My Rubik's Cube implementation in Python builds on top of the skeleton program provided and enables the mechanism described by the problem. In order to adapt to the requirement, certain changes had to be made in the class functions arguments. The primary one is distinguishing between the default solved state of the cube and a separate provided string. My motivation for this was the implementation of the random function needing to reapply the random sequence of moves to the initial permuted state. As such, the random function is the only one changing the cube's self.string parameter from the default solved value. Conversely, the functions norm, goal, applyMovesStr, and print either use the provided state string or clone the default state in order to return the desired output. The two-dimensional representation of the state has been implemented through a 24x24 matrix. In order to map the sticker indices to the required position, I added an additional indices dictionary to indicate the string character's position in the matrix, all of which are used in the norm function. The print function simply takes the default's normalized state, or normalizes the given state if the argument is provided, and adds some further cosmetic changes, such as when to add a space and when not to, and prints out the desired output. Other additions to the skeleton program are moves_list containing all possible moves in order to iterate through them when generating random moves and the is_proper function which validates the inputted arguments and prints out an error if it fails the test.
The goal function has been implemented to consider any combination of face colors as a solution, not just the provided default of "WWWW RRRR GGGG YYYY OOOO BBBB". For example, "RRRR WWWW GGGG YYYY OOOO BBBB" would also be acceptable. For the shuffle function, I opted to randomly generate one move at a time and apply it to the state rather than generate N moves and feeding them into applyMovesStr. Finally, for the random function, I decided to use some of the same functionality present in the shuffle function due to the need to record the sequence of moves leading to a goal state, the permutations of the cube, as well as having to account for the goal state being reached before the completion of the entire set of provided moves. Using the shuffle function in the implementation of the random function would have required modifications to it which were not specified in its implementation. The last noteworthy documentation is the validation of the string input inside of the command logic implementation rather than in the class functions.
I have tested my code both with the commands given as an example in the problem layout, as well as with my own input, both locally and on tux. The results are the following:

```
iustintoader@Iustins-MBP RC part1 % sh run.sh print
   WW
   WW
OO GG RR BB
OO GG RR BB
   YY
   YY
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh printsh run.sh
print "RWOR GOYB BOGW BRBW YWYO RGYG"
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh print "RWOR GOYB
BOGW BRBW YWYO RGYG"
   RW
   OR
YW BO GO RG
YO GW YB YG
   BR
   BW
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh print "RWOR GOYB
BOGW BRBW YWYO RGY"
Wrong input string
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh print "RWOR GOYB
BOGW BRBW YWYO RGYY"
Wrong input string
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh print "RWOR GOYB
BOGW BRBW YWYORGYY"
Wrong input string
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh goal "RWOR GOYB
BOGW BRBW YWYO RGYG"
False
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh goal "WWWW RRRR
GGGG YYYY OOOO BBBB"
True
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh goal "WWWW RRRR
GGGG YYYY BBBB OOOO"
True
iustintoader@Iustins-MBP RC part1 % sh run.sh applyMovesStr "R U' R'"
"WWWW RRRR GGGG YYYY OOOO BBBB"
   GW
   WR
WB OG YR BR
OO GW GR BB
   YO
   YY
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh shuffle 10
D' D U' R R U R' L' U D'

   GR
   BO
YW RG WY BO
YY GG WW OO
   RR
   BB
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh applyMovesStr "D'
D U' R R U R' L' U D'"
   GR
   BO
YW RG WY BO
YY GG WW OO
   RR
   BB
iustintoader@Iustins-MBP RC part1 % sh run.sh random "L D' R' F R D'"
6 10
```

```
D L' D' B U L'

   BW
   GW
OY RR BB OY
WR BY OO WG
   YG
   RG

   BW
   GW
OY RR BB OY
WG WR BY OO
   RY
   GG

   RW
   WW
YG RR BB OG
OW GR BY OB
   OY
   YG

   RW
   WW
YG RR BB OG
GR BY OB OW
   YG
   OY

   BB
   WW
WG RR BY OO
RR BY OO WG
   YG
   YG

   WB
   WB
RR BY OO WG
RR BY OO WG
   YG
   YG

   BB
   BB
RR YY OO WW
RR YY OO WW
   GG
   GG
```

```
52274
2.3
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh applyMovesStr "L
D' R' F R D'"
   BW
   GW
OY RR BB OY
WR BY OO WG
   YG
   RG
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh applyMovesStr "L
D' R' F R D' D L' D' B U L'"
   BB
   BB
RR YY OO WW
RR YY OO WW
   GG
   GG
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh random "L' B' U'
D L' F B" 7 10
F F D' D' B

   GG
   BB
WW RR YY RR
BB OO GG OO
   YY
   WW

   GG
   BW
WY OR BY RR
BY OR BG OO
   GY
   WW

   GG
   YY
WG OO BY RR
BY RR WG OO
   BB
   WW

   GG
   YY
WG OO BY RR
RR WG OO BY
   BW
   BW
   GG
   YY
WG OO BY RR
WG OO BY RR
   WW
   BB

   YY
   YY
GG OO BB RR
GG OO BB RR
   WW
   WW

2733
0.14
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh applyMovesStr "L'
B' U' D L' F B"
   GG
   BB
WW RR YY RR
BB OO GG OO
   YY
   WW
(base) iustintoader@Iustins-MBP RC part1 % sh run.sh applyMovesStr "L'
B' U' D L' F B F F D' D' B"
   YY
   YY
GG OO BB RR
GG OO BB RR
   WW
   WW
```