

```
/* Sudoku-Solver*/
```

```
/* Eingabe des Sudokufeldes als Liste, freie Stellen werden als Nullen angegeben */
```

```
L: [ [2,6,7,0,0,0,0,0,0],  
      [0,0,0,7,8,9,0,0,0],  
      [0,0,0,0,0,0,3,5,0],  
      [5,0,0,0,2,4,0,0,0],  
      [1,2,0,0,0,8,0,0,0],  
      [0,0,0,0,0,0,9,0,3],  
      [0,0,4,0,6,0,0,9,2],  
      [0,0,0,0,9,3,0,0,8],  
      [0,7,1,0,0,0,0,0,4] ] $
```

```
/* Umwandlung der Liste in eine Matrix */
```

```
matrix: apply('matrix,L)$
```

```
/* Hilfsfunktionen: testen, ob sich eine bestimmte Zahl in der jeweiligen Zeile, Spalte oder Submatrix befindet */
```

```
test_zeile(zeile,zahl):= member(zahl,zeile) $  
test_spalte(spalte,zahl):= member(zahl,spalte) $  
test_sub(sub,zahl):= member(zahl, flatten(args(sub)))$
```

```
/* Hilfsfunktionen: lesen die einzelnen Zeilen und Spalten(durchnummeriert mit den Zahlen von 1 bis 9) und die Submatrizen(anhand der Zeilen und Spalten) aus der Matrix heraus */
```

```
zeile(n,m):= args(m)[n]$  
spalte(n,m):= zeile(n, transpose(m)) $  
sub(z,s,m):= block ( [su],  
  if z<=3 then (  
    if s<=3 then su:submatrix(4,5,6,7,8,9,m,4,5,6,7,8,9)  
    else if s<=6 then su:submatrix(4,5,6,7,8,9,m,1,2,3,7,8,9)  
    else su:s:submatrix(4,5,6,7,8,9,m,1,2,3,4,5,6) )  
  else if z<=6 then (  
    if s<=3 then su:submatrix(1,2,3,7,8,9,m,4,5,6,7,8,9)  
    else if s<=6 then su:submatrix(1,2,3,7,8,9,m,1,2,3,7,8,9)  
    else su:submatrix(1,2,3,7,8,9,m,1,2,3,4,5,6) )  
  else (  
    if s<=3 then su:submatrix(1,2,3,4,5,6,m,4,5,6,7,8,9)  
    else if s<=6 then su:submatrix(1,2,3,4,5,6,m,1,2,3,7,8,9)  
    else su:submatrix(1,2,3,4,5,6,m,1,2,3,4,5,6) ),  
  su )$
```

```

/* testet für ein bestimmtes "Kästchen" des Sudokufeldes welche Zahlen möglich sind und gibt diese in einer
Liste zurück */
test(z,s,m):= block([N,t,n,L],
  L:[],
  N:[1,2,3,4,5,6,7,8,9],
  for n in N do (
    if test3(n,z,s,m) then (
      L: cons(n,L)
    )
  ),
  L )$
/* Hilfsfunktion für test: Testet, ob ausgeschlossen werden kann, dass eine Zahl n in der jeweiligen Zeile,
Spalte und Submatrix noch nicht vorhanden ist */
test3(n,z,s,m):= not test_zeile(zeile(z,m), n)
               and not test_spalte(spalte(s,m), n)
               and not test_sub(sub(z,s,m), n)$

/* erstellt eine leere Liste für das gelöste Sudokufeld */
define_variable(loesung,[],any)$
/* erstellt eine Variable stop, die später als Abbruchbedingung dient */
define_variable(stop,false,any)$

/* setzt stop und Lösung wieder auf den ursprünglichen Wert und ruft die Funktion solver2 auf*/
solver(m):= (loesung:[], stop:false, solver2(m,1,1))$
/* Überprüft jede Stelle in der Matrix durch Durchzählen der Zeilen und Spalten. An jeder dieser freien
Stellen wird für jede mögliche Zahl eine neue Matrix erstellt und die Funktion mit der neuen Matrix
aufgerufen. So wird jede Möglichkeit ausprobiert und das Sudoku rekursiv gelöst. Sobald die letzte Stelle
erreicht ist (bei z=9 und s>9), bricht die Funktion ab und gibt die fertige Matrix als Lösung zurück*/
solver2(m,z,s):= block( [c],
  if ( z=9 and s>9) then ( stop:true, loesung:m )
  else if not stop then (
    if ( s>9 ) then ( s:1, z:z+1 ),
    if m[z,s]#0 then solver2(m,z,s+1)
    else for i in test(z,s,m) do(
      c: copymatrix(m),
      c[z,s]: i,
      solver2(c,z,s+1) ) ),
  loesung )$

compile(solver,solver2,sub,test,test3,test_zeile,test_spalte,test_sub,zeile,spalte)$

```