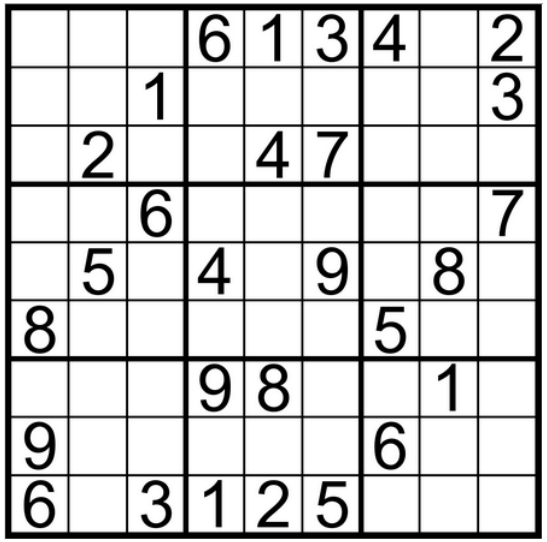
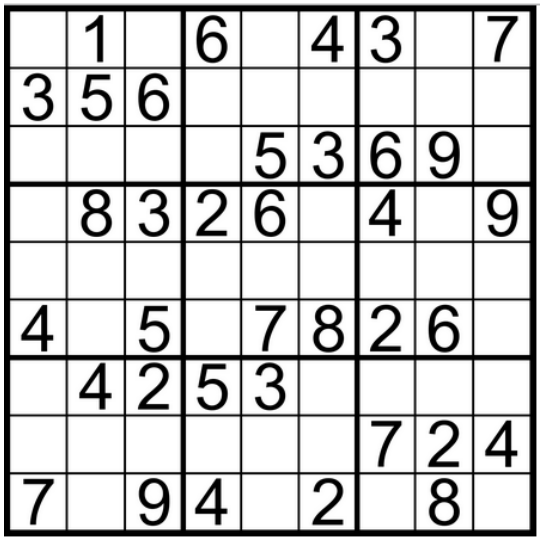
# Object Oriented Programming 31695

# Final Course Project

# SUDOKU



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



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

# Files Organization

- Download sudoku.zip from: http://www.samyzaf.com/braude/OOP/PROJECTS/sudoku.zip
- Unzip this file in drive C (or D), so your project will reside in: C:\sudoku (or D:\sudoku)
- You will find there all the files you need for the project
- You can also access individual files from: http://www.samyzaf.com/braude/OOP/PROJECTS/Sudoku
- Make sure to edit the **README.txt** file and enter all the required information (name, email, phones, etc.)
- After completing your project, you should zip this directory back to **sudoku.zip** and upload it to: http://www.samyzaf.com/braude/OOP/upload.html

# Submission Policy

- Deadline: June 07, 2014 (till midnight)
  - Upload site will be closed after this date!
- Work in pairs is OK (but not triples!)
- A 30 minutes project review will be held for each partner separately!
- Office slots for project reviews will be posted early June
- Use the scheduling tool to schedule a meeting: http://www.samyzaf.com/cgi-bin/appsched.cgi

# SuDoKu Puzzle Rules

- Information based on Project Euler: http://projecteuler.net
- The objective of a **SuDoKu** puzzle is to replace the blanks (or zeroes) in a 9x9 grid such that each row, column, and 3x3 box contains each of the digits 1 to 9
- The **Diagonal-SuDoKu** puzzle requires each **diagonal** contains all 1-9 digits
- Below is a textual display of a typical puzzle grid and its solution grid:

```
SUDOKU PUZZLE                   SOLUTION
+-------+-------+-------+       +-------+-------+-------+
| 0 0 3 | 0 2 0 | 6 0 0 |       | 4 8 3 | 9 2 1 | 6 5 7 |
| 9 0 0 | 3 0 5 | 0 0 1 |       | 9 6 7 | 3 4 5 | 8 2 1 |
| 0 0 1 | 8 0 6 | 4 0 0 |       | 2 5 1 | 8 7 6 | 4 9 3 |
+-------+-------+-------+       +-------+-------+-------+
| 0 0 8 | 1 0 2 | 9 0 0 |       | 5 4 8 | 1 3 2 | 9 7 6 |
| 7 0 0 | 0 0 0 | 0 0 8 |       | 7 2 9 | 5 6 4 | 1 3 8 |
| 0 0 6 | 7 0 8 | 2 0 0 |       | 1 3 6 | 7 9 8 | 2 4 5 |
+-------+-------+-------+       +-------+-------+-------+
| 0 0 2 | 6 0 9 | 5 0 0 |       | 3 7 2 | 6 8 9 | 5 1 4 |
| 8 0 0 | 2 0 3 | 0 0 9 |       | 8 1 4 | 2 5 3 | 7 6 9 |
| 0 0 5 | 0 1 0 | 3 0 0 |       | 6 9 5 | 4 1 7 | 3 8 2 |
+-------+-------+-------+       +-------+-------+-------+
```

# SuDoKu – Graphical View

- We will also be interested in drawing SuDoKu diagrams on top of a graphical canvas as shown here:

**SUDOKU PUZZLE**

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

**SOLUTION**

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

# SuDoKu - Solution

- A well constructed **SuDoku** puzzle has a **unique** solution and can be solved by logic, although it may be necessary to employ "**guess and test**" methods in order to eliminate options (there is much contested opinion over this).
- The complexity of the search determines the difficulty of the puzzle
- the example above is considered easy because it can be solved by straight forward direct deduction

http://www.mirror.co.uk/news/weird-news/worlds-hardest-sudoku-can-you-242294

Mirror NEWS

How we use cookies

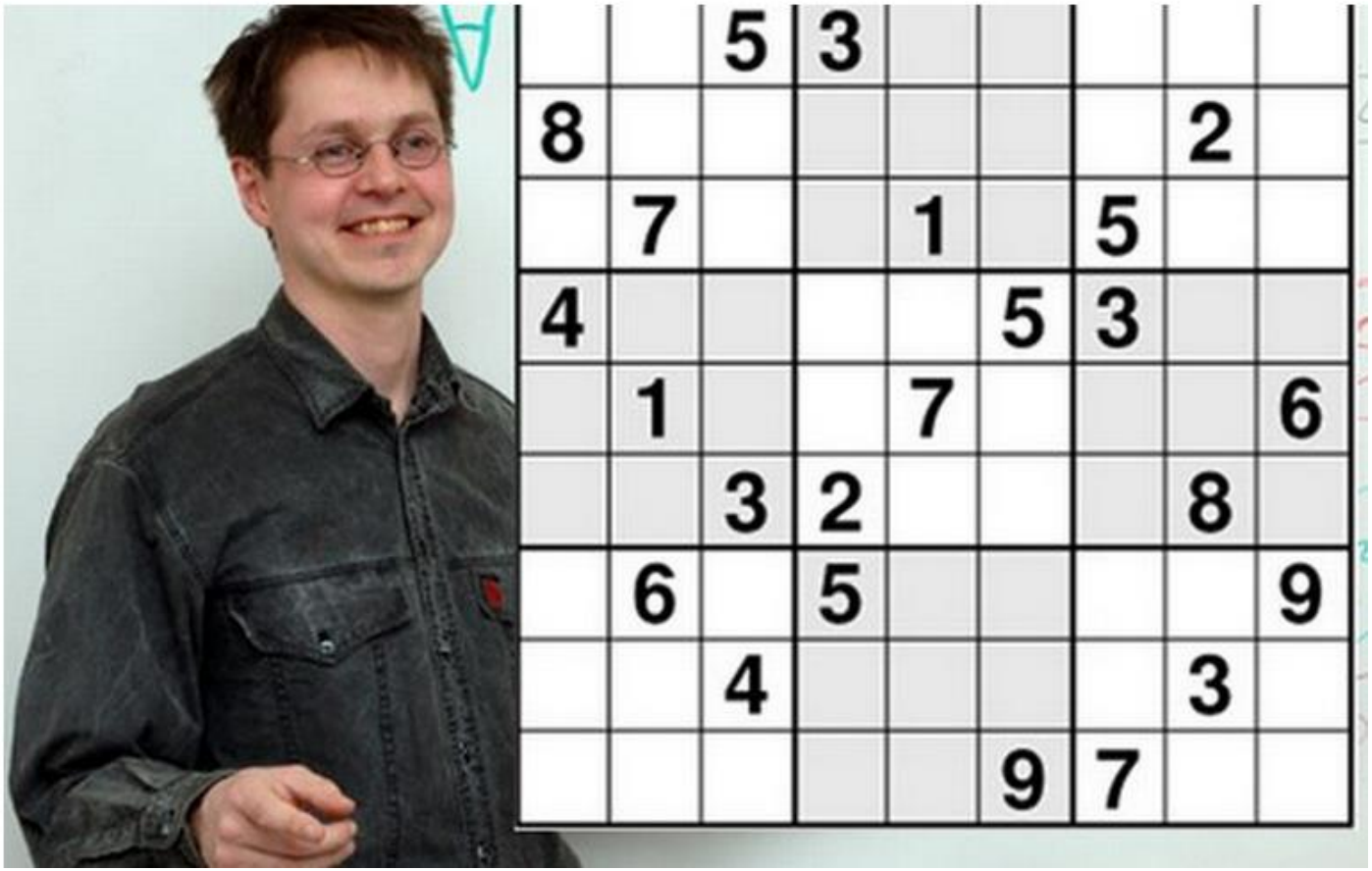
Real news, real entertainment . . . in real time

FRONT PAGE | NEWS | SPORT | 3AM | TV | LIFESTYLE | MONEY | PLAY | OPINION | Search

News › Weird News

# World's hardest sudoku: Can you solve Dr Arto Inkala's puzzle?

Could this be the toughest sudoku puzzle ever devised?



# USA Today Journal, June 11 2006

## Mathematician claims to have penned hardest sudoku

Updated 11/7/2006 10:11 AM ET

E-mail | Print | RSS

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

Enlarge

By Peter Ritmeester, www.PZZL.com

HELSINKI (AFP) — A Finnish mathematician on Monday claimed he had created the world's hardest sudoku puzzle, a brain-teaser which required three months' work and a billion combinations to produce.

"Al Escargot is the most difficult sudoku-puzzle known so far," the puzzle's 37-year-old creator and applied mathematician Arto Inkala told AFP.

"I called the puzzle Al Escargot, because it looks like a snail. Solving it is like an intellectual culinary pleasure. Al are my initials," Inkala added. According to a rating system published on the forum of www.sudoku.com, Al Escargot claims the top spot for sudoku's most baffling puzzles.

Escargot demands those tackling it to consider eight casual relationships simultaneously while the most complicated variants attempted by the general public only require people to think of one or two combinations at any one time, Inkala said.

**Can your program solve this one?**

# Sudoku Puzzle Databases

- The zip package you have downloaded contains several puzzle databases
- **easy10.txt**
  - These are 10 very easy puzzles you should use for tests
  - Your program should handle these puzzles very quickly! (so you don't have to wait to long for tests)
- **top160.txt**
  - A set of 160 very difficult puzzles
- **top2500.txt**
  - If you need more: database with 2500 puzzles
- **diag200.txt**
  - 200 diagonal Sudoku puzzles

# Your Mission

- You need to
  - Build classes: **Cell, Sudoku, Sudokiller, Partition**
  - Define several function for reading puzzles from files, finding all the possible solutions of a Sudoku puzzle, and printing solutions in a pretty format (textual and graphical!)
- Use the files:
  - **cell.py**
  - **sudoku.py**
  - **sudokiller.py**
  - **file_solver.py**
- Make sure to follow the directions there precisely, and complete the missing code so that they eventually work

# Graphics

- The graphics that we use for drawing our Sudoku boards are:
  graphics.py
  point.py
  line.py
  rectangle.py
- These files are contained in the sudoku.zip package, so you do not have to download them
- These files implement our basic graphical environment. Specifically, it defines a canvas window on which we can draw points, lines, rectangles, and other geometrical shapes
- There is no need to read or understand the code in these modules, you're only required to use it for drawing your Sudoku puzzles
- If you want to learn more about the Tkinter graphics programming, you may start with: http://www.tkdocs.com/tutorial/

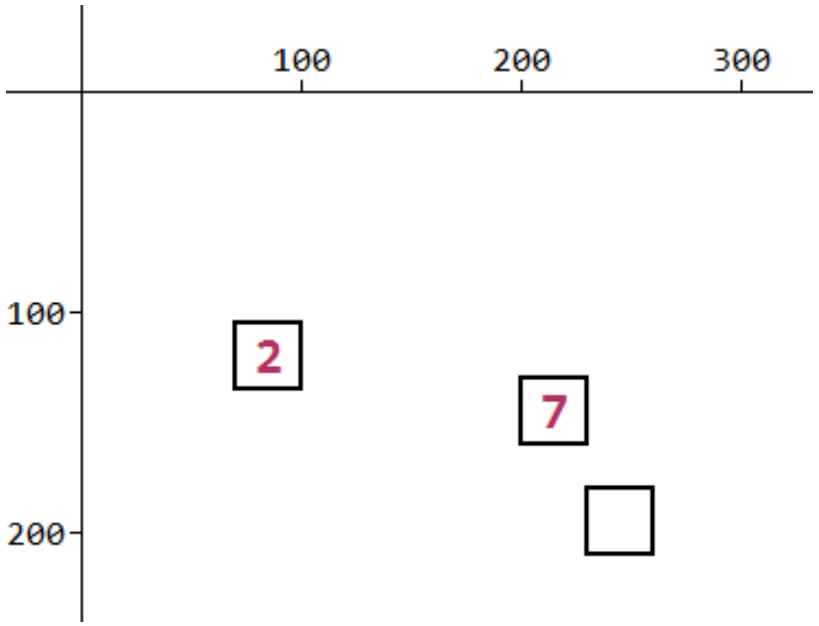
# Task 1: Cell Class

```
class Cell(Rectangle):
    size = 30
    font = "Consolas 12 bold"
    color = "Maroon"
    outline = "black"
    def __init__(self, data=0):
        Rectangle.__init__(self, 0, 0, self.size, self.size)
        self._data = data
        self.id = 0           # Canvas id

    def data(self, new_data=None):
        # Two uses:
        #   1. Change the Cell data to new_data:    c.data(7)
        #   2. Return the current data:             c.data()

    def draw(self):
        # Draw Cell object on the canvas
        # Graphics should consist of a Rectangle object and a text label
        # Label is: 1, 2, 3, ..., 9
        # An empty cell (0) has no label
        # Key command:
        #   canvas.create_text(p.x, p.y, text=label, font=self.font, fill=self.color)
        #   p is the Cell center Point
```

# Cell Class Test

- If you have designed your Cell class well, then it should pass the following test

```
def Cell_test():
    a = Cell(7)
    b = Cell(2)
    c = Cell(0)
    a.place(200,130)
    b.place(70,105)
    c.place(230,180)
    a.draw()
    b.draw()
    c.draw()
    print c.size
    show_canvas()
```



# Task 2: Sudoku Class

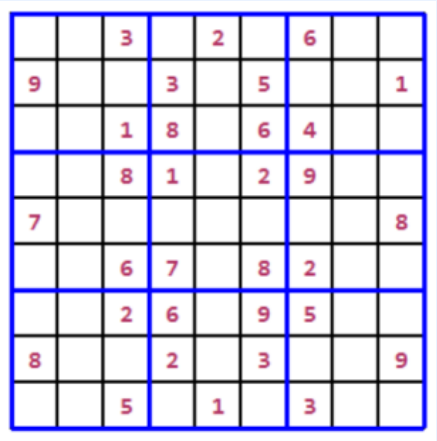
- Edit the file **sudoku.py** and complete the needed work there
- Please follow the class structure and make sure you find all the solutions to the puzzle (not the first one only!)

# Sudoku Constructor

```
puzzle1 = """
        0 0 3 0 2 0 6 0 0
        9 0 0 3 0 5 0 0 1
        0 0 1 8 0 6 4 0 0
        0 0 8 1 0 2 9 0 0
        7 0 0 0 0 0 0 0 8
        0 0 6 7 0 8 2 0 0
        0 0 2 6 0 9 5 0 0
        8 0 0 2 0 3 0 0 9
        0 0 5 0 1 0 3 0 0
    """

puzzle2 = "005080700 700204005 320000084 060105040 008000500 070803010 450000091 600508007 003010600"

puzzle3 = "4.....8.5.3.........7......2.....6.....5.4......1.......6.3.7.5..2.....1.9......"

s1 = Sudoku(puzzle1)
s2 = Sudoku(puzzle2)
s3 = Sudoku(puzzle3)

HINT: All we need to do is read 81 digits ! We simply skip everything else!
```

# Sudoku Constructor (more)

```
# Even this should work !
puzzle4 = """
      +-------+-------+-------+
      | 0 0 3 | 0 2 0 | 6 0 0 |
      | 9 0 0 | 3 0 5 | 0 0 1 |
      | 0 0 1 | 8 0 6 | 4 0 0 |
      +-------+-------+-------
      | 0 0 8 | 1 0 2 | 9 0 0 |
      | 7 0 0 | 0 0 0 | 0 0 8 |
      | 0 0 6 | 7 0 8 | 2 0 0 |
      +-------+-------+-------
      | 0 0 2 | 6 0 9 | 5 0 0 |
      | 8 0 0 | 2 0 3 | 0 0 9 |
      | 0 0 5 | 0 1 0 | 3 0 0 |
      +-------+-------+-------+
    """

s = Sudoku(puzzle4)

# After reading the input, we may do all kinds of methods:
s.draw()
print s.is_valid()
s.solve()
```



# Cell Storage

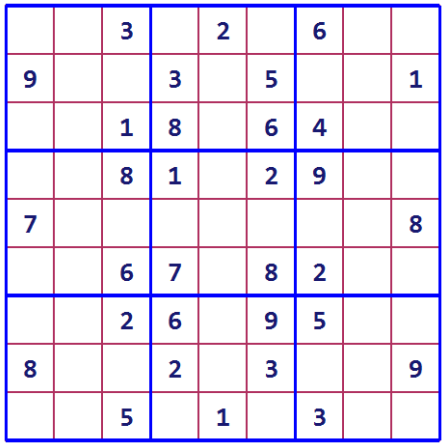
```
puzzle = """
    0 0 3 0 2 0 6 0 0
    9 0 0 3 0 5 0 0 1
    0 0 1 8 0 6 4 0 0
    0 0 8 1 0 2 9 0 0
    7 0 0 0 0 0 0 0 8
    0 0 6 7 0 8 2 0 0
    0 0 2 6 0 9 5 0 0
    8 0 0 2 0 3 0 0 9
    0 0 5 0 1 0 3 0 0
  """

# s consists of 81 ints stored in a dictionary: s.cell:
s = Sudoku(puzzle)
for i,j in s.cell:
    print s.cell[i,j]

for i,j in s.cell:
    s.cell[i,j] = 0
```

# Sudoku Display

- There are two methods for displaying a Sudoku object:
- `print s`

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

- `s.draw()`




Hint: The cells are Rectangle objects with a label in the middle.
The blue blocks can be 9 Rectangles or a simple draw_grid() – (8 Lines)

# row(), col(), block(), diagonals()

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

(puzzle4)

```
s = Sudoku(puzzle4)

s.row(2)      =>  [9, 0, 0, 3, 0, 5, 0, 0, 1]
s.row(5)      =>  [7, 0, 0, 0, 0, 0, 0, 0, 8]
s.col(2)      =>  [0, 0, 0, 0, 0, 0, 0, 0, 0]
s.col(5)      =>  [2, 0, 0, 0, 0, 0, 0, 0, 1]
s.block(4,5)  =>  [1, 0, 2, 0, 0, 0, 7, 0, 8]
s.diagonals() =>  ([0, 0, 1, 1, 0, 8, 5, 0, 0], [0, 0, 4, 2, 0, 7, 2, 0, 0])
```

# Task 3: Sudoku Solver

```
# This is the hardest part of the project !!!
# You will need to design a method:
def solve(self):
    # self is our Sudoku puzzle instance
    # All solutions should go into a list self.solutions
```

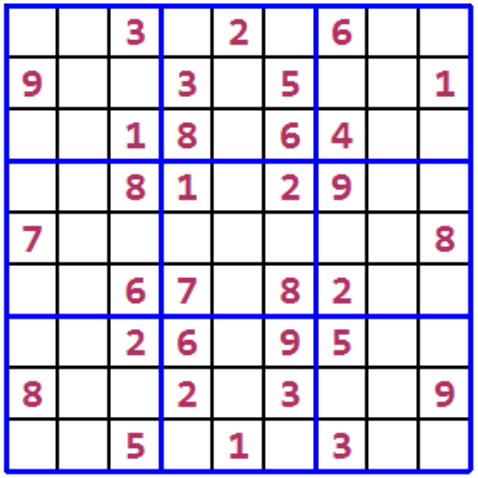
# Sudoku Algorithm: Part 1

- For any cell compute the list of **valid values**
- Example: Valid values of cell (4,1) = 3, 4, 5
  - Start by computing what cannot be in the cell
- A cell (i,j) is called a **singleton** if it has exactly **one valid value**
- Example: Cell (5,6) is a singleton!
  - What is his single valid value?
- A good strategy is to first fill the singleton cells!
- After solving singletons
  - The number of valid values in other cells will be reduced and will make the puzzle easier
  - After solving all singletons, you will get new singletons …
  - So you may need to repeat singleton resolution again …

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

# Sudoku Algorithm: Part 2

- An invalid cell is a cell that cannot be assigned any valid value
- After resolving all singletons, all remaining cells have **multiple valid values**
- The simplest strategy is to select a cell, compute his valid values, and try them all one by one
- Once you have selected a valid value and inserted it to the cell, you get a new puzzle with a smaller set of options!
- You may use recursion and send the new puzzle and then call yourself recursively !



# Task 4: Solve a Database of Puzzles

- Edit the file: **file_solver.py**
- You will find there the skeleton for the needed function
- You have to design a function **solve_sudoku_file(file)** which accepts a file name of Sudoku puzzles (like **top160.txt**) and solves all the puzzles in the file
- You have to write all the solutions in a new file (like: **top160.sol**)
- See next slide for the format in which the solutions should be written

# Task 4: Solve a Database of Puzzles

## top160.txt

File Edit Tools Syntax Buffers Window Help

```
000658000004000000120000000000096070003005000020800030019008003060000040000473000
530020900024030050009000000000010827000700000000098100000000000064000091020504300
047080001000000000000600700600003570000005000010060000280040000090100040000020690
963000000100008000000205000040800000010000700000030025700000030009020407000000900
000800009087300040600700000008500970000000000043007500000003000030001450400002001
000360000850000009040080000000068000000000170090045000105000604000090020000030000
020000000305062009068000300050000000000640802004700900003000001000006000170430000
023700006800060590900007000000409703070960020000000005004700000000020000800000000
020300000006008090830500000000200080709005000000060040000000100010004022007008090
000030090000020000105090000000000000010208040608050002007500000040100600300000406 0
000600040060300001004005077000008050008000006080000900020900004000032000097001000
000500000000000506970000020004802000250100030080030000000004070013050090020003100
904005000250600100310000008070009000400260000001470000700000002000300806040000090
000002000000070001700300090800700000020890600013006000090050824000008910000000000
032000005800300000904280001000400039000600050000010000020006708000004000095000060
050090000100000600000308000008040009514000000030000200000000004080006007700150060
```

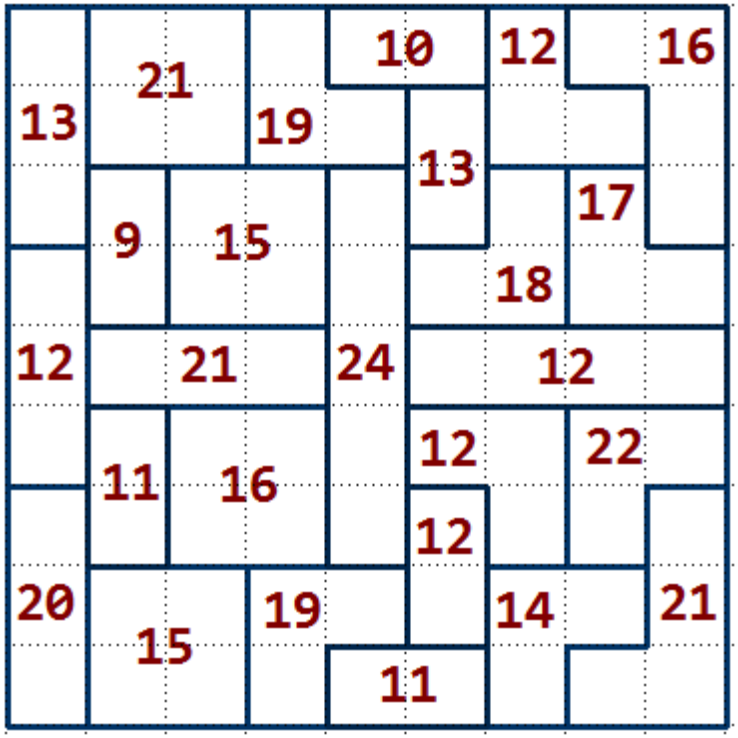
This is the database of 160 difficult puzzle which you will find in your Sudoku directory

## top160.sol

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

Number of solutions for puzzle 1: 1
+-------+-------+-------+
| 9 3 7 | 6 5 8 | 2 4 1 |
| 8 6 4 | 2 9 1 | 7 3 5 |
| 1 2 5 | 7 3 4 | 9 8 6 |
+-------+-------+-------+
| 5 8 3 | 4 1 9 | 6 2 7 |
| 6 4 9 | 3 7 2 | 5 1 8 |
| 7 1 2 | 5 8 6 | 4 9 3 |
+-------+-------+-------+
| 4 7 1 | 9 6 3 | 8 5 2 |
| 3 9 6 | 8 2 5 | 1 7 4 |
| 2 5 8 | 1 4 7 | 3 6 9 |
+-------+-------+-------+

Time = 0.6 seconds

----------------------------------------

Puzzle 2:
+-------+-------+-------+
| 5 3 0 | 0 2 0 | 9 0 0 |
| 0 2 4 | 0 3 0 | 0 5 0 |
| 0 0 9 | 0 0 0 | 0 0 0 |
+-------+-------+-------+
| 0 0 0 | 0 1 0 | 8 2 7 |
| 0 0 0 | 7 0 0 | 0 0 0 |
```

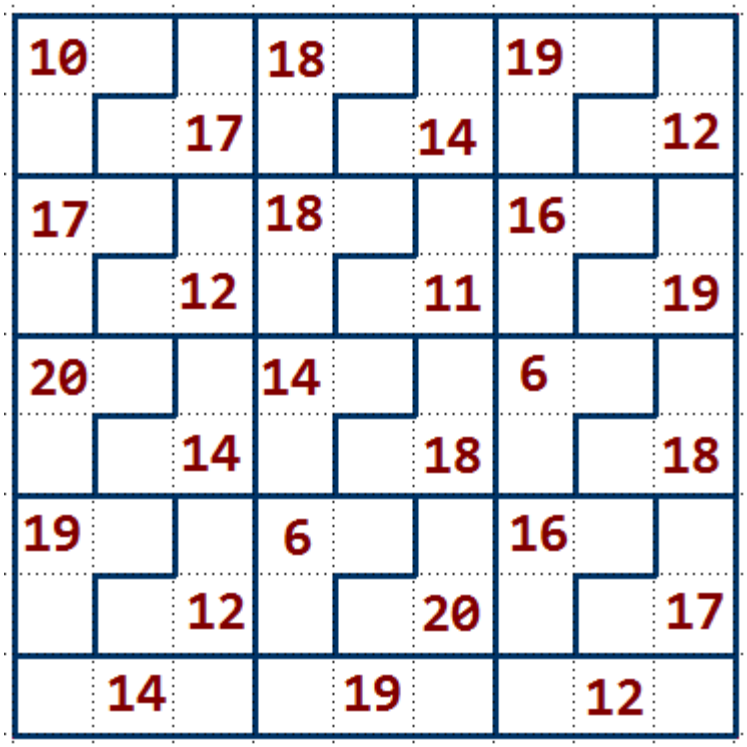
# Sudokiller (Bonus)

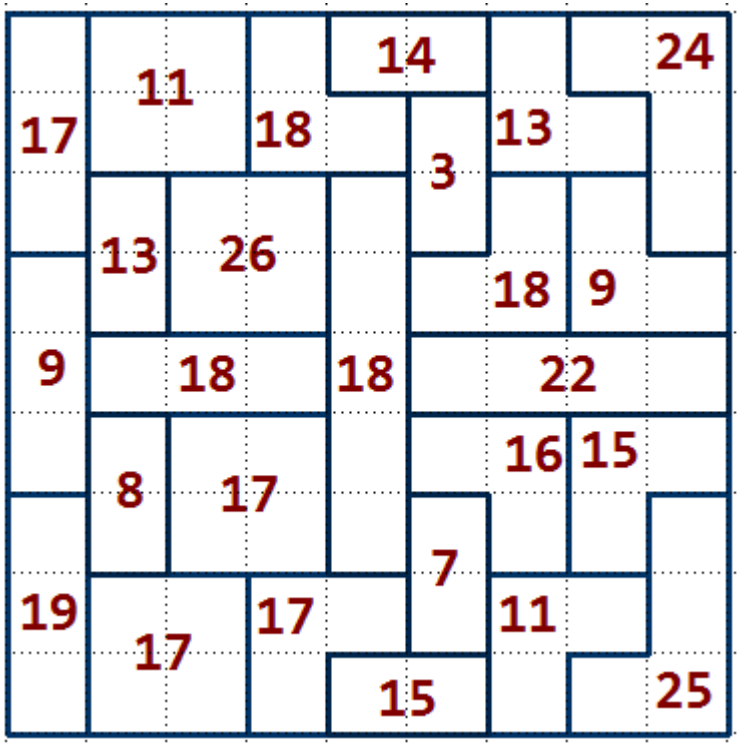
- The project grade weight is 30% but Sudokiller problem can raise it 45%
- An algorithm for solving Killer Sudoku puzzles can earn you extra 15% weight to your course grade! (so project grade is 45%)
- A **Sudokiller** puzzle consists of a partition of the board to cages. The number on each cage is the sum of the cells in the cage
- You must design the needed data structures and the algorithms needed for solving <u>any sudokiller puzzle</u> in a reasonable time frame (the faster the better your grade is …)
- The following puzzle has exactly **<u>two solutions</u>**.
  How fast can you find them?



# Diagonal Sudokiller

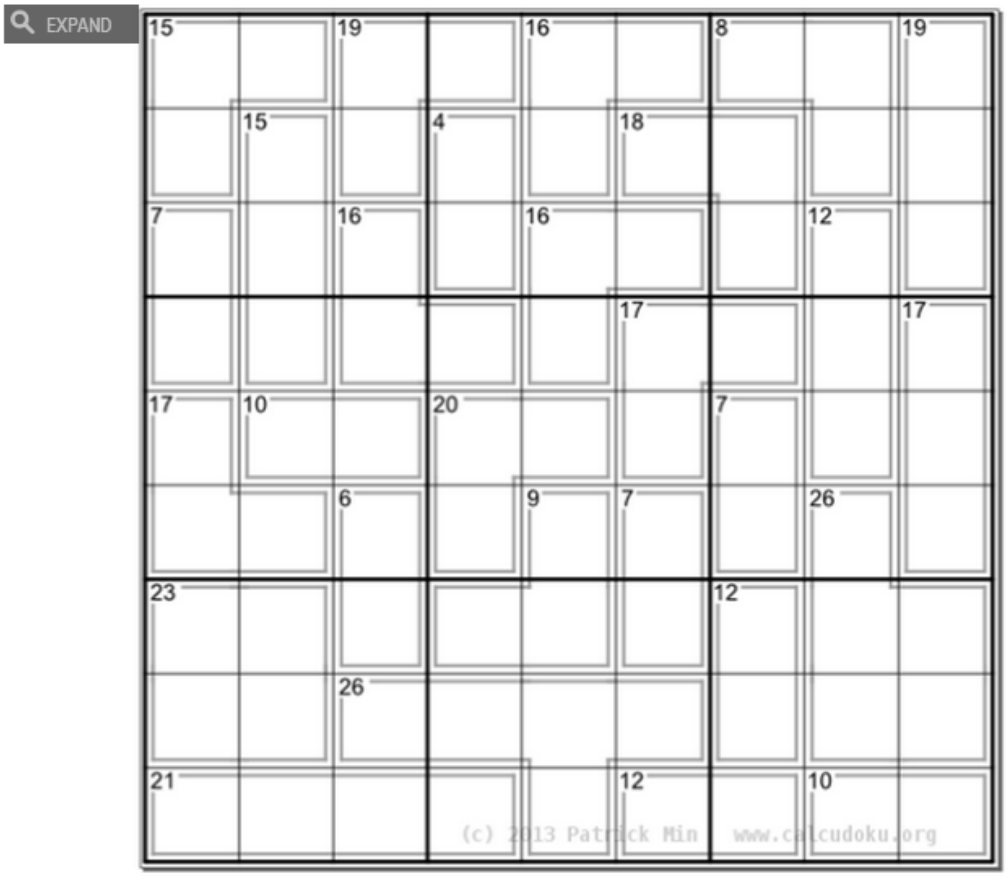
- The following three Sudokiller puzzles have also a **diagonal** solution
- You algorithm should also support diagonals too!
- Make sure the runtime of your programs does not exceed 10 hours (even for the hardest problem)



# Sudokiller (puzzle #3)

## 3. The World's Hardest Killer Sudoku



# Diagonal Sudokiller (puzzle #4)

- This puzzle has two solutions, diagonals included!
- You must find them both