Sudoku = Number Place (CUI - Python)

S.Matoike

目次

第 1 章 Python による Sudoku 2
1.0.1 盤面を表示する 2
1.0.2 数値を置けるか判定する 3
1.0.3 問題を解く 4

参考文献 5

第 1 章

Python による Sudoku

Sudoku は、アメリカのデルマガジンズ社が 1979 年以降に Number Place の名前で販売していた数字パズルを、株式会社ニコリの創業者・社長である鍛治真起氏が「数独」と命名し、1986 年に月刊ニコリストで紹介した。2005 年には Sudoku の名で英国で大流行し、世界へ広まった。（英辞郎より）

このプログラムは、短い簡易な作りになっているのが魅力的な所です

1.0.1　盤面を表示する

まず、board リストに盤面を定義し、盤面の印刷を行う関数 print_board() を定義します

ソースコード 1.1　board

```
board = [[5,3,0,0,7,0,0,0,0],\
         [6,0,0,1,9,5,0,0,0],\
         [0,9,8,0,0,0,0,6,0],\
         [8,0,0,0,6,0,0,0,3],\
         [4,0,0,8,0,3,0,0,1],\
         [7,0,0,0,2,0,0,0,6],\
         [0,6,0,0,0,0,2,8,0],\
         [0,0,0,4,1,9,0,0,5],\
         [0,0,0,0,8,0,0,7,9]]

def print_board():
    global board
    for y in range(9):
        for x in range(9):
            print(' ',end='')
            if x in [2,5]:
                print(board[y][x], end=' |')
            else:
                print(board[y][x], end=' ')
        if y in [2,5]:
            print('\n---------|---------|--------')
        else:
            print()
```

print_board() をそのまま呼び出して、動作を確認してみます

```
# 動作確認
```

```
print_board()
```

```
  5  3  0 | 0  7  0 | 0  0  0
  6  0  0 | 1  9  5 | 0  0  0
  0  9  8 | 0  0  0 | 0  6  0
 ---------|---------|--------
  8  0  0 | 0  6  0 | 0  0  3
  4  0  0 | 8  0  3 | 0  0  1
  7  0  0 | 0  2  0 | 0  0  6
 ---------|---------|--------
  0  6  0 | 0  0  0 | 2  8  0
  0  0  0 | 4  1  9 | 0  0  5
  0  0  0 | 0  8  0 | 0  7  9
```

1.0.2　数値を置けるか判定する

次に、引数 y と x で指定されたスロットに、第三引数で受け取った n を置く事ができるか否かを判定する関数 possible() を定義します

(1)　縦一列の中に、n と同じ数字があったら置けませんので、False を返します

(2)　横一行の中に、n と同じ数字があったら置けませんので、False を返します

(3)　3×3 の枠の中に、n と同じ数字があったら置けませんので、False を返します

上記（1)、(2)、(3）の何れでもないなら、True を返します

ソースコード 1.2　possible

```
1  def possible(y,x,n):
2      global board
3      for i in range(0,9):
4          if board[y][i] == n:
5              return False
6      for i in range(0,9):
7          if board[i][x] == n:
8              return False
9      x0 = (x//3)*3
10     y0 = (y//3)*3
11     for i in range(0,3):
12         for j in range(0,3):
13             if board[y0+i][x0+j] == n:
14                 return False
15     return True
```

5 行 5 列目 (board リストの 0 行目や 0 列目から数え始めるので、引数は x=y=4) の空きスロットに値を指定して、possible() の動作を確認してみます

```
# 動作確認
print( possible(4,4,4) )
print( possible(4,4,5) )
```

1.0.3　問題を解く

最後に、問題を解く関数 solve() を定義します

y 行 x 列目のスロットに着目して、そこが 0 ならば空きスロットですから、そのスロットに、1 から 10 までの数字を順に指定して、possible() を呼びだしてはチェックしていきます

もし、possible() 関数が True を返したら、それは一つの候補ですので、引き続き solve() を繰り返して空きスロットを埋めていきます

solve() が return で None を返したときは、選ばれた候補は使えなかったという事ですので、盤面のスロットを空（0）に戻しています

全ての空欄が埋まったなら、print_board() を呼んで結果（答え）を印刷しています

ソースコード 1.3　solve

```
1  def solve():
2      global board
3      for y in range(9):
4          for x in range(9):
5              if board[y][x] == 0:
6                  for n in range(1,10):
7                      if possible(y,x,n):
8                          board[y][x] = n
9                          solve()
10                         board[y][x] = 0
11                 return
12     print_board()
```

solve() 関数の中から、自分自身である solve() 関数を呼び出す方法、再帰呼び出し、を使うことによってプログラムを短く記述できています

```
# 動作確認
solve()
```

```
 5  3  4 | 6  7  8 | 9  1  2
 6  7  2 | 1  9  5 | 3  4  8
 1  9  8 | 3  4  2 | 5  6  7
---------|---------|--------
 8  5  9 | 7  6  1 | 4  2  3
 4  2  6 | 8  5  3 | 7  9  1
 7  1  3 | 9  2  4 | 8  5  6
---------|---------|--------
 9  6  1 | 5  3  7 | 2  8  4
 2  8  7 | 4  1  9 | 6  3  5
 3  4  5 | 2  8  6 | 1  7  9
```

参考文献

[1]