**Cube Solver**

**Objective:-**

This is a python-based project designed to solve the classic Rubik’s Cube using the advanced algorithms provided by the Kociemba module. It has a command line interface to interact with the user. This program can be a helpful tool to beginners who are looking forward to solve their scrambled Rubik’s Cube

**Input and Output:-**This program is designed to interact with the user through a command line interface (CLI). The input scramble/moves will be taken as input and the solve formula will be displayed after a valid scramble for the cube is obtained. The solve will be demonstrated as well to better guide the user to solve the cube

**Features:-**1> The advanced Kociemba’s Two-Phase algorithm will fetch an optimum solution of 15-20 moves for any scramble of the classic Rubik’s cube

2> The program also has a cube playground for the user to scramble the Rubik’s Cube

3> The step-by-step demonstration of the solve formula can effectively guide the user to solve the cube

**Limitations:-**

1> The user may face difficulty in operating the program before going through the controls properly

2> The cases of corner-twists are not taken into account. Such scrambles will thus be labelled as invalid

**Dependencies:-**

The program makes use of a python module named “kociemba” which needs to be installed before the program can be run. It can be installed by the following command



**Code:-**

import time

import kociemba

class ARCS():

    cube = {"U" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

            "R" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

            "F" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

            "D" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

            "L" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

            "B" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]]}

    def \_\_init\_\_(*self*):

        pass

    def U(*self*, *i*):

        for k in range(0, *i*):

            ctr = "U"

            block = *self*.cube[ctr][2][2]

            (*self*.cube[ctr][0],

[*self*.cube[ctr][0][2], *self*.cube[ctr][1][2], *self*.cube[ctr][2][2]],

*self*.cube[ctr][2],

            [*self*.cube[ctr][0][0], *self*.cube[ctr][1][0], *self*.cube[ctr][2][0]]) = (

            [*self*.cube[ctr][2][0], *self*.cube[ctr][1][0], *self*.cube[ctr][0][0]],

*self*.cube[ctr][0],

            [*self*.cube[ctr][2][2], *self*.cube[ctr][1][2],

*self*.cube[ctr][0][2]], *self*.cube[ctr][2])

*self*.cube[ctr][2][0] = block

            for j in range(0, 3):

                (*self*.cube["L"][0][j], *self*.cube["F"][0][j],

*self*.cube["R"][0][j], *self*.cube["B"][0][j]) = (

*self*.cube["F"][0][j], *self*.cube["R"][0][j],

*self*.cube["B"][0][j], *self*.cube["L"][0][j])

    def D(*self*, *i*):

        for k in range(0, *i*):

            ctr = "D"

            block = *self*.cube[ctr][2][2]

            (*self*.cube[ctr][0],

            [*self*.cube[ctr][0][2], *self*.cube[ctr][1][2], *self*.cube[ctr][2][2]],

*self*.cube[ctr][2],

            [*self*.cube[ctr][0][0], *self*.cube[ctr][1][0], *self*.cube[ctr][2][0]]) = (

            [*self*.cube[ctr][2][0], *self*.cube[ctr][1][0], *self*.cube[ctr][0][0]],

*self*.cube[ctr][0],

            [*self*.cube[ctr][2][2], *self*.cube[ctr][1][2], *self*.cube[ctr][0][2]],

*self*.cube[ctr][2])

*self*.cube[ctr][2][0] = block

            for j in range(0, 3):

                (*self*.cube["L"][2][j], *self*.cube["F"][2][j],

*self*.cube["R"][2][j], *self*.cube["B"][2][j]) = (

*self*.cube["B"][2][j], *self*.cube["L"][2][j],

*self*.cube["F"][2][j], *self*.cube["R"][2][j])

    def L(*self*, *i*):

        for k in range(0, *i*):

            ctr = "L"

            block = *self*.cube[ctr][2][2]

            (*self*.cube[ctr][0],

            [*self*.cube[ctr][0][2], *self*.cube[ctr][1][2], *self*.cube[ctr][2][2]],

*self*.cube[ctr][2],

            [*self*.cube[ctr][0][0], *self*.cube[ctr][1][0], *self*.cube[ctr][2][0]]) = (

            [*self*.cube[ctr][2][0], *self*.cube[ctr][1][0], *self*.cube[ctr][0][0]],

*self*.cube[ctr][0],

            [*self*.cube[ctr][2][2], *self*.cube[ctr][1][2], *self*.cube[ctr][0][2]],

*self*.cube[ctr][2])

*self*.cube[ctr][2][0] = block

            for j in range(0, 3):

                (*self*.cube["U"][j][0], *self*.cube["F"][j][0],

*self*.cube["D"][j][0], *self*.cube["B"][2-j][2]) = (

*self*.cube["B"][2-j][2], *self*.cube["U"][j][0],

*self*.cube["F"][j][0], *self*.cube["D"][j][0])

    def R(*self*, *i*):

        for k in range(0, *i*):

            ctr = "R"

            block = *self*.cube[ctr][2][2]

            (*self*.cube[ctr][0],

            [*self*.cube[ctr][0][2], *self*.cube[ctr][1][2], *self*.cube[ctr][2][2]],

*self*.cube[ctr][2],

            [*self*.cube[ctr][0][0], *self*.cube[ctr][1][0], *self*.cube[ctr][2][0]]) = (

            [*self*.cube[ctr][2][0], *self*.cube[ctr][1][0], *self*.cube[ctr][0][0]],

*self*.cube[ctr][0],

            [*self*.cube[ctr][2][2], *self*.cube[ctr][1][2], *self*.cube[ctr][0][2]],

*self*.cube[ctr][2])

*self*.cube[ctr][2][0] = block

            for j in range(0, 3):

                (*self*.cube["U"][j][2], *self*.cube["F"][j][2],

*self*.cube["D"][j][2], *self*.cube["B"][2-j][0]) = (

*self*.cube["F"][j][2], *self*.cube["D"][j][2],

*self*.cube["B"][2-j][0], *self*.cube["U"][j][2])

    def F(*self*, *i*):

        for k in range(0, *i*):

            ctr = "F"

            block = *self*.cube[ctr][2][2]

            (*self*.cube[ctr][0],

            [*self*.cube[ctr][0][2], *self*.cube[ctr][1][2], *self*.cube[ctr][2][2]],

*self*.cube[ctr][2],

            [*self*.cube[ctr][0][0], *self*.cube[ctr][1][0], *self*.cube[ctr][2][0]]) = (

            [*self*.cube[ctr][2][0], *self*.cube[ctr][1][0], *self*.cube[ctr][0][0]],

*self*.cube[ctr][0],

            [*self*.cube[ctr][2][2], *self*.cube[ctr][1][2], *self*.cube[ctr][0][2]],

*self*.cube[ctr][2])

*self*.cube[ctr][2][0] = block

            for j in range(0, 3):

                (*self*.cube["U"][2][j], *self*.cube["R"][j][0],

*self*.cube["D"][0][2-j], *self*.cube["L"][2-j][2]) = (

*self*.cube["L"][2-j][2], *self*.cube["U"][2][j],

*self*.cube["R"][j][0], *self*.cube["D"][0][2-j])

    def B(*self*, *i*):

        for k in range( 0, *i*):

            ctr = "B"

            block = *self*.cube[ctr][2][2]

            (*self*.cube[ctr][0],

            [*self*.cube[ctr][0][2], *self*.cube[ctr][1][2], *self*.cube[ctr][2][2]],

*self*.cube[ctr][2],

            [*self*.cube[ctr][0][0], *self*.cube[ctr][1][0], *self*.cube[ctr][2][0]]) = (

            [*self*.cube[ctr][2][0], *self*.cube[ctr][1][0], *self*.cube[ctr][0][0]],

*self*.cube[ctr][0],

            [*self*.cube[ctr][2][2], *self*.cube[ctr][1][2], *self*.cube[ctr][0][2]],

*self*.cube[ctr][2])

*self*.cube[ctr][2][0] = block

            for j in range(0, 3):

                (*self*.cube["U"][0][j], *self*.cube["R"][j][2],

*self*.cube["D"][2][2-j], *self*.cube["L"][2-j][0]) = (

*self*.cube["R"][j][2], *self*.cube["D"][2][2-j],

*self*.cube["L"][2-j][0], *self*.cube["U"][0][j])

    def move\_cube(*self*, *moves*):

        move = *moves*

        move\_no = int(move[1])

        if move[0] == "U":

*self*.U(move\_no)

        elif move[0] == "R":

*self*.R(move\_no)

        elif move[0] == "F":

*self*.F(move\_no)

        elif move[0] == "D":

*self*.D(move\_no)

        elif move[0] == "L":

*self*.L(move\_no)

        elif move[0] == "B":

*self*.B(move\_no)

        else:

            pass

    def scramble(*self*):

*self*.display()

        print()

        try:

            print("1> Solve")

            print("2> Return")

            print()

            mv = input("Enter move : ")

            print()

            if len(mv) < 3:

                if len(mv) == 1:

                    if mv == "1":

*self*.solve()

                    elif mv == "2":

*self*.main()

                    else:

                        mv += "1"

                else:

                    mv = mv.replace("\'", "3")

                l = []

                for i in range(1, 4):

                    for j in ["L", "R", "U", "D", "F", "B"]:

                        l.append(str(j + str(i)))

                if mv in l:

*self*.move\_cube(mv)

                    print()

                else:

                    raise ValueError

            else:

                raise ValueError

        except ValueError:

            print("Invalid Input, Try Again")

*self*.scramble()

    def solve(*self*):

        cube = *self*.set\_clear\_state()

        cube\_scramble = *self*.get\_cube\_state()

        try:

            solve\_formula = kociemba.solve(cube\_scramble)

        except ValueError:

            print("Invalid Input, Re-enter scramble")

            print()

*self*.cube = cube

*self*.get\_cube\_scramble()

        formula = solve\_formula.split()

        cube = *self*.set\_solved\_state(cube)

        if *self*.cube == cube:

            print("Cube Already Solved !!!")

            print()

*self*.main()

        else:

            for i in range(0, len(formula)):

                j = formula[i].replace("\'", "3")

                if len(j) == 1:

                    j += "1"

                print()

                print(formula[i])

                print()

*self*.move\_cube(j)

*self*.display()

                time.sleep(1)

            print("Formula used :", *end*=" ")

            for j in formula:

                print(j, *end*=" ")

            print()

            print()

*self*.main()

    def menu(*self*):

        print("         ~~~ARCS~~~         ")

        print("Advanced Rubik's Cube Solver")

        print()

        print("1> Scramble a Cube")

        print("2> Solve a Cube")

        print("3> Controls")

        print("4> Exit")

        print()

    def set\_cube\_state(*self*, *cube\_str*):

        x = 0

        if type(*cube\_str*) == type(list()):

            cube\_l = *cube\_str*

*cube\_str* = ""

            for i in cube\_l:

*cube\_str* += i

        l = ["U", "R", "F", "D", "L", "B"]

        for i in l:

            for j in range(0, 3):

                for k in range(0, 3):

*self*.cube[i][j][k] = *cube\_str*[x]

                    x += 1

    def get\_cube\_state(*self*):

        cube\_str = ""

        l = ["U", "R", "F", "D", "L", "B"]

        for i in l:

            for j in range(0, 3):

                for k in range(0, 3):

                    cube\_str += *self*.cube[i][j][k]

        l = [("Y", "U"), ("R", "R"), ("B", "F"), ("W", "D"), ("O", "L"), ("G", "B")]

        for i in l:

            cube\_str = cube\_str.replace(i[0], i[1])

        return cube\_str

    def get\_cube\_scramble(*self*):

        cube\_str = *self*.get\_cube\_state()

        cube\_l = cube\_str

        cube\_str = []

        for i in cube\_l:

            cube\_str.append(i)

        i = 0

        while i < 54:

            while True:

                cube\_str[i] = "\_"

*self*.set\_cube\_state(cube\_str)

*self*.display()

                print("1> Back")

                print("2> Return")

                print()

                ch = input("Enter color : ")

                ch = ch.upper()

                print()

                try:

                    if ch in ["Y", "R", "B", "W", "O", "G", "1", "2"]:

                        if ch == "1":

                            cube\_str[i] = " "

                            if i == 0:

                                i -= 1

                            else:

                                i -= 2

                        elif ch == "2":

*self*.main()

                        else:

                            cube\_str[i] = ch

                        break

                    else:

                        raise ValueError

                except ValueError:

                    print("Invalid Input, Try Again")

                    print()

                    continue

            i += 1

*self*.set\_cube\_state(cube\_str)

*self*.display()

        print()

*self*.solve()

    def set\_clear\_state(*self*):

        return {"U" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

                "R" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

                "F" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

                "D" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

                "L" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]],

                "B" : [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]]}

    def set\_solved\_state(*self*, *cube*):

        l = ["U", "R", "F", "D", "L", "B"]

        col = ["Y", "R", "B", "W", "O", "G"]

        for i in range(0, 6):

            for j in range(0, 3):

                for k in range(0, 3):

*cube*[l[i]][j][k] = col[i]

        return *cube*

    def controls(*self*):

        print("~~~Moves~~~")

        for i in ["U", "R", "F", "D", "L", "B"]:

            print(f'{i}     {i}2     {i}\'')

        print()

        print("~~~Colors~~~")

        print("Y - yellow   |   R - Red")

        print("B - Blue     |   W - White")

        print("O - Orange   |   G - Green")

        print()

*self*.main()

    def display(*self*):

        for i in range(0, 3):

            print(" "\*13, *end* = "")

            for j in range(0, 3):

                print(f"[{*self*.cube['U'][i][j]}]", *end* = " ")

            print()

        print()

        for i in range(0, 3):

            for j in range(0, 3):

                print(f"[{*self*.cube['L'][i][j]}]", *end* = " ")

            print(*end* = " ")

            for j in range(0, 3):

                print(f"[{*self*.cube['F'][i][j]}]", *end* = " ")

            print(*end* = " ")

            for j in range(0, 3):

                print(f"[{*self*.cube['R'][i][j]}]", *end* = " ")

            print(*end* = " ")

            for j in range(0, 3):

                print(f"[{*self*.cube['B'][i][j]}]", *end* = " ")

            print()

        print()

        for i in range(0, 3):

            print(" "\*13, *end* = "")

            for j in range(0, 3):

                print(f"[{*self*.cube['D'][i][j]}]", *end* = " ")

            print()

        print()

    def main(*self*):

*self*.menu()

*self*.cube = *self*.set\_clear\_state()

        try:

            ch = input("Enter your choice : ")

            ch = ch.strip()

            print()

            if ch == "1":

*self*.cube = *self*.set\_solved\_state(*self*.cube)

*self*.scramble()

            elif ch == "2":

*self*.get\_cube\_scramble()

            elif ch == "3":

*self*.controls()

            elif ch == "4":

                exit()

            else:

                raise ValueError

        except ValueError:

            print("Invalid Input, Try Again")

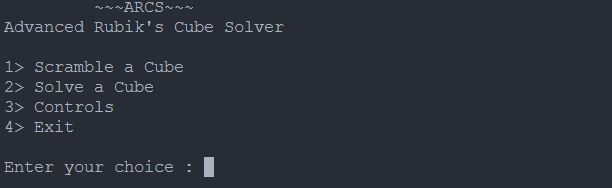
*self*.main()

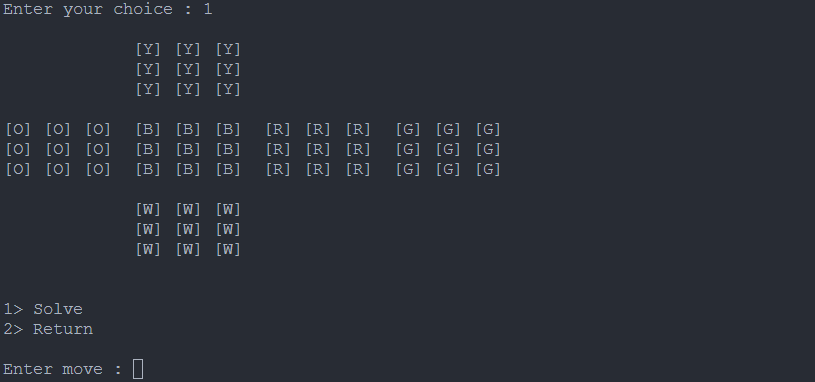
if \_\_name\_\_ =="\_\_main\_\_":

    ARCS().main()

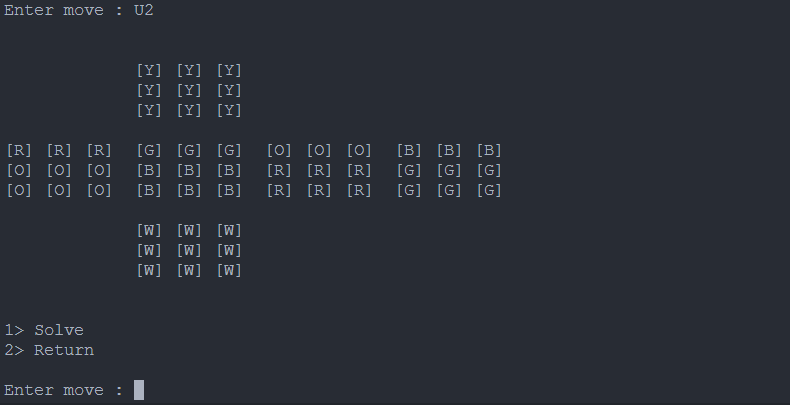
**Output:-**

This is how the output will look like

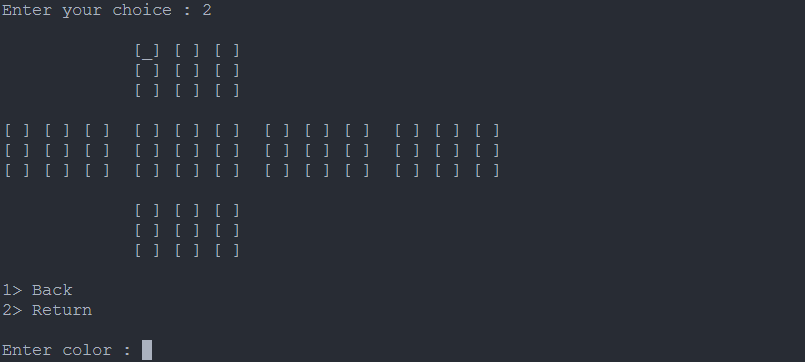
when we enter “1” as our choice, it opens the cube playground

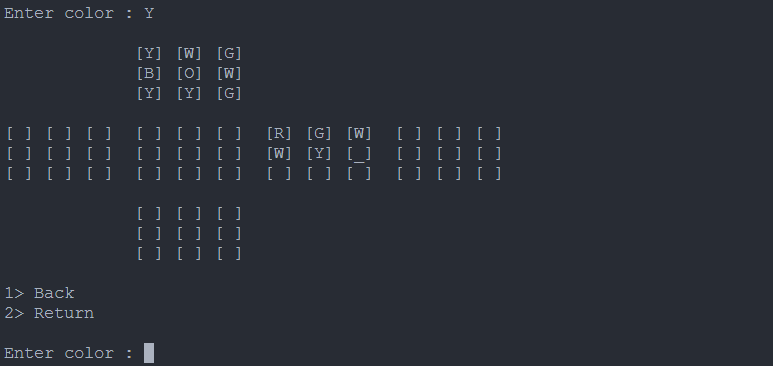


we can make various moves. I have made the move “U2” as an example

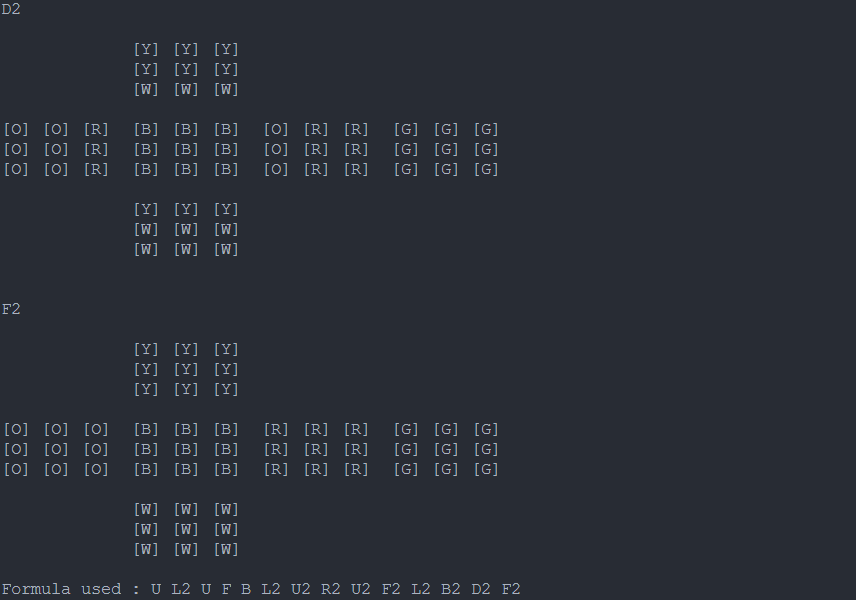


when we give “2” as our choice, it opens an empty cube, where we have to input the scramble. The empty cube will fill up as per our input.

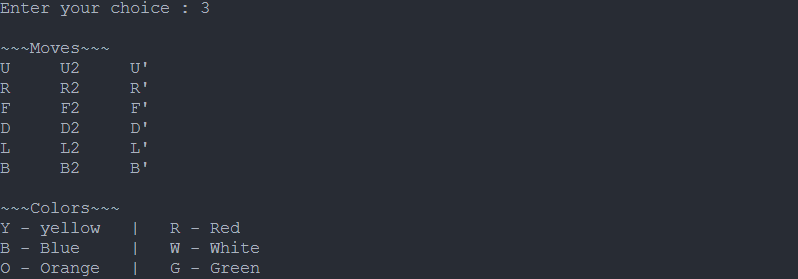




after the cube is completely filled, it will demonstrate the solve step by step



when we give “3” as our choice, it displays the controls



this is an overall sample output of the program



and this message will be displayed every time we give a wrong input

**Python Version:-**

The program is written in python 3.12.0

**Hardware Specifications:-**

Hardware specifications are as follows :

Processor : Intel(R) Core(TM) i5-10210U CPU @1.60GHz 2.11 GHz

Ram : 8.00 GB

Windows Version : 22H2

OS Build : 22621.2134