

# Kulliyyah of Information & Communication Technology Department of Computer Science

**Sem 2, 2018/2019**

**Intelligent Systems CSC 2301**

**Section 01**

**Lecturer’s Name: DR. NORZALIZA BINTI MD NOR**

**SUDOkU GAME USING PYTHON**

|  |  |  |
| --- | --- | --- |
| 1 | Zian Md Afique Amin | 1631005 |
| 2 | Khan Nasik Sami | 1638153 |
| 4 | Suhib Ahmad | 1423183 |

**INTRODUCTION**

Python is a general-purpose language, which means it can be used to build just about anything, which will be made easy with the appropriate tools or libraries. Professionally, Python is great for backend web development, data analysis, artificial intelligence and scientific computing.

We are planning to make a game named “**Sudoku”** using Python script.

The classic **Sudoku** game involves a grid of 81 squares. The grid is divided into nine blocks, each containing nine squares.

The **rules** of the game are simple:

1. Each of the nine blocks has to contain all the numbers 1-9 within its squares.
2. Each number can only appear once in a row, column or box.

# PROJECT OBJECTIVES

* Develop a fully functioning Sudoku game with Python programming language.
* Provide an aesthetically enhance and interactive interfaces develop with python.
* The code will be fully working and usable by using simple and easy syntax.

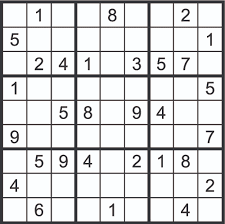
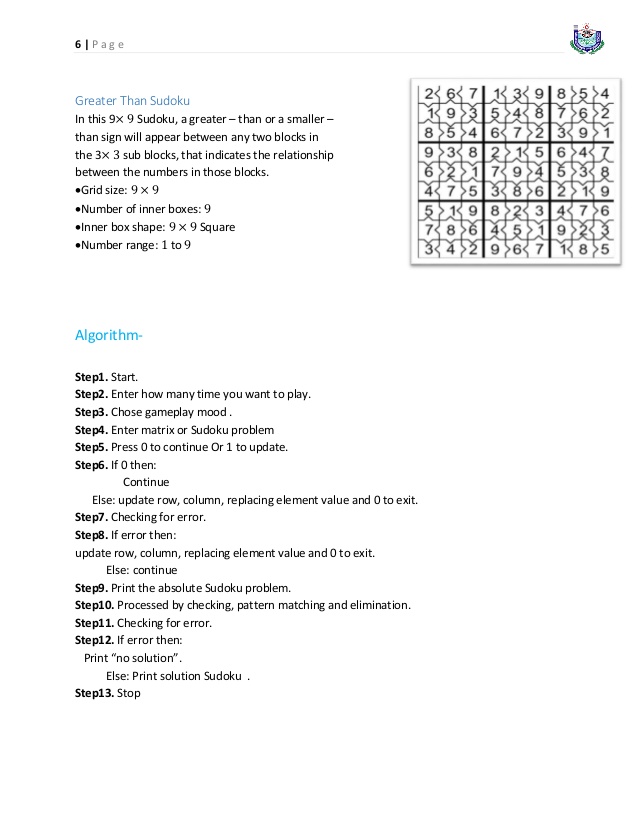


Figure: Sudoku

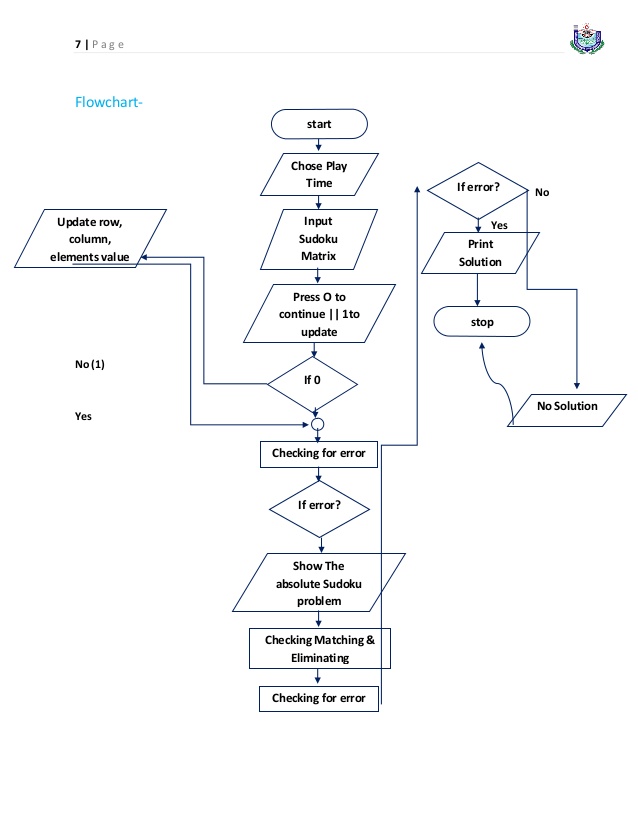
**Expected output:**

As per game rules and regulations, A person can win the game by putting all the right digits to it’s suitable places. It’s a lengthy game. And the answer is randomly changed by the computer so every time if a player restart the game , the anser will also be changed. If a person press the wrong keyword, the program able to detect where is the right digit.

**Rules Of Algorithm:**



**FlowChart For Sudoku:**



**Code Using Python:**

import curses

import random

import time

class Block(object):

"""Store a single 3x3 block in the complete field."""

def \_\_init\_\_(self):

self.\_block = [0, 0, 0, 0, 0, 0, 0, 0, 0]

def \_\_setitem\_\_(self, point, value):

"""Set the value, check that value is not already in the block."""

# n == -n, as n < 0 means user generated abs(n).

if abs(self.\_block[point[1] + 3 \* point[0]]) == abs(value):

self.\_block[point[1] + 3 \* point[0]] = value

return

if value != 0 and (value in self.\_block or -value in self.\_block):

raise ValueError("Block already contains %d" % value)

self.\_block[point[1] + 3 \* point[0]] = value

def \_\_getitem\_\_(self, point):

"""Return the value at the given point."""

return self.\_block[point[1] + 3 \* point[0]]

class Sudoku(object):

def \_\_init\_\_(self):

self.\_field = [Block() for i in range(9)]

# Cached values used in solve() and populate()

self.\_values = list(range(1, 10))

self.\_points = [(x, y) for y in range(9) for x in range(9)]

def clear(self):

"""Reset the game to empty."""

for b in self.\_field:

b.\_block = [0, 0, 0, 0, 0, 0, 0, 0, 0]

def candidates(self, point):

"""Return all candidates at the point (x, y)."""

candidates = set()

previous = self[point]

for i in self.\_values:

try:

self[point] = i

except ValueError:

pass

else:

candidates.add(i)

self[point] = previous

return candidates

def populate(self, n=36):

"""Populate can fields of the Sudoku.

Clear the Sudoku, run the solver using random values and

remove as many values as possible.

"""

# Randomize the list of points and values

random.shuffle(self.\_points)

random.shuffle(self.\_values)

self.clear()

self.solve()

for point in self.\_points:

if self[point] == 0:

continue

val = self[point]

for v in self.candidates(point):

if v == val:

continue

self[point] = v

if self.solve(True):

self[point] = val

break

else:

if (81 - sum(b.\_block.count(0) for b in self.\_field) < n):

self[point] = val

break

self[point] = 0

for point in self.\_points:

self[point] \*= -1

def \_\_str\_\_(self):

"""Represent the Sudoku as a string (for debugging purposes)."""

s = ""

for y in range(9):

for x in range(9):

s += '%d ' % self[x, y]

if (x + 1) % 3 == 0:

s += " "

s += "\n"

if (y + 1) % 3 == 0:

s += "\n"

return s.strip()

def \_\_getitem\_\_(self, p):

pb = (p[0] // 3, p[1] // 3)

block = self.\_field[pb[1] + 3 \* pb[0]]

return block[p[0] % 3, p[1] % 3]

def \_\_setitem\_\_(self, p, val):

if self[p] < 0 and val >= 0:

raise ValueError("Value at point %s is pre-defined" % str(p))

if val != 0:

for i in range(9):

if i != p[1] and abs(self[p[0], i]) == abs(val):

raise ValueError("Already in column: %d" % val)

if i != p[0] and abs(self[i, p[1]]) == abs(val):

raise ValueError("Already in row: %d" % val)

pb = (p[0] // 3, p[1] // 3)

block = self.\_field[pb[1] + 3 \* pb[0]]

block[p[0] % 3, p[1] % 3] = val

def is\_solved(self):

return all(0 not in self.\_field[i].\_block for i in range(0, 9))

def solve(self, reset=False):

"""Solve the Sudoku.

If 'reset' is True, just check whether the sudoku can be solved,

after return the sudoku will be identical to before the call.

"""

if self.is\_solved():

return True

field = None

for x in range(9):

if field is not None:

break

for y in range(9):

if self[x, y] == 0:

field = (x, y)

break

if field is None:

return True

for new in self.\_values:

try:

self[field] = new

except ValueError:

continue

else:

if self.solve(reset):

if reset:

self[field] = 0

return True

else:

self[field] = 0

class CursesUI(object):

"""Command-line 'curses' interface."""

def \_\_init\_\_(self):

self.\_screen = curses.initscr()

self.\_sudoku = Sudoku()

# Draw the borders

self.\_screen.vline(0, 4 \* 3 - 2, curses.ACS\_VLINE, 2 \* 9 - 1)

self.\_screen.vline(0, 4 \* 6 - 2, curses.ACS\_VLINE, 2 \* 9 - 1)

self.\_screen.hline(2 \* 3 - 1, 0, curses.ACS\_HLINE, 4 \* 9 - 3)

self.\_screen.hline(2 \* 6 - 1, 0, curses.ACS\_HLINE, 4 \* 9 - 3)

self.\_screen.addch(2 \* 3 - 1, 4 \* 3 - 2, curses.ACS\_PLUS)

self.\_screen.addch(2 \* 3 - 1, 4 \* 6 - 2, curses.ACS\_PLUS)

self.\_screen.addch(2 \* 6 - 1, 4 \* 3 - 2, curses.ACS\_PLUS)

self.\_screen.addch(2 \* 6 - 1, 4 \* 6 - 2, curses.ACS\_PLUS)

self.\_draw\_sudoku()

curses.noecho()

curses.cbreak()

self.\_screen.keypad(1)

def \_\_enter\_\_(self, \*a):

return self

def \_\_exit\_\_(self, \*a):

curses.nocbreak()

self.\_screen.keypad(0)

curses.echo()

curses.endwin()

def \_draw\_sudoku(self):

for y in range(0, 9):

for x in range(0, 9):

value = self.\_sudoku[x, y]

attr = curses.A\_BOLD if value < 0 else 0

# Mark cells with only one possible solution.

#if len(self.\_sudoku.candidates((x,y))) == 1:

# attr |= curses.A\_UNDERLINE

self.\_screen.addch(2 \* y, 4 \* x, ord('0') + abs(value), attr)

def \_print\_string(self, string):

self.\_screen.move(20, 0)

self.\_screen.deleteln()

self.\_screen.addstr(string)

def \_help(self):

self.\_print\_string("Commands: (q)uit, (p)opulate, (P)opulate, "

"(r)eset, (c)andidates, (s)olve")

def main(self):

x = 0

y = 0

self.\_help()

while True:

self.\_screen.move(y, x)

c = chr(self.\_screen.getch())

if c == ':':

self.\_print\_string(":")

curses.echo()

#self.\_screen.move(20, 0)

c = self.\_screen.getstr()

curses.noecho()

if c == 'q':

break # Exit the while()

elif c == 'h':

self.\_help()

elif c in ('p', 'P'):

if c == 'P':

self.\_print\_string("Enter number of fields: ")

curses.echo()

n = int(self.\_screen.getstr())

curses.noecho()

else:

n = 36

start = time.time()

self.\_sudoku.populate(n)

end = time.time()

self.\_draw\_sudoku()

self.\_print\_string("Populated with %d fields in %.3f seconds"

% (n, end - start))

elif c == 's':

if not self.\_sudoku.solve():

self.\_print\_string("Could not solve sudoku")

self.\_draw\_sudoku()

elif c == 'r':

self.\_sudoku = Sudoku()

self.\_draw\_sudoku()

elif c == 'c':

point = (x // 4, y // 2)

self.\_print\_string("Candidates for %s are: %s" %

(point, self.\_sudoku.candidates(point)))

elif c == chr(curses.KEY\_LEFT):

x -= x >= 4 and 4 or 0

elif c == chr(curses.KEY\_RIGHT):

x += x + 4 < 4 \* 9 and 4 or 0

elif c == chr(curses.KEY\_UP):

y -= y >= 2 and 2 or 0

elif c == chr(curses.KEY\_DOWN):

y += y + 2 < 2 \* 9 and 2 or 0

elif c in "0123456789":

try:

self.\_sudoku[x // 4, y // 2] = ord(c) - ord('0')

self.\_screen.addch(y, x, c)

# If single-candidate cells should be highlighted, use the

# following instead of self.\_screen.addch():

#self.\_draw\_sudoku()

except Exception as e:

self.\_print\_string(str(e))

else:

if self.\_sudoku.is\_solved():

self.\_print\_string("Solved")

else:

self.\_print\_string("")

else:

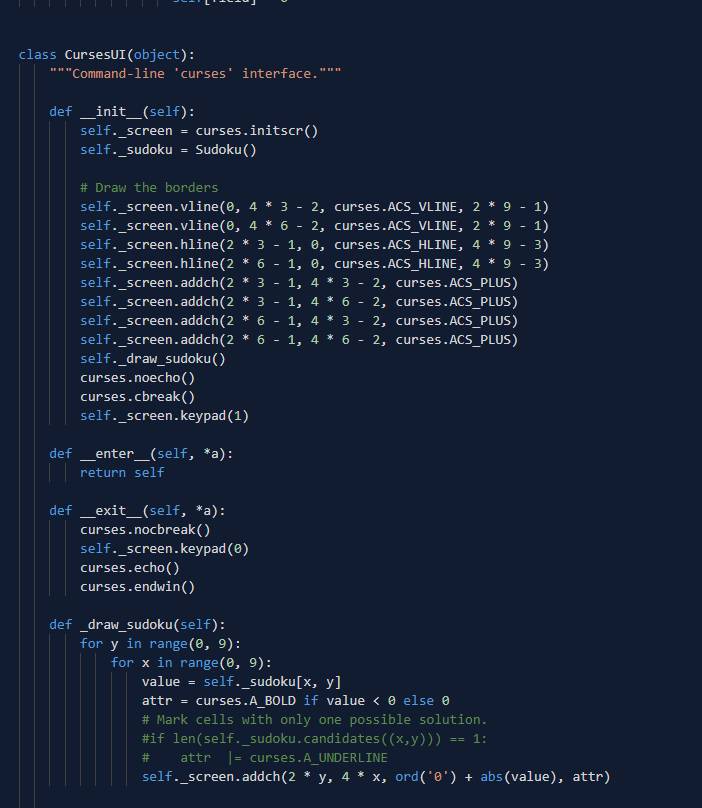
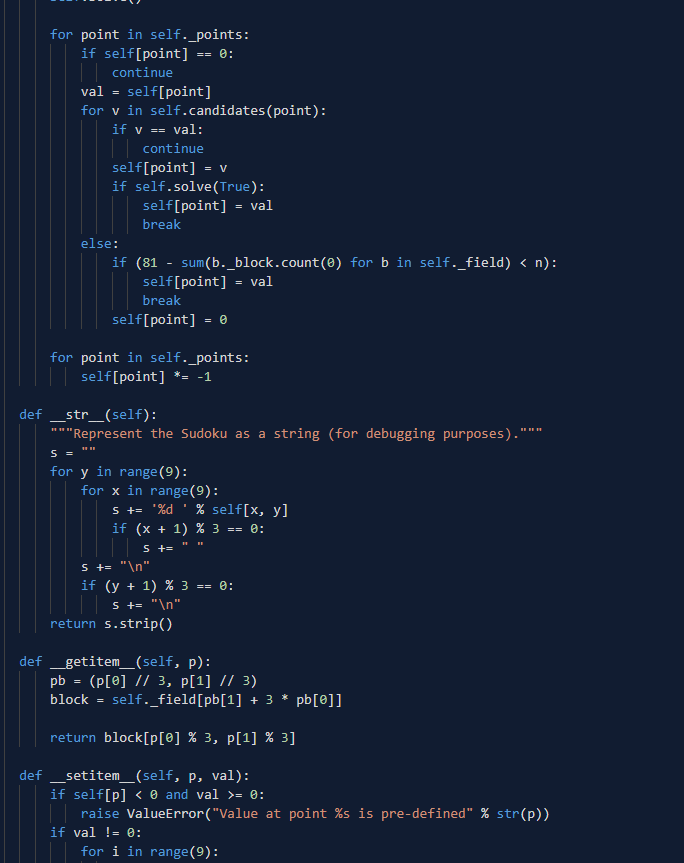
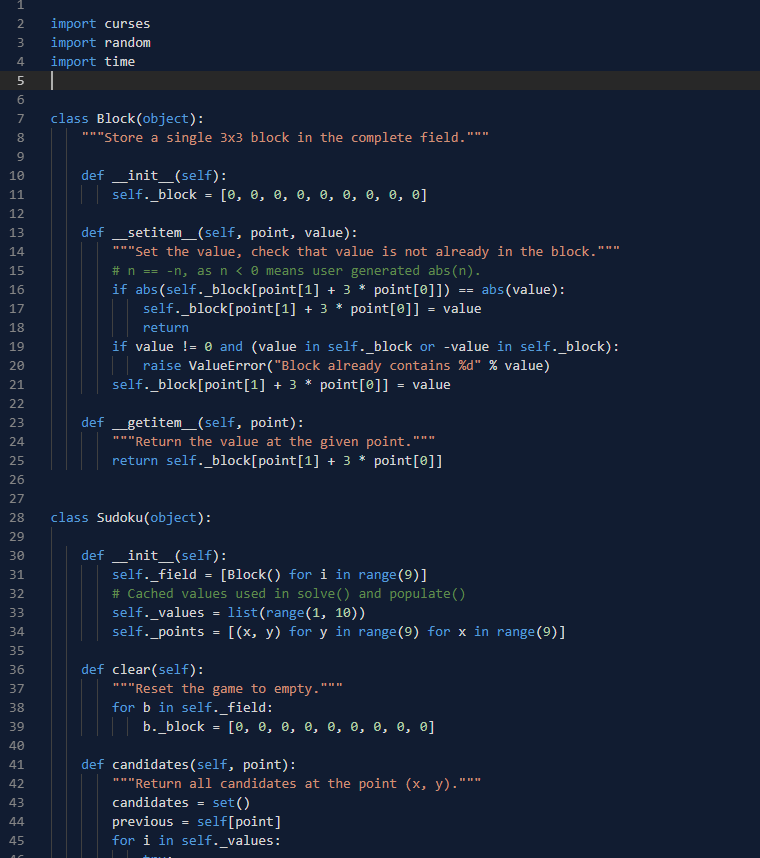
self.\_print\_string("Error: Invalid command '%s'" % c)

if \_\_name\_\_ == '\_\_main\_\_':

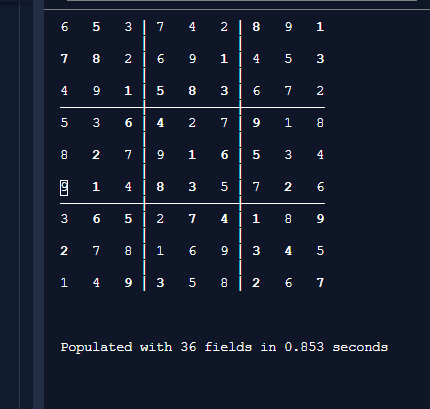
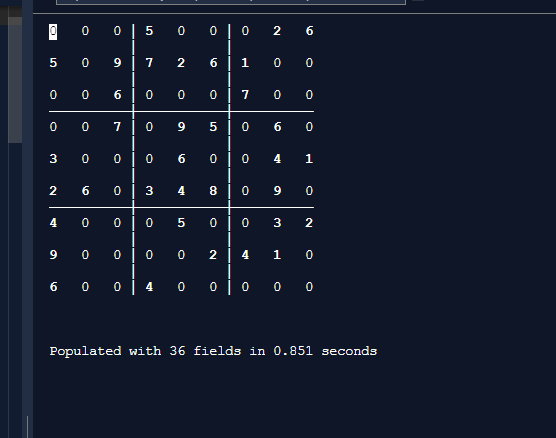
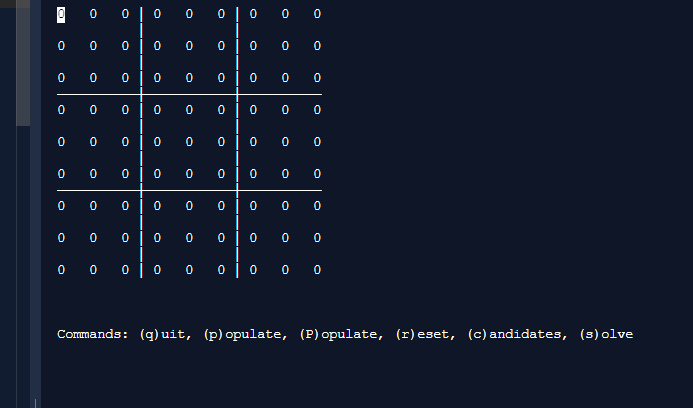
with CursesUI() as ui:

ui.main()

**ScreenShots:**



**Output:**



**References:**

1. <https://repl.it/@afiquezian/MassiveStylishSphere>
2. <https://www.google.com/imgres?imgurl=https://sudokustudylib.readthedocs.io/en/latest/_images/flowchart.png&imgrefurl=https://sudokustudylib.readthedocs.io/en/latest/program.html&h=837&w=586&tbnid=nFZL70AYIuK0QM&q=sudoku+flowchart&tbnh=144&tbnw=101&usg=AI4_-kRDUkTOpfl8bU5Af2EVo85SdUevCA&vet=1&docid=nq19FD8OgX1cyM&itg=1&sa=X&ved=2ahUKEwj4yqfK7fzhAhVVk3AKHSXLCSUQ_h0wC3oECAsQBA>
3. <https://en.wikipedia.org/wiki/Sudoku>
4. <http://www.sudokuwiki.org/sudoku.htm>