# Report for the Sudoku Solver in Python
# Topic Paper #20

Alex Laird
CS-3210-01
4/22/09
Survey of Programming Languages
Spring 2009
Computer Science, (319) 360-8771
alexdlaird@cedarville.edu

| Grading Rubric | | Max | Earn |
|---|---|---|---|
| | On Time/Format | 1 | |
| | Correct | 5 | |
| | Clear | 2 | |
| | Concise | 2 | |
| | **Total** | 10 pts | |

| Peer Reviewer |
|---|
| |

## ABSTRACT
This paper discusses the complete Sudoku solver written using the Python programming language.

## Keywords
Sudoku, Python, Design, Implementation, Testing

## 1. INTRODUCTION
Sudoku is a form of puzzle that is three rows and three columns of squares. Each square then holds another three by three set of boxes. This gives a board of 81 boxes that need to be filled with the numbers one through nine. Each row must contain the set from one to nine, each column must contain the set from one to nine, and each box must contain the set from one to nine. Certain boxes are already filled with a number to start the puzzle, and the solver must fill in the remaining numbers to complete the puzzle.

This mind puzzle can easily be solved algorithmically or through brute force by writing a simple program, and that is what we did. A simple Sudoku Solver was written using the Python programming language and is powerful enough to solve any proper Sudoku puzzle of any difficult.

## 2. DESIGN DETAILS AND IMPLEMENTATION
The user specifies a puzzle through an input file. An input file may be specified via command line arguments, or if an input file is not specified the default "sudoku.txt" is used. If no input file is given, the program will not run.

The input file must contain a valid Sudoku puzzle in a form the same as below:

5 3 - | - 7 - | - - -

6 - - | 1 9 5 | - - -

- 9 8 | - - - | - 6 -

---

8 - - | - 6 ...

Squares are separated by spaces; boxes are ended horizontally by a vertical bar (|) and horizontally by at least three dashes (---) in a row. An empty box is represented by a single dash (-).

The program then loops through every box, using brute force methods to check the entire row, column, and square that the box is in to find an appropriate value the place in the box.

Appropriate errors are thrown if invalid numbers are given or if some critical fault occurs when attempting to solve. If the Sudoku puzzle given is not able to be complete, the program will get as close as is possible.

## 3. TESTING AND RESULTS
Initially stepping through each solution individually, we then moved on to test numerous valid Sudoku puzzles into the solver and ensured that it solved them correctly. Though the solver couldn't always completely solve the puzzle (when we gave it impossible puzzles), it did get as close as was possible using brute force.

Since brute force is used and no more efficient algorithms in the solving, this solver is obviously not the fastest in production. However, given the miniscule amount of time it takes to solve even the most complicated Sudoku puzzles, this isn't that big of a drawback.

## 4. CONCLUSIONS
The Sudoku solver was fairly easy to implement and test. Though it is not very efficient, it will solve any valid puzzle correctly and in a small amount of time, and if the puzzle isn't valid it will solve it as far as it can.


*Alex Laird, Cedarville University, Cedarville, Ohio, 45314*