**PRODIGY INFOTECH**

**Build a Temperature Conversion Program**

**Create a program that converts temperatures between Celsius, Fahrenheit, and Kelvin scales. The program should prompt the user to input a temperature value and the original unit of measurement. It should then convert the temperature to the other two units and display the converted values to the user. For example, if the user enters a temperature of 25 degrees Celsius, the program should convert it to Fahrenheit and Kelvin, and present the converted values as outputs.**

def celsius\_to\_fahrenheit(celsius):

return (celsius \* 9/5) + 32

def celsius\_to\_kelvin(celsius):

return celsius + 273.15

def fahrenheit\_to\_celsius(fahrenheit):

return (fahrenheit - 32) \* 5/9

def fahrenheit\_to\_kelvin(fahrenheit):

return (fahrenheit + 459.67) \* 5/9

def kelvin\_to\_celsius(kelvin):

return kelvin - 273.15

def kelvin\_to\_fahrenheit(kelvin):

return (kelvin \* 9/5) - 459.67

def main():

# Prompt user for input

temp\_value = float(input("Enter the temperature value: "))

temp\_unit = input("Enter the original unit (C for Celsius, F for Fahrenheit, K for Kelvin): ").upper()

if temp\_unit == 'C':

fahrenheit = celsius\_to\_fahrenheit(temp\_value)

kelvin = celsius\_to\_kelvin(temp\_value)

print(f"{temp\_value}°C is {fahrenheit}°F and {kelvin}K")

elif temp\_unit == 'F':

celsius = fahrenheit\_to\_celsius(temp\_value)

kelvin = fahrenheit\_to\_kelvin(temp\_value)

print(f"{temp\_value}°F is {celsius}°C and {kelvin}K")

elif temp\_unit == 'K':

celsius = kelvin\_to\_celsius(temp\_value)

fahrenheit = kelvin\_to\_fahrenheit(temp\_value)

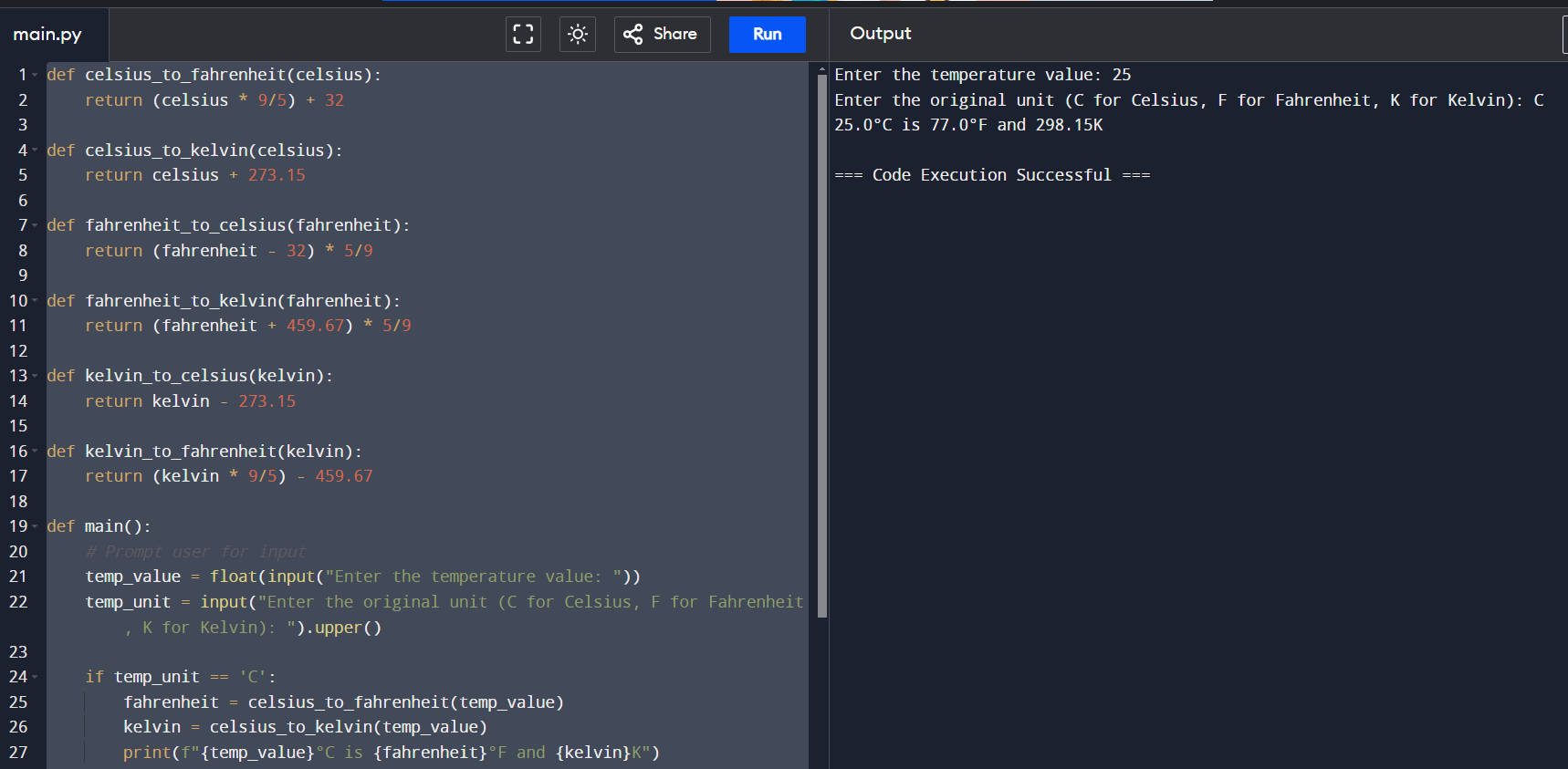
print(f"{temp\_value}K is {celsius}°C and {fahrenheit}°F")

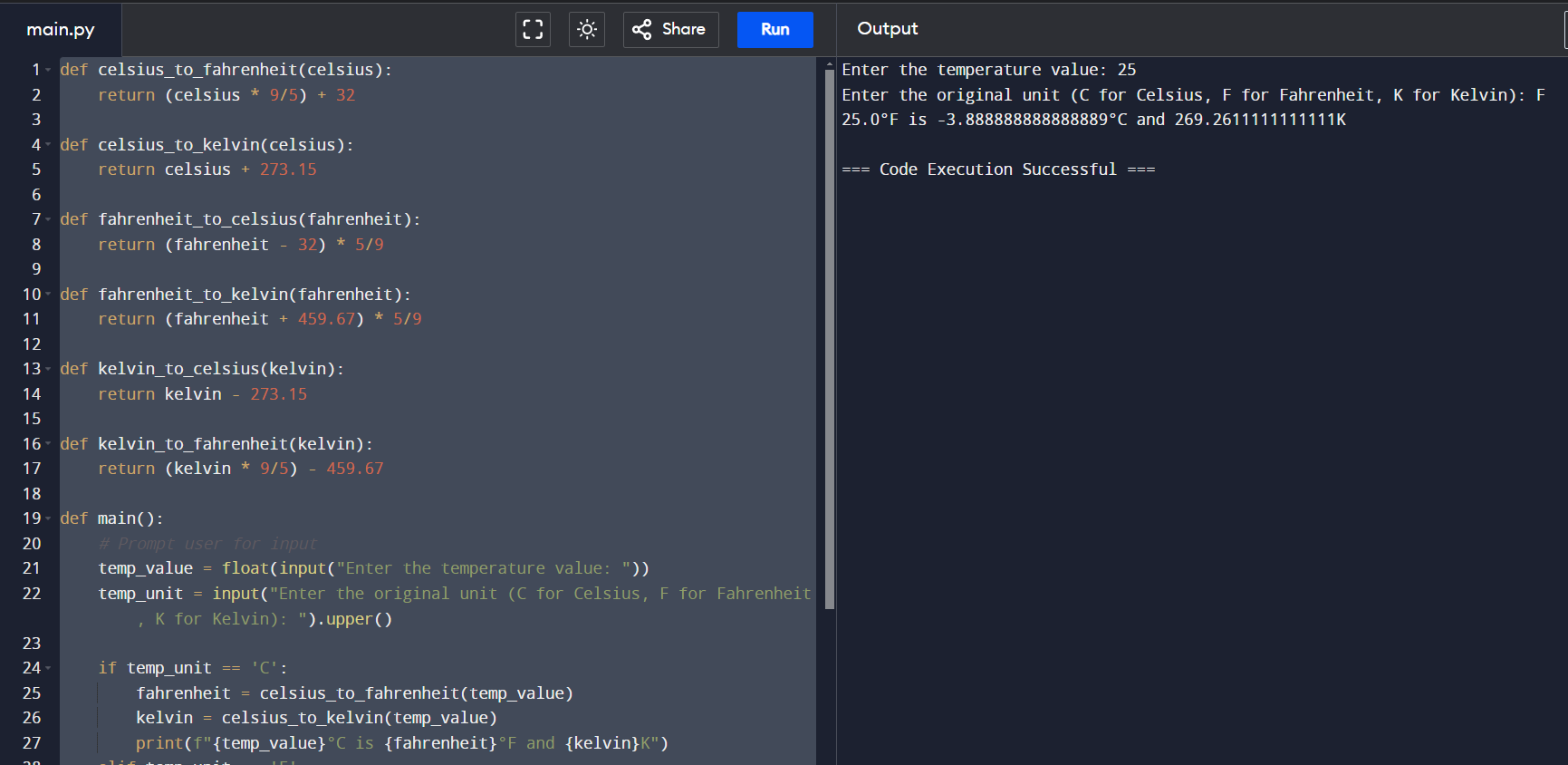
else:

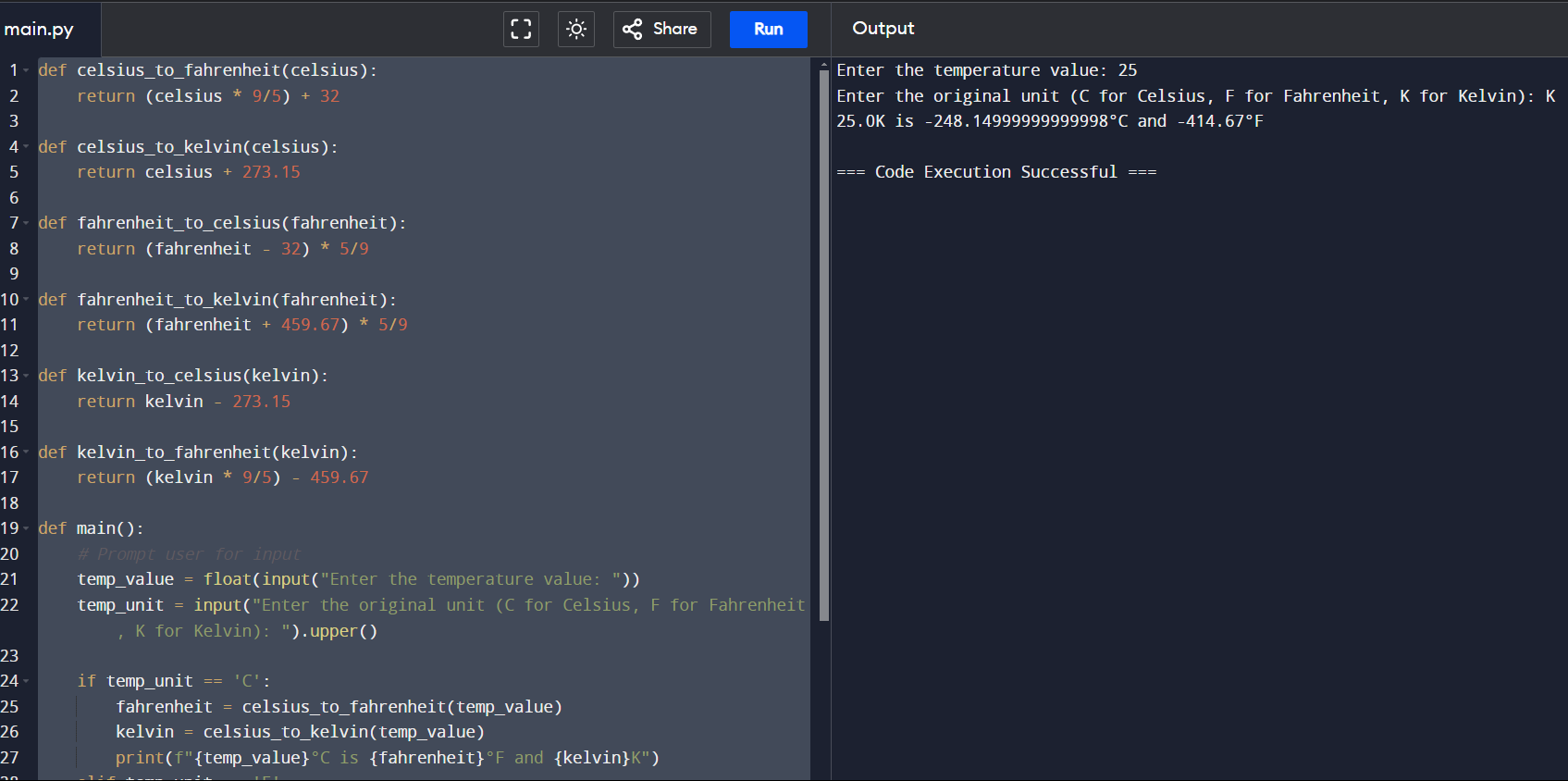
print("Invalid unit of temperature. Please enter C, F, or K.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

S





**Task-02**

**PRODIGY INFOTECH**

**Create a Guessing Game**

**Build a program that generates a random number and challenges the user to guess it. The program should prompt the user to input their guess, compare it to the generated number, and provide feedback if the guess is too high or too low. It should continue until the user correctly guesses the number and then display the number of attempts it took to win the game.**

import random

def guessing\_game():

print("Welcome to the Guessing Game!")

# Generate a random number between 1 and 100 (adjust range as needed)

secret\_number = random.randint(1, 100)

attempts = 0

while True:

try:

guess = int(input("Enter your guess (between 1 and 100): "))

attempts += 1

if guess < 1 or guess > 100:

print("Please enter a number within the valid range (1 to 100).")

continue

if guess < secret\_number:

print("Too low! Try again.")

elif guess > secret\_number:

print("Too high! Try again."

else:

print(f"Congratulations! You guessed the number {secret\_number} correctly.")

print(f"It took you {attempts} attempts to win the game.")

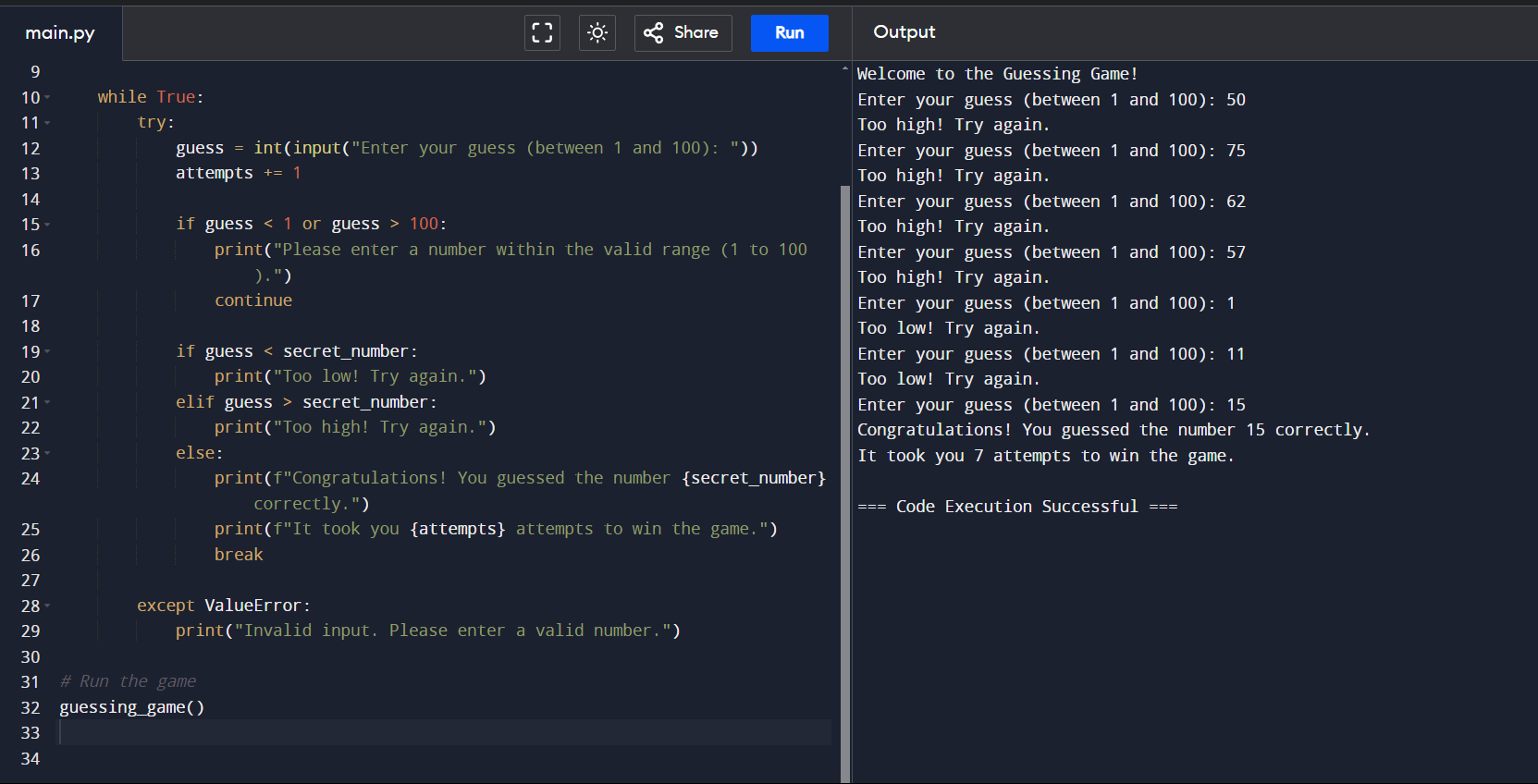
break

except ValueError:

print("Invalid input. Please enter a valid number.")

# Run the game

guessing\_game()



**Task-04**

**Comillas Negras PRODIGY INFOTECH**

**Implement a Sudoku Solver**

**Create a program that solves Sudoku puzzles automatically. The program should take an input grid representing an unsolved Sudoku puzzle and use an algorithm to fill in the missing numbers.**

**It should use backtracking or other suitable techniques to explore possible solutions and find the correct arrangement of numbers for the puzzle. Once solved, the program should display the completed Sudoku grid.**

def print\_grid(grid):

"""Utility function to print the Sudoku grid."""

for row in grid:

print(" ".join(map(str, row)))

def find\_empty\_location(grid):

"""Find empty location in the grid (cell with 0)."""

for row in range(9):

for col in range(9):

if grid[row][col] == 0:

return (row, col)

return None

def is\_safe(grid, row, col, num):

"""Check if it's safe to place 'num' in grid[row][col]."""

# Check row

if num in grid[row]:

return False

# Check column

for r in range(9):

if grid[r][col] == num:

return False

# Check 3x3 sub-grid

start\_row = row - row % 3

start\_col = col - col % 3

for r in range(3):

for c in range(3):

if grid[r + start\_row][c + start\_col] == num:

return False

return True

def solve\_sudoku(grid):

"""Solve Sudoku using backtracking."""

find = find\_empty\_location(grid)

if not find:

return True # Puzzle solved

row, col = find

for num in range(1, 10):

if is\_safe(grid, row, col, num):

grid[row][col] = num

if solve\_sudoku(grid):

return True

grid[row][col] = 0 # Backtrack

return False # Trigger backtracking

# Example Sudoku grid (0 represents empty cells)

grid = [

[5, 3, 0, 0, 7, 0, 0, 0, 0],

[6, 0, 0, 1, 9, 5, 0, 0, 0],

[0, 9, 8, 0, 0, 0, 0, 6, 0],

[8, 0, 0, 0, 6, 0, 0, 0, 3],

[4, 0, 0, 8, 0, 3, 0, 0, 1],

[7, 0, 0, 0, 2, 0, 0, 0, 6],

[0, 6, 0, 0, 0, 0, 2, 8, 0],

[0, 0, 0, 4, 1, 9, 0, 0, 5],

[0, 0, 0, 0, 8, 0, 0, 7, 9]

]

print("Input Sudoku grid:")

print\_grid(grid)

if solve\_sudoku(grid):

print("\nSolution:")

print\_grid(grid)

else:

print("\nNo solution exists.")

