Introduction

Objective:
Use Z3, a SMT solver for Python to solve sudokus. First, we'll see a naive way to model the problem; then we'll use some presolve on our instances to see how we can improve solving speed.

Sudokus:
Sudokus is a classic logical-combinatorial puzzle. Usually, it is presented as a 9 by 9 grid; divided in 9 squares of size 3 by 3. The player has some clues (ie some squares already contains a number) and must fill every empty square according to three rules:

- Each line contains every number exactly once. **(Line constraints)**
- Each collumn contains every number exactly once. **(Collumn constraints)**
- Each square contains every number exactly once. **(Square constraints)**

Instances:
We propose here to do slightly more than just solving 9 by 9 grids: we will solve n by n grids for $n \in \{4, 9, 16, 25\}$, and allow ourselves to use letter from a to f, plus capitals A to F for puzzles up to 25.
Thus, an instance presents itself as:

```
9
3XX 1XX XXX
28X XXX X36
X4X 2X6 XXX
7XX 69X 5X2
XXX X4X XXX
8X9 X23 XXX
XXX 5X4 X2X
62X XXX X79
XXX XX2 XX1
```

Figure 1: Instance 1

First number determines the size of a puzzle; all subsequent lines correspond to one line of the puzzle. A 'X' means it is a clue to uncover.

Z3 solving

Model:

- Let $X_{i,j} \in \{1..9\}$[1] be the value of the square at position (i, j).[2]
- Let C be the clue set; a clue $c \in C$ is of the form (i, j, v); where (i, j) correspond to a position in the grid; $v \in \{1..9\}$
- $\forall (i, j, v) \in C : X_{i,j} = v$ **(Clues)**
- $\forall i, j, j' \in \{1..9\}, j \neq j' \Rightarrow X_{i,j} \neq X_{i,j'}$ **(Line Constraints)**
- $\forall i, i', j \in \{1..9\}, i \neq i' \Rightarrow X_{i,j} \neq X_{i',j}$ **(Collumn Constraints)**
- $\forall i, j, i', j' \in \{1..9\}, (i = i'[3]) \wedge (j = j'[3]) \wedge ((i, j) \neq (i', j')) \Rightarrow X_{i,j} \neq X_{i',j'}$ [3] **(Square Constraints)**

Implementation details:
The aformentionned model is implemented in **project.py**. The function *line constraints* generates the constraints for the lines and the collumns; these being similar. *integrity constraints* make sure every $X_{i,j}$ takes its value in $\{1..9\}$.
The functions *str conversion* and *int conversion* convert str types in int and vice versa. In the code, $X_{i,j}$ is an integer for Z3, but when we need to save it (or read the instance) we need to have a conversion tool. The utility of these functions becomes more obvious when we have $n > 9$.
Other functions are self-explanatory.

Results:

Presolve

[1] Say $n = 9$ to keep it simple.
[2] First line is top line; first collumn is left collumn.
[3] The 3 comes from $\sqrt{9}$; understand $\sqrt{n}$ here.