]*len(l)
    for x in range(len(l)):
        lnew[n-l[x]]=n-x
    return lnew

def turn90(l):
    n = len(l)-1
    lnew = [0]*len(l)
    for x in range(len(l)):
        lnew[n-l[x]]=x
    return lnew

def turn270(l):
    n = len(l)-1
    lnew = [0]*len(l)
    for x in range(len(l)):
        lnew[l[x]]=n-x
    return lnew
```

das geht mit Listenmethode am schnellsten

Brettgröße n ermitteln
Rückgabeliste vorbereiten
jedes Element der alten Liste
in die neue eintragen

In der Hauptfunktion `distinctqueens` fügen wir `antwort[0]` in `dist` ein entfernen alle symmetrischen Bilder von `antwort[0]` aus `antwort`, falls sie dort noch vorhanden sind (manche Lösungen bleiben gleich, wenn man sie dreht oder spiegelt – dann sind sie bereits vorher entfernt. Denke an die zwei Lösungen von queens(4)) . Wir machen so weiter, bis `antwort` leer ist.

```
def distinctqueens(anzahl):
    global antwort
    antwort = []
    solve(anzahl,[])

    dist = []

    while antwort != []:
        solution,antwort = antwort[0],antwort[1:]
        dist.append(solution)

        for f in [mirrorx,mirrory,turn180, \
                  mirrord,mirrord1,turn90,turn270]:
            symm = f(solution)
            if symm in antwort:
                antwort.remove(symm)

    for i in range(len(dist)):
        print "%3d:"%(i+1),dist[i]
```

Lösungen bestimmen

unterschiedliche Lösungen

die erste ist sicher neu

alle symmetrischen Bilder erzeugen

wenn vorhanden, entfernen

Bildschirmausgabe

Ein Testlauf:

```
>>> distinctqueens(8)
  1: [0, 4, 7, 5, 2, 6, 1, 3]
  2: [0, 5, 7, 2, 6, 3, 1, 4]
  3: [1, 3, 5, 7, 2, 0, 6, 4]
  4: [1, 4, 6, 0, 2, 7, 5, 3]
  5: [1, 4, 6, 3, 0, 7, 5, 2]
  6: [1, 5, 0, 6, 3, 7, 2, 4]
  7: [1, 5, 7, 2, 0, 3, 6, 4]
  8: [1, 6, 2, 5, 7, 4, 0, 3]
  9: [1, 6, 4, 7, 0, 3, 5, 2]
 10: [2, 4, 1, 7, 0, 6, 3, 5]
 11: [2, 4, 7, 3, 0, 6, 1, 5]
 12: [2, 5, 1, 4, 7, 0, 6, 3]
```

Du erkennst vielleicht die erste Lösung wieder – sie ist am Anfang des Textes abgebildet!

Jetzt kennen wir die Lösung genau:

n	Anzahl
1	1
2	0
3	0
4	1
5	2
6	1
7	6
8	12
9	46
10	92

Das 8-Damenproblem hat 92 verschiedene Lösungen. 12 davon sind tatsächlich unterschiedlich, die übrigen 80 gehen durch Drehung oder Spiegelung aus ihnen hervor.

Alternative Version:
Man könnte auch einen Lösungsweg versuchen: Belege alle Felder des Brettes mit der Markierung 'sicher'. Setze eine Dame in die erste Zeile. Alle bedrohten Felder des Brettes werden mit 'bedroht' gekennzeichnet. Beim Setzen der weiteren Damen braucht man nun nicht wie oben alle möglichen horizontalen Koordinaten durchzuprobieren, sondern nur diejenigen, deren Markierung 'sicher' lautet. Vorteil: viel weniger Tests, seltenere Rückzüge im Backtracking (nur dann, wenn eine gesamte Zeile 'bedroht' ist und keine Dame aufnehmen kann). Nachteil: bei jedem Rekursionsschritt muss die gesamte Brettstruktur mit übergeben werden (Schachcomputer müssen dies tun und verwenden trickreich komprimierte (spart Speicherplatz) Feldinformationen, die allerdings immer wieder aus- und eingepackt werden müssen (kostet Rechenzeit).

Weitere Aufgabenstellungen:

- Was ist die kleinste Zahl von Damen auf einem nxn-Brett, sodass alle Felder des Bretts entweder besetzt oder bedroht sind? (n=8 Antwort=5)
- Was ist die kleinste Zahl von Damen auf einem nxn-Brett, sodass alle Felder des Bretts besetzt oder bedroht sind, wobei keine Dame eine andere Dame bedrohen darf?
- Und wieviele unterschiedliche Stellungen gibt es dabei jeweils?

- Wie viele andere Schachfiguren lassen sich derart positionieren?
- Und jewqeils in wie vielen Möglichkeiten?

Leider sind alle diese Fragestellungen vermutlich NP-schwere Probleme mit exponentieller Laufzeit. Aber vielleicht findest ja Du einen polynomialen Algorithmus!