PYTHON

GAME

Submitted By :

Shubham Kumar	24045
Teesha Jain	24023
Jivanshu Magon	24033
Ruchi	24059
Priyal Jain	24025

ABOUT PYTHON

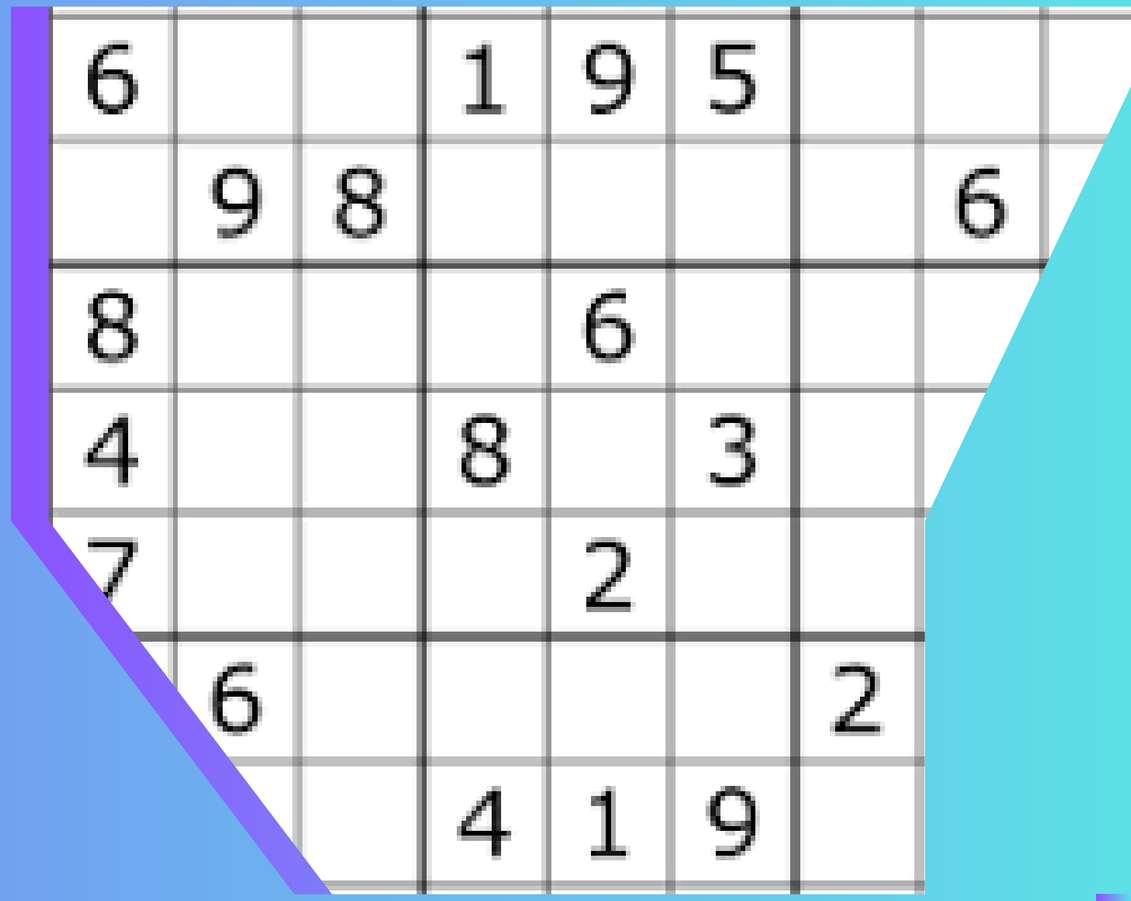
01

Python is a high-level, interpreted, general-purpose programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in 1991. Python is widely used in various domains such as web development, data science, machine learning, automation, scripting, game development, and more.

02

Why Use Python for Projects?

- Short development time
- Large community support
- Excellent third-party libraries
- Beginner-friendly yet powerful for advanced tasks



ABOUT GAME SUDOKU

Sudoku is a popular logic-based number puzzle game that challenges players to fill a 9x9 grid with digits from 1 to 9. The grid is divided into nine 3x3 subgrids, often called "boxes" or "regions." The objective of the game is simple: fill in the entire grid so that every row, every column, and every 3x3 box contains all the digits from 1 to 9 without repeating.

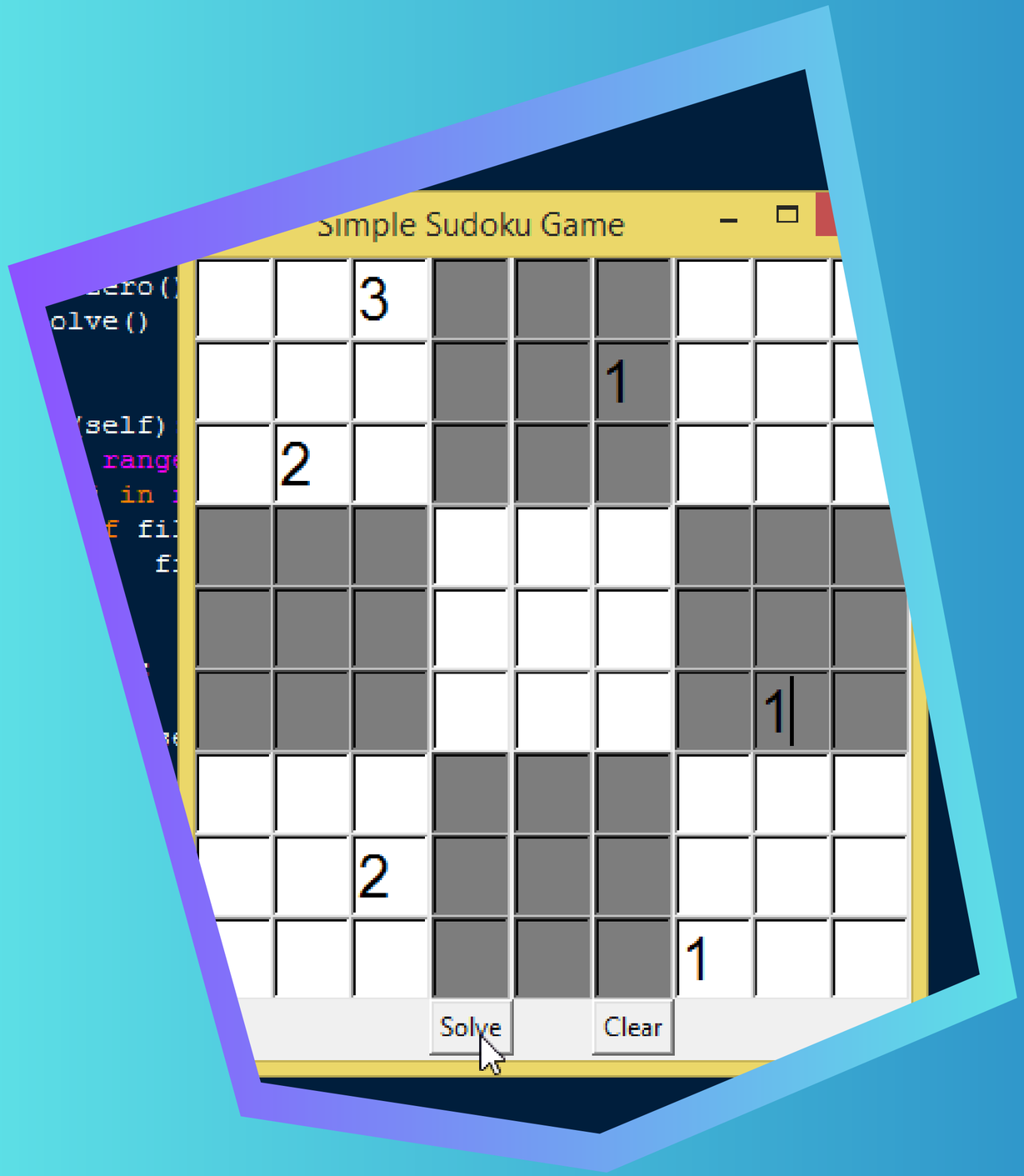
PYGAME

- Pygame is a free and open-source Python library used for developing video games and multimedia applications. It provides a simple and easy-to-use framework for creating 2D games and handling game elements such as graphics, sounds, and user inputs.
- Pygame is built on top of the Simple DirectMedia Layer (SDL) library, which allows access to hardware like graphics cards, sound devices, and game controllers. It is well-suited for beginners in game development, as it abstracts complex low-level programming and makes game creation easier and more fun.

LIBRARIES USED FOR SUDOKU :

Pygame	used to render the Sudoku grid and interact with the user
Time	used to add delay for visual steps during solving
Sys	for exiting the game

4	3		
1	2	3	
		2	
2	1		



CODE STRUCTURE OVERVIEW

01 Grid - represents the full board logic and interface

02 Cube – represents individual cells

03 solve() – backtracking function

04 is_valid() – checks if placing a number is valid

05 draw() – draws the board and numbers

06 main() – runs the game loop

```python
import pygame
import time
import sys

pygame.font.init()

# Window setup
WIDTH, HEIGHT = 540, 600
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Sudoku Solver Game")

# Colors and Fonts
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GRAY = (220, 220, 220)
BLUE = (0, 0, 255)
FONT = pygame.font.SysFont("comicsans", 40)
SMALL_FONT = pygame.font.SysFont("comicsans", 30)

# Initial puzzle
initial_board = [
    [7, 8, 0, 4, 0, 0, 1, 2, 0],
    [6, 0, 0, 0, 7, 5, 0, 0, 9],
    [0, 0, 0, 6, 0, 1, 0, 7, 8],
    [0, 0, 7, 0, 4, 0, 2, 6, 0],
    [0, 0, 1, 0, 5, 0, 9, 3, 0],
    [9, 0, 4, 0, 6, 0, 0, 0, 5],
    [0, 7, 0, 3, 0, 0, 0, 1, 2],
    [1, 2, 0, 0, 0, 7, 4, 0, 0],
    [0, 4, 9, 2, 0, 6, 0, 0, 7]
]

board = [row[:] for row in initial_board]
selected = None

def draw_grid(win):
    gap = WIDTH // 9
    for i in range(10):
        line_width = 4 if i % 3 == 0 else 1
        pygame.draw.line(win, BLACK, (0, i * gap), (WIDTH, i * gap), line_width)
        pygame.draw.line(win, BLACK, (i * gap, 0), (i * gap, WIDTH), line_width)

def draw_numbers(win, board):
    gap = WIDTH // 9
    for i in range(9):
        for j in range(9):
            if board[i][j] != 0:
                text = FONT.render(str(board[i][j]), True, BLACK)
                win.blit(text, (j * gap + 20, i * gap + 10))

def draw_window(win, board):
    win.fill(WHITE)
    draw_grid(win)
```

```
    pygame.display.update()

def find_empty(board):
    for i in range(9):
        for j in range(9):
            if board[i][j] == 0:
                return (i, j)
    return None

def is_valid(board, num, pos):
    row, col = pos
    if any(board[row][j] == num for j in range(9) if j != col):
        return False
    if any(board[i][col] == num for i in range(9) if i != row):
        return False
    box_x, box_y = col // 3, row // 3
    for i in range(box_y * 3, box_y * 3 + 3):
        for j in range(box_x * 3, box_x * 3 + 3):
            if board[i][j] == num and (i, j) != pos:
                return False
    return True

def solve_visual(board):
    find = find_empty(board)
    if not find:
        return True
    else:
        row, col = find

    for num in range(1, 10):
        if is_valid(board, num, (row, col)):
            board[row][col] = num
            draw_window(WIN, board)
            pygame.time.delay(50)

            if solve_visual(board):
                return True

            board[row][col] = 0
            draw_window(WIN, board)
            pygame.time.delay(50)

    return False

def click(pos):
    if pos[0] < WIDTH and pos[1] < WIDTH:
        gap = WIDTH // 9
        x = pos[0] // gap
        y = pos[1] // gap
        return (y, x)
    else:
        return None
```

```python
def sketch(num):
    if selected:
        row, col = selected
        if initial_board[row][col] == 0:
            board[row][col] = num

def main():
    global selected, board
    run = True
    draw_window(WIN, board)

    while run:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                pygame.quit()
                sys.exit()

            if event.type == pygame.MOUSEBUTTONDOWN:
                pos = pygame.mouse.get_pos()
                clicked = click(pos)
                if clicked:
                    selected = clicked

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_1:
                    sketch(1)
```

```python
                if event.key == pygame.K_2:
                    sketch(2)
                if event.key == pygame.K_3:
                    sketch(3)
                if event.key == pygame.K_4:
                    sketch(4)
                if event.key == pygame.K_5:
                    sketch(5)
                if event.key == pygame.K_6:
                    sketch(6)
                if event.key == pygame.K_7:
                    sketch(7)
                if event.key == pygame.K_8:
                    sketch(8)
                if event.key == pygame.K_9:
                    sketch(9)
                if event.key == pygame.K_SPACE:
                    solve_visual(board)
                if event.key == pygame.K_r:
                    board = [row[:] for row in initial_board]

        draw_window(WIN, board)

if __name__ == "__main__":
    main()
```

KEY FUNCTIONS

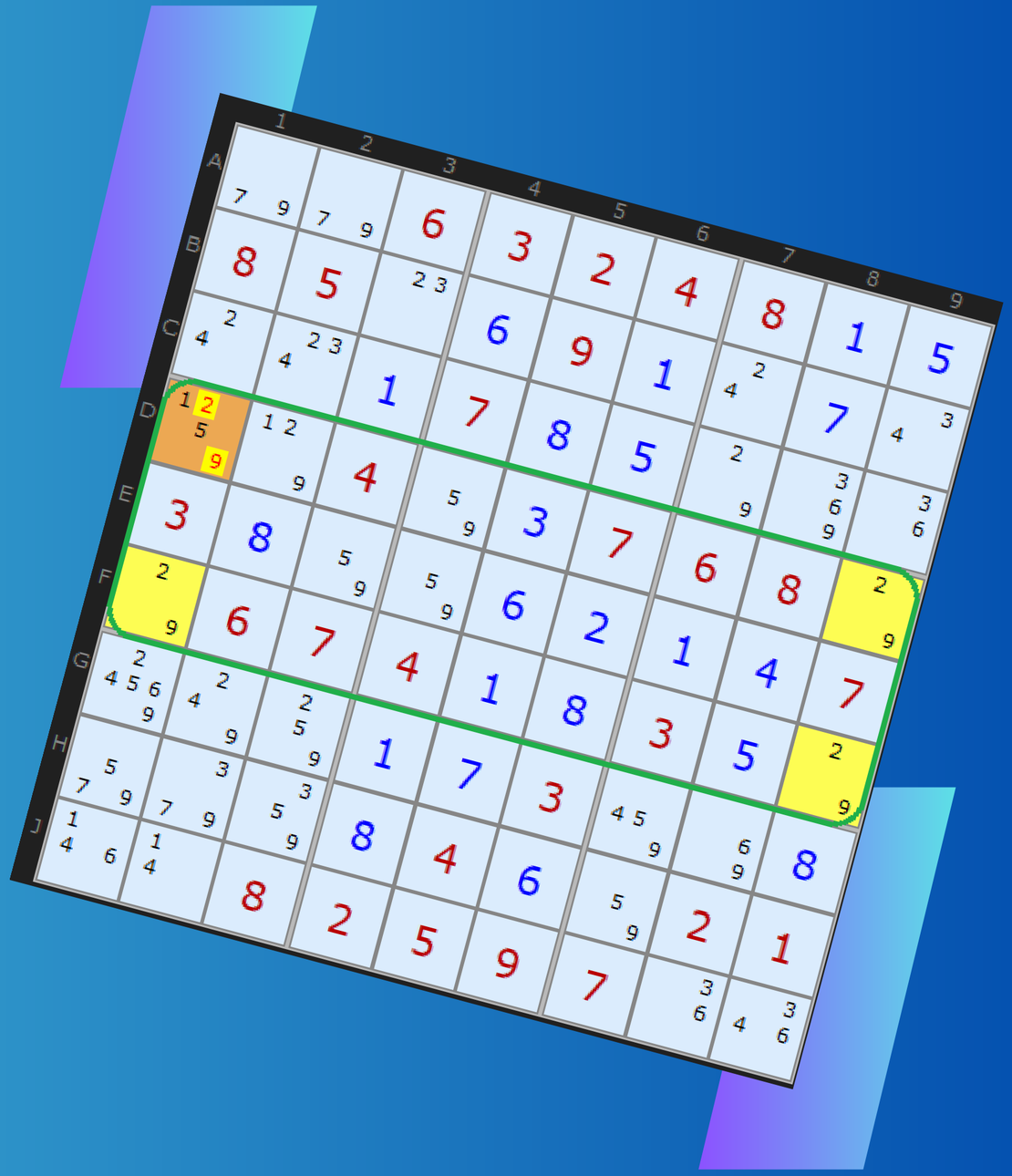
01 draw_grid(): Draws the grid lines

02 draw(): Displays values inside each

03 update_model(): Syncs visual grid with backend

04 place(): Attempts to place a number based on user input

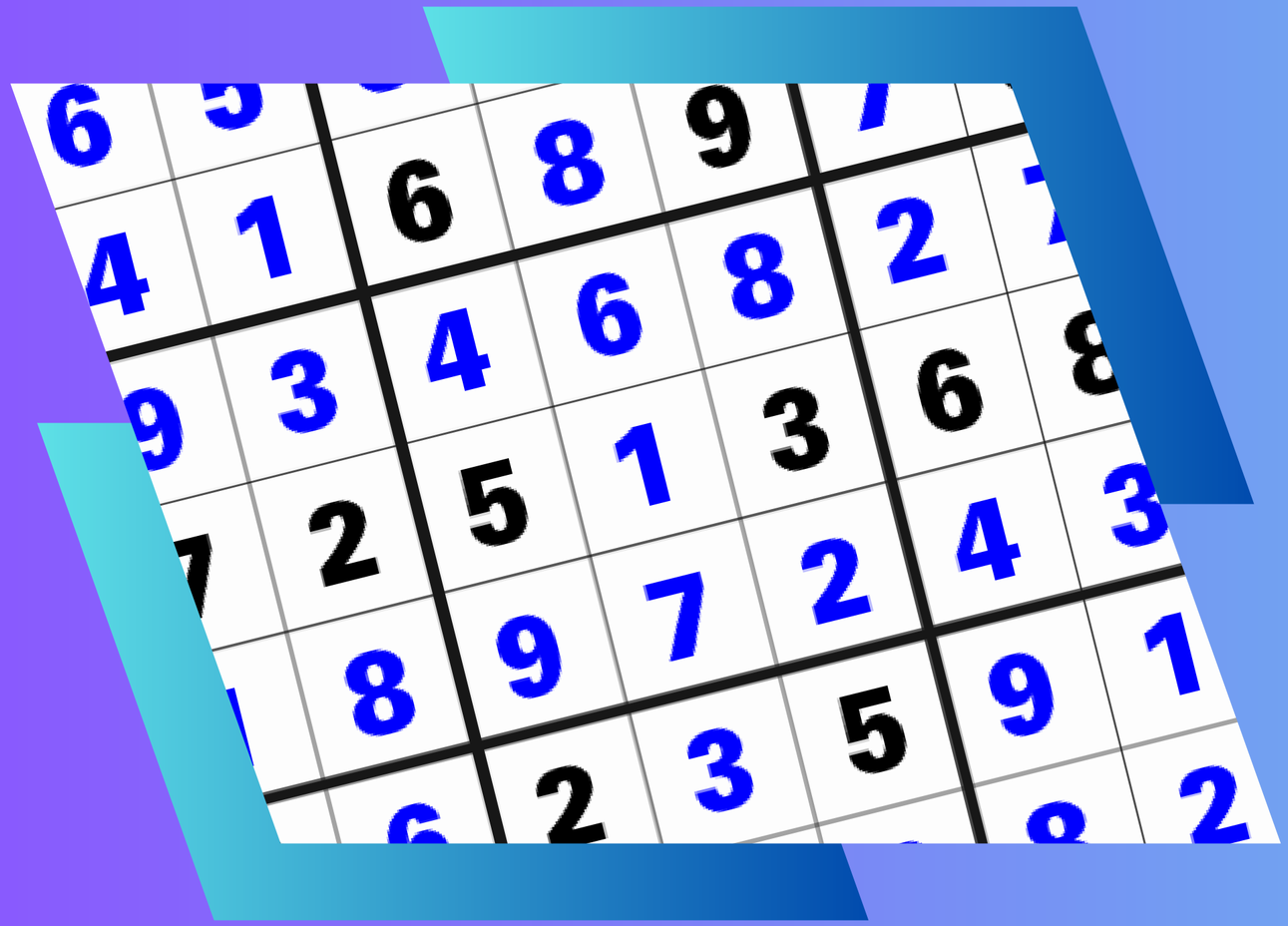
05 solve_gui(): Visual version of solve using Pygame updates



CHALLENGES & LEARNINGS

- Integrating logic with GUI
- Real-time updates using pygame.display.update()

- Visualizing recursive steps
- Managing user input vs program solving



RECOMMENDATIONS & FUTURE SCOPE

- Add difficulty levels
- Allow puzzle import via image (OCR)
- Improve GUI (colors, themes)
- Add timer or leaderboard for user solving

THANKS