

Python Sudoku Program – Overview

- This document and the code it describes can be found here:
 - <https://github.com/sgarrow/soduko>
- Create the “first order” candidate list
- Prune the candidate list based on:
 - Non-hidden and hidden “tuples” (pair, triplet, ...) for a given house (row, col, square)
 - X-Wing
 - Pointing Pairs
 - Y-Wing
- Fill in solution cells based on:
 - Only a single value exists in the “candidates”
 - Analysis of the histogram of the candidates for a given “house” (row, col or square)

10,000 Foot Software Pseudo-Code (Actual ~1500 Lines of Code)

```
main()    # Main program
```

```
puzzle = readPuzzle
```

```
canLst = Create1stOrderCanLst(puzzle)
```

```
loop
```

```
    puzzle = fillSingleCandidates(puzzle, canLst)
```

```
    puzzle = fillHistogram(puzzle, canLst)
```

```
    canLst = Create1stOrderCanLst(puzzle)
```

```
    canLst = pruneCanLst(canLst)
```

```
    if solved(puzzle):
```

```
        break loop
```

```
pruneCanLst(canLst)    # Function
```

```
loop
```

```
    totNumPrunned = 0
```

```
    loop
```

```
        numPrunned = pruneTuples()
```

```
        totNumPrunned += numPrunned
```

```
        if numPrunned = 0:
```

```
            break loop
```

```
    loop
```

```
        numPrunned = pruneXwing()
```

```
        totNumPrunned += numPrunned
```

```
        if numPrunned = 0:
```

```
            break loop
```

```
    loop
```

```
        numPrunned = prunePointiungPairs()
```

```
        totNumPrunned += numPrunned
```

```
        if numPrunned = 0:
```

```
            break loop
```

```
    loop
```

```
        numPrunned = pruneYwing()
```

```
        totNumPrunned += numPrunned
```

```
        if numPrunned = 0:
```

```
            break loop
```

```
if totNumPrunned = 0:
```

```
    break loop
```

```
return prunnedCanLst
```

Create First Order Candidate List

----- PUZZLE -----										----- CANDIDATES -----									
c0	c1	c2	c3	c4	c5	c6	c7	c8		c0	c1	c2	c3	c4	c5	c6	c7	c8	
					5	9			r0	1 3	123	23					3	123	r0
									r0	4 7	7	467	67	4 7			4 8	4	r0
8	9		3					6	r1			2			1	12			r1
									r1			4 7		4 7 4	45	45			r1
			8	2	9	7			r2	1 3	1 3	3					3	1 3	r2
									r2	4	5	4 6					45	45	r2
	4			1	7		2	9	r3										r3
									r3	3		3				3			r3
									r3			8	5			56			r3
2		9	4	3	6	8		7	r4		1						5		r4
									r4										r4
5	6		9	8			1		r5			3			2	3		3	r5
									r5			7				4		4	r5
									r5										r5
		5	1	6	8				r6	3	23					23	3	23	r6
									r6	4 7	7					4	479	4	r6
9					3		6	8	r7		2	2				12			r7
									r7		7	4 7	57	457		45			r7
6		1	2						r8		3			45	4	453	453	453	r8
									r8		78			7 9			7 9		r8

Finding the “first order” candidates for cell (7,2):

In the PUZZLE:

Row 7 already has: 3,6,8,9

Col 2 already has: 1,5,9

Sqr 6 already has: 1,5,6,9

Union of row 7, col 2, sqr 6:

1, ,3, ,5,6, ,8,9

So, what’s left for CANDIDATES is:

2,4,7

A Word About Lists in Python

```
tst.py - C:\01-home\14-python\soduko\tst.py

simpleVariable = 3

simpleList = [1,2,3]

# puzzle looks like this
# c0 c1 c2
puzzle = [
    [ 1, 2, 3 ], # r0
    [ 4, 5, 6 ], # r1
    [ 7, 8, 9 ] # r2
]

# canidates looks like this
# c0      c1      c2
canidates = [
    [ [1], [3,5], [4,8] ],
    [ [2,5], [6], [8,9] ],
    [ [7,8,9], [5,8], [1] ]
]

print(simpleVariable, '\n')
print(simpleList, '\n')
print(puzzle, '\n')
print(canidates, '\n')

print(puzzle[0][2], '\n')
print(canidates[2][0], '\n')
print(canidates[2][0][1], '\n')
```

```
Windows PowerShell
PS C:\01-home\14-python\soduko> .\tst.py
3
[1, 2, 3]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[[1], [3, 5], [4, 8]], [[2, 5], [6], [8, 9]], [[7, 8, 9], [5, 8], [1]]]
3
[7, 8, 9]
8
PS C:\01-home\14-python\soduko>
```

The point of all this is:

The “puzzle” is a 2-dimensional array (a list of lists)

The “candidates” are a 3-dimensional array (a list of lists of lists)

Fill Cells With Only One Candidate

Filling solution cells that have only 1 candidate

c0	c1	c2	c3	c4	c5	c6	c7	c8	
1	23	23						123	r0
4		4		4				4	r0
7	7	7		7					r0
		2				2			r1
		4		4		45	4		r1
		7		7					r1
1							3	1 3	r2
4							4	4	r2
									r3
									r4
						3		3	r5
						4		4	r5
		7							r5
	23					23		23	r6
4						4		4	r6
7	7								r6
	2	2							r7
		4		5					r7
	7	7	7	7					r7
						3	3	3	r8
						5		5	r8
							7		r8

Placing 4 at 1,7 Placing 7 at 5,2 Placing 7 at 7,3

Fill Cells Via (Row) Histogram Analysis

Filling solution cells thru Row Hist Analysis									
c0	c1	c2	c3	c4	c5	c6	c7	c8	
1	23	23						123	r0 [(0, 4), (1, 2), (2, 3), (3, 3), (4, 4), (7, 4)]
4		4		4				4	r0
7	7	7		7					r0
		2				2			r1 [(0, 5), (2, 2), (4, 4), (5, 1), (7, 2)]
		4		4		45	4		r1 Placing 5 at 1,6
		7		7					r1
1							3	1 3	r2 [(0, 6), (1, 2), (3, 2), (4, 3)]
4							4	4	r2
									r3 [(0, 9)]
									r4 [(0, 9)]
						3		3	r5 [(0, 6), (3, 2), (4, 2), (7, 1)]
						4		4	r5
		7							r5
	23					23		23	r6 [(0, 5), (2, 3), (3, 3), (4, 3), (7, 2)]
4						4		4	r6
7	7								r6
	2	2							r7 [(0, 5), (2, 2), (4, 1), (5, 1), (7, 4)]
	4			5					r7 Placing 4 at 7,2
	7	7		7					r7
						3	3	3	r8 [(0, 6), (3, 3), (5, 2), (7, 1)]
						5		5	r8 Placing 7 at 8,7
						7			r8

The histogram shows that only one cell on row 1 can contain a 5 – so that cell must be a 5.

The histogram shows that only one cell on row 7 can contain a 4 – so that cell must be a 4.

The histogram shows that only one cell on row 8 can contain a 7 – so that cell must be a 7.

Fill Cells Via (Col) Histogram Analysis

Filling solution cells thru Col Hist Analysis

	c0	c1	c2	c3	c4	c5	c6	c7	c8	
++	++	++	++	++	++	++	++	++	++	++
	1	23	23						123	r0
	4		4		4				4	r0
	7	7	7		7					r0
++	++	++	++	++	++	++	++	++	++	++
			2				2			r1
			4		4		45	4		r1
			7		7					r1
++	++	++	++	++	++	++	++	++	++	++
	1							3	1 3	r2
	4						4		4	r2
++	++	++	++	++	++	++	++	++	++	++
										r3
++	++	++	++	++	++	++	++	++	++	++
										r4
++	++	++	++	++	++	++	++	++	++	++
							3		3	r5
							4		4	r5
			7							r5
++	++	++	++	++	++	++	++	++	++	++
		23					23		23	r6
	4						4		4	r6
	7	7								r6
++	++	++	++	++	++	++	++	++	++	++
		2	2							r7
			4		5					r7
		7	7	7	7					r7
++	++	++	++	++	++	++	++	++	++	++
							3	3	3	r8
							5		5	r8
							7			r8

```

[(0, 6), (1, 2), (4, 3), (7, 2)]
[(0, 6), (2, 3), (3, 2), (7, 3)]
[(0, 5), (2, 3), (3, 1), (4, 3), (7, 4)]
Placing 3 at 0,2
[(0, 8), (7, 1)]
[(0, 6), (4, 2), (5, 1), (7, 3)]
Placing 5 at 7,4
[(0, 9)]
[(0, 5), (2, 2), (3, 3), (4, 3), (5, 2)]
[(0, 6), (3, 2), (4, 2), (7, 1)]
[(0, 4), (1, 2), (2, 2), (3, 5), (4, 4), (5, 1)]
Placing 5 at 8,8
    
```

Same explanation as with row histogram analysis except this time look at the col histograms instead.

Fill Cells Via (Sqr) Histogram Analysis

Filling solution cells thru Sqr Hist Analysis

c0	c1	c2	c3	c4	c5	c6	c7	c8	
1	23	23						123	r0
4		4		4				4	r0
7	7	7		7					r0
		2				2			r1
		4		4		45	4		r1
		7		7					r1
1							3	1 3	r2
4							4	4	r2
									r3
									r4
						3		3	r5
						4		4	r5
		7							r5
4	23					23		23	r6
7	7					4		4	r6
		2	2						r7
		4		5					r7
	7	7	7	7					r7
						3	3	3	r8
						5		5	r8
						7			r8

```

[(0, 4), (1, 2), (2, 3), (3, 2), (4, 4), (7, 4)]
[(0, 7), (4, 2), (7, 2)]
[(0, 4), (1, 2), (2, 2), (3, 3), (4, 5), (5, 1)]
[(0, 8), (7, 1)]
[(0, 9)]
[(0, 7), (3, 2), (4, 2)]
[(0, 5), (2, 3), (3, 1), (4, 2), (7, 4)]
Placing 3 at 6,1
[(0, 7), (5, 1), (7, 2)]
[(0, 4), (2, 2), (3, 5), (4, 2), (5, 2), (7, 1)]
    
```

Same explanation as with row histogram analysis except this time look at the square histograms instead.

Pruning Naked Pairs

c0	c1	c2	c3	c4	c5	c6	c7	c8	
4 6						4 6		4 6	r0
						8			r0
		3 2					3 2		r1
	4 6	4				4 6	6 4 6		r1
							9 9		r1
2	3					1 3	12		r2
5	5								r2
						7 7 7			r2
1						1 1			r3
5 6						6 6 5 6			r3
9						9 9			r3
12		2				1 1			r4
4 5 4 5 4 5						4		4 5	r4
8 8						7 7 7			r4
4 6 4 6 4									r5
9		9							r5
	9	7 9							r6
4 5 4 5 4 5						6 6			r7
8 8 7						7 7			r7
									r8

```
{'row': 0, 'tripIdxs': (0, 8), 'tripVals': {4, 6}}
remove {4, 6} from (0,6)
```

Two cells on row 0 contain (4,6) and only (4,6). We don't know which one is going to be a 4 or which one is going to be a 6, but it doesn't matter. One of them is going to be a 4 and the other is going to be a 6 – that's for sure.

What it means is that 4 or 6 cannot be a candidate for any other cell on this row.

You can apply a similar technique to cols and squares to eliminate candidates within the col or square, respectively.

```
{'row': 7, 'tripIdxs': (6, 7), 'tripVals': {6, 7}}
remove {7} from (7,2)
```

Pruning Naked Triples

c0	c1	c2	c3	c4	c5	c6	c7	c8	
123	123	23				1 3	12	23	r0
89	89	8							r0
123	123	23	23	2			12	23	r1
6	4	4	6				4	6	r1
7	9	9	7	9	9		7	7	r1
23	23	23	23	2			2		r2
6	4	4	6				4	6	r2
7	8	7	8	8			7		r2
123	123	23	523	526	23	1 3	12		r3
89	589	5	689	89	789	7	9	789	r3
		23	23	2	23	1 3		23	r4
		8	89	89	789	7	9	78	r4
23		523	523		523	3	2		r5
89		8	89		89	9	89		r5
	42	426	126		42	4	6	4	r6
	8	78	89		89	7	9	789	r6
6	4			56	456	4			r7
78	8			8	8	8			r7
23	23			2	2				r8
6	4			456	4	4	6	4	r8
8	8			89	89	9	89	8	r8

```
{'row': 0, 'tripIdxs': (6, 7, 8), 'tripVals': {1, 2, 3}}
remove {1, 2, 3} from (0,0)
remove {1, 2, 3} from (0,1)
remove {2, 3} from (0,2)
```

Three cells on row 0 contain (combined) only three numbers (1,2,3). Three cells, only three numbers – that’s the key. We don’t know which one is which, but, again, it doesn’t matter – one is going to be 1 one a 2 and one a 3, that’s for sure.

What it means is that 1 or 2 or 3 cannot be a candidate for any other cell on this row.

Generally, this is easier to spot when all three square contain all 3 numbers (1,2,3) but they don’t have to.

You can apply a similar technique to cols and squares to eliminate candidates within the col or square, respectively.

Pruning Hidden Pairs

	c0	c1	c2	c3	c4	c5	c6	c7	c8	
++	+	+	+	+	+	+	+	+	+	++
		12				1		12	2	r0
		8						78	78	r0
++	+	+	+	+	+	+	+	+	+	++
	12	12	1 3			1		23 123	23	r1
	6	5	6	6		6		5 5	5	r1
	78	8	78	8		9		8 789	78	r1
++	+	+	+	+	+	+	+	+	+	++
	1	1	1 3					3 1 3		r2
	6	5	6	6				5 5		r2
	8	8	8	8				8 89		r2
++	+	+	+	+	+	+	+	+	+	++
			1	23		23		23	23	r3
				4		4				r3
			8	8				8	8	r3
++	+	+	+	+	+	+	+	+	+	++
	2	2				2 3		2 3		r4
	4	4				4 5		8 5		r4
	8	8			8					r4
++	+	+	+	+	+	+	+	+	+	++
						2		2	2	r5
					6	456		45	5	r5
			8		8			8	8	r5
++	+	+	+	+	+	+	+	+	+	++
	4 6			4 6	6 4 6			56	5	r6
	78			7	9 9			78	78	r6
++	+	+	+	+	+	+	+	+	+	++
	1	1		23 1				23	23	r7
	4 6 4			4 6	6				6	r7
	7			7				7		r7
++	+	+	+	+	+	+	+	+	+	++
	1	1	1	23		123		23		r8
	6		6	6		6				r8
	78	89 78	7			9		8	78	r8
++	+	+	+	+	+	+	+	+	+	++

```
{'row': 4, 'tripIdxs': (5, 7), 'tripVals': [3, 5]}
remove {2, 4} from (4,5)
remove {8, 2} from (4,7)
```

Ok, hang on things are about to get weird.

On row 4 there are only two places where 3 and 5 can exist. You'd see them as a naked pair, if they weren't hidden by extra numbers.

Because 3 and 5 can only exist in two of those cells (**no other cells will accept either of them**), that means they must be in those two cells, leaving no room for any other. Even though you don't know which is which, you can remove all other candidates from the within those two cells.

Note that with naked pairs candidates outside the pair's cells are removed whereas with hidden pairs candidates inside the pair's cells are removed.

Pruning Hidden Triples

	c0	c1	c2	c3	c4	c5	c6	c7	c8	
++	++	++	++	++	++	++	++	++	++	++
	123	123	23				1 3	12	23	r0
		5	5							r0
	89	89	8							r0
++	++	++	++	++	++	++	++	++	++	++
	123	123	23	23	2			12	23	r1
	6	4	4 6					4 6		r1
	7 9	9 7		9	9			7	7	r1
++	++	++	++	++	++	++	++	++	++	++
	23	23	23	23	2			2		r2
	6	4	4 6					4 6		r2
	78	8	78	8	8			7		r2
++	++	++	++	++	++	++	++	++	++	++
	123	123	23	23	2	23	1 3	12		r3
		5	5	56	56					r3
	89	89	8	89	89	789	7 9	789		r3
++	++	++	++	++	++	++	++	++	++	++
			23	23	2	23	1 3		23	r4
			8	89	89	789	7 9		78	r4
++	++	++	++	++	++	++	++	++	++	++
	23		23	23		23	3	2		r5
			5	5		4				r5
	89		8	89		89	9	89		r5
++	++	++	++	++	++	++	++	++	++	++
		2	2	12		2				r6
		4	4 6	6		4	4 6	4 6		r6
		8	78	89		89	7 9	789	78	r6
++	++	++	++	++	++	++	++	++	++	++
	67	4		56	56	4				r7
	8	8		8	48	8				r7
++	++	++	++	++	++	++	++	++	++	++
	23	23			2	2				r8
	6	4			456	4	4 6	4 6	5	r8
	8	8			89	89	9	89	8	r8
++	++	++	++	++	++	++	++	++	++	++

```
{'row': 7, 'tripIdxs': (0, 3, 4), 'tripVals': [5, 6, 7]}
  remove {8} from (7,0)
  remove {8} from (7,3)
  remove {4, 8} from (7,4)
```

The algorithm for hidden triplets leverages off the algorithm for naked triplets in the same way that hidden pairs leverages off the naked pairs.

Again, the naked/hidden pair/triple algorithms can be applied to a row, col or square.

BTW the same techniques work for quads (4 cells 4 values), etc.

Have fun. Hard by hand easier by computer especially if you have a single routine that just accepts “N” as a parameter!! Which I do.

Pruning X-Wing

```

{'A_rows': [1, 5], 'B_cols': [3, 7], 'C_val': 7}
remove 7 from (3,7)
remove 7 from (4,3)
remove 7 from (4,7)
remove 7 from (8,3)
remove 7 from (8,7)

```

Only two cells in row 1 can be a 7.
Only two cells in row 5 can be a 7.
 The cols that the 7's appear in in rows 1 and 5 are the same – they form an “X”.

As such, the 7's in cols 3 and 7 must appear on rows 1 and 5 – either at 1,3 and 5,7 or at 1,7 and 5,3.

Therefore any 7's that appear in cols 3 and 7 (and not on rows 1 and 5) can be eliminated.

Note: this is not a great example (but still valid) because cell 1,3 is a “single candidate”.

Pruning Pointing Pairs

	c0	c1	c2	c3	c4	c5	c6	c7	c8	
++-----++										
	23	2	2	23	23	2				r0
	6	56	4 6	6 4	45	45	4			r0
	789	789	7 9	9 7	9 7	9 7	9 7	789		r0
++-----++										
	2	2	12	12			79			r1
	75	79	759	479	9		45			r1
++-----++										
	63	62	62		23	2				r2
	78	789	7 9	7 9	7 9	7 9	789			r2
++-----++										
	78			289	749	749	749			r3
++-----++										
	36	4 6	451	1	41			416		r4
	7	7	7	7	7			7		r4
++-----++										
	3	2	2	1	2	1	2	1	3	r5
	6	6	4 6	45	4	45	456	4 6		r5
	78	78	78	789	89	7 9	7 9	7 9	7 9	r5
++-----++										
	56	16	56	1 6	136	13	413	436		r6
	78	789	789	89	89	9	7 9	7 9		r6
++-----++										
	1 6	6	126			1 3	63	16		r7
	89	89	89			9	9	9		r7
++-----++										
	1 6		1 6	1	1					r8
	7 9		9	9	9	7 9				r8
++-----++										

```
{ 'aRow': 5, 'bCols': [1, 2], 'cVal': 2 }
remove 2 from (5,3)
remove 2 from (5,4)
remove 2 from (5,5)
```

In square 3, only 2 cells can be a 2. Because they are on the same row (and because one of them is going to definitely be a 2) we can eliminate 2 as a candidate from other cells on that row.

Pruning Y Wings

c0	c1	c2	c3	c4	c5	c6	c7	c8		
			6	6	1		26	126	r0	
			9	89	9			8	r0	
1					1			1	r1	
			45		4			5	r1	
8								8	r1	
1				56			56	156	r2	
8				8				8	r2	
									r3	
			59	59				35	r4	
							75	75	r5	
	2					3		23	r6	
4	6							4	6	r6
			3	4	6			4	16	r7
						1				r7
4	6			3	6	4		2	23	r8
					9	9		7	7	r8

```
key      = 145
cord     = [[0, 3], [7, 3], [8, 5]]
sqrs    = [1, 7, 7]
vals     = [[6, 9], [4, 6], [4, 9]]
pIdx     = 1
Z        = 9
rmvIdx   = [(1,5), (8,3), (6,3), (0,5), (2,5)]
  remove 9 from (0,5)
```

Find 3 cells each of which has exactly 2 values and for which those values are of the form $[(X,Y), (X,Z), (Y,Z)]$.

The cells must be arranged such that one of the cells can “see” both or the other 2 but those other 2 cannot “see” each other. The first cell is called the “pivot” and the other two are called “wings”. Drawing lines from the pivot to each wing forms a “Y”.

The intersection of the cells seen by each of the two wings is called “the intersection”.

The value (there will only be one) that's shared between the wings can be eliminated from "the intersection".

Guessing

	c0	c1	c2	c3	c4	c5	c6	c7	c8	
++-----++										
		26	256	4 6	36	234	25	5		r0
		789	789	7		7	7 9	7 9	789	r0
++-----++										
	75	72	257							r1
++-----++										
			26	6		2				r2
			789	7		7	279		789	r2
++-----++										
							7 9	7 9		r3
++-----++										
			476			47			76	r4
++-----++										
	6	26	426	4		4	153	356	1 6	r5
	7	7	7	7 9		7 9	7 9	7 9	7 9	r5
++-----++										
	56	1 6	56		36	13		36		r6
	7	7 9	7 9			9		7 9		r6
++-----++										
		16	6				13	36	1 6	r7
		89	89				9	9	9	r7
++-----++										
		16		6	1	1				r8
		7 9		9		9	7 9			r8
++-----++										

```
length canidates - sqrs
[[[0, 5, 6], [3, 2, 4], [4, 3, 3]],
 [[2, 2, 3], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 5], [2, 0, 2], [3, 0, 3]],
 [[0, 0, 0], [0, 0, 0], [2, 2, 0]],
 [[0, 0, 3], [0, 0, 2], [0, 0, 2]],
 [[2, 3, 4], [3, 0, 3], [5, 5, 4]],
 [[3, 4, 4], [0, 2, 3], [0, 4, 0]],
 [[0, 4, 3], [0, 0, 0], [3, 3, 3]],
 [[0, 4, 0], [2, 0, 2], [3, 0, 0]]]
```

```
max length canidates by 3 cols
[[6, 4, 4],
 [3, 0, 0],
 [5, 2, 3],
 [0, 0, 2],
 [3, 2, 2],
 [4, 3, 5],
 [4, 3, 4],
 [4, 0, 3],
 [4, 2, 3]]
```

```
firstTryCoord
[[0, 5], [1, 2], [2, 6]]

canVals
[[2, 3, 4, 7], [2, 5, 7], [2, 7, 9]]

canValsLst
[[2, 2, 2], [2, 2, 7], [2, 2, 9],
 [2, 5, 2], [2, 5, 7], [2, 5, 9],
 [2, 7, 2], [2, 7, 7], [2, 7, 9],
 [3, 2, 2], [3, 2, 7], [3, 2, 9],
 [3, 5, 2], [3, 5, 7], [3, 5, 9],
 [3, 7, 2], [3, 7, 7], [3, 7, 9],
 [4, 2, 2], [4, 2, 7], [4, 2, 9],
 [4, 5, 2], [4, 5, 7], [4, 5, 9],
 [4, 7, 2], [4, 7, 7], [4, 7, 9],
 [7, 2, 2], [7, 2, 7], [7, 2, 9],
 [7, 5, 2], [7, 5, 7], [7, 5, 9],
 [7, 7, 2], [7, 7, 7], [7, 7, 9]]
```

Find three cells all in different rows, cols and squares but all in the same “row of squares”.