A Security Device

GitHub: https://github.com/zeiadamin/A-Security-Device

The program simulates a **Security System** for an Office or a Home.

The System gets Locked after entering the lock passcode: **561761**, and it gets unlocked after entering the unlock code: **561764.**

If the user entered a **wrong lock passcode** the system **won't lock** and if the user entered a **wrong unlock passcode** the system **won't unlock**.

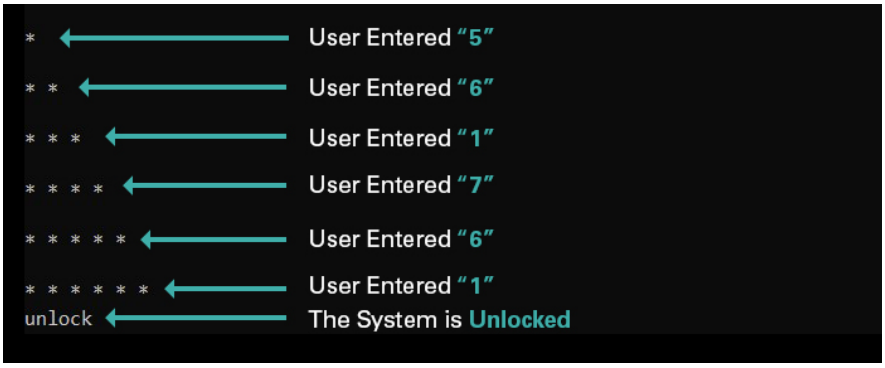
If the user entered a **correct unlock code**, the system would give the message **"Unlock".**

If the user entered a correct lock code, the system would give the message **"Lock".**

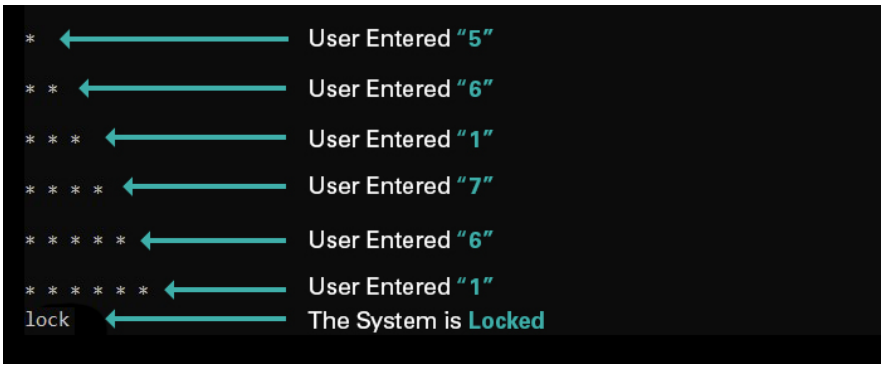
The System **does not show the characters that the user entered,** instead it shows **"*"** symbol indicating that the user entered a character.

If the user entered **non-valid characters** (any characters other than numbers from 0 to 9), the system would simply ignore them and gives the message **"Wrong Character".**

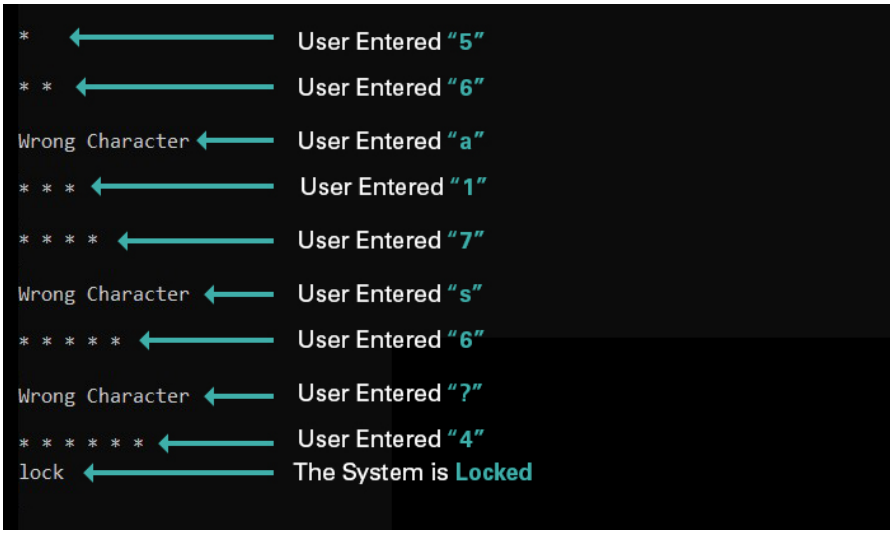
Let's assume that a user tried to enter a correct unlock passcode "561761":



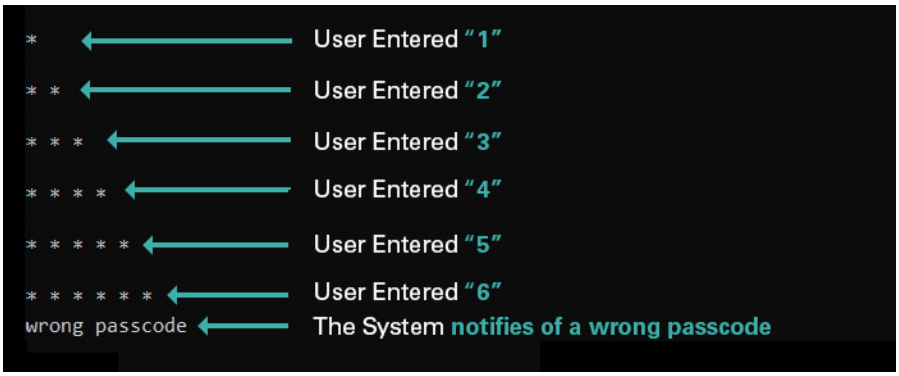
Let's assume that a user tried to enter a correct lock passcode "561764":



Let's assume that a user tried to enter the correct lock code, but he entered invalid characters within the passcode "56a17s6?4":

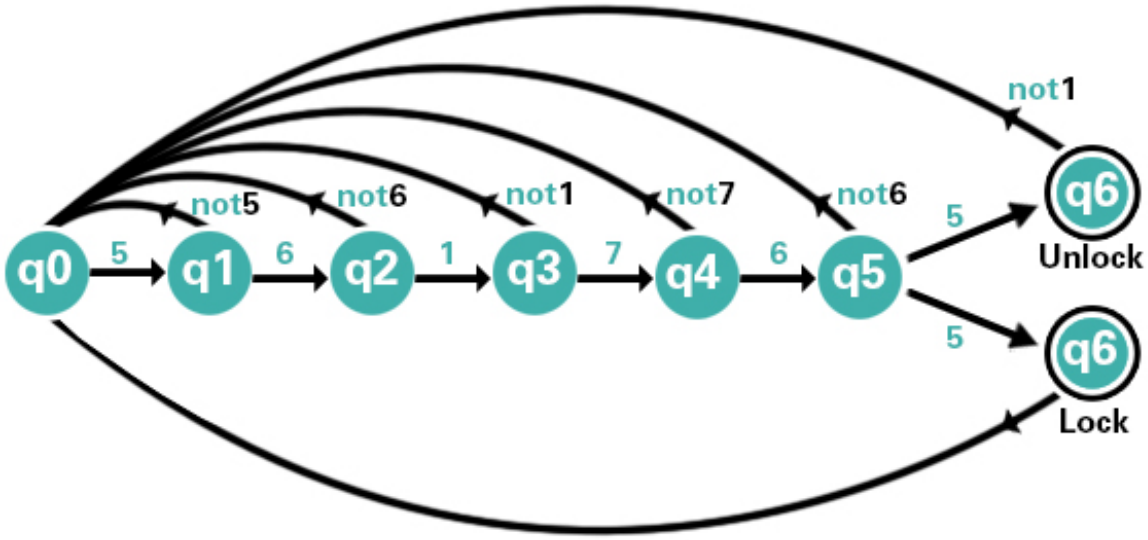


Let's assume that a user tried to enter an incorrect code "123456":



The Logic:

The Project use **Finite Automata Concepts** to achieve its purpose. Where the systems start with an **initial state** of locked **q0** the when the user enters a correct key it changes it's state up to q5. If it reaches **q6.**



That means that the user entered the correct key for **transitioning from the state 6 times in a row**. These keys are the passcode for the security system.

If at any point the user entered an incorrect **key.** The Transition function will move it to the state **q0** again to **start from the beginning**.

The Finite Automata consists of an **initial state q0** and a **list of states Q= [q0, q1, q2, q3, q4, q5, q6L, q6UL]**

It also consists of a list of **possible keys K= [5,6,1,7,6,1,4]**

And a **Transition Function Q x K -> q**

The Implementation:

```
import random
import getpass
#We need to design a lock system where the user keep entering characters and these
characters will unlock the system
#if the user enter the correct code in a row

#Ex:
#9135203344124510334441239700011123344513344101O1
#  unlock--^         ^--lock        unlock--^

#561761 is the unlocking code
#561764 is the locking code

#_____________________________________________________________________________
#This will happen taking a Finite Automata approach where there are states and keys
and if the key is correct
#the state will move to the next

#_____________________________________________________________________________
#To Achieve that we need to keep track of the current_State (q0,q1,q2,q3,q4,q5) and
we also need to keep track if
#the system is locked or unlocked let's  call that the lock_State

#_____________________________________________________________________________
#To Start we need to get input from the user
#input_string = input()

#we need to set initial values for the current_state and the lock_state let's assume
that the initial state of the
#system is locked
#0 is locked and 1 is unlocked
current_state = 0

#the possible lock states are (0,1,2,3,4,5,6) where when you achieve 6 (up the
ladder) and when that happen you
#switch the current_state and then you reset the lock_state to 0
lock_state = 0

#now we need to start thinking about the logic of the of the code itself
#we need to make sure that whenever we have a new input we need to check if the
input is correct or not and change
#the state of the current_State and the lock_State as needed
#to handle this we need to have a function that handles this change of state and
also handles the input checking

#Transition function that takes the user input as an argument and deals with the
changes in the current_State
#and lock_State

#A transition function that keeps track of the input key and the lock state and
changes the current lock based on the
#Finite Automata logic
def transition (char):
   global lock_state
   global current_state
   if lock_state == 0 and char == "5":
```

```
       #now we are in the state q0 if the key is correct:6 then incement the state
and move to state q1
       lock_state += 1
       #print (lock_state)
   elif lock_state == 1 and char == "6":
       #now we are in the state q1 if the key is correct:6 then incement the state
and move to state q2
       lock_state += 1
       #print (lock_state)
   elif lock_state == 2 and char == "1":
       #now we are in the state q2 if the key is correct:6 then incement the state
and move to state q3
       lock_state += 1
       #print (lock_state)
   elif lock_state == 3 and char == "7":
       #now we are in the state q3 if the key is correct:6 then incement the state
and move to state q4
       lock_state += 1
       #print (lock_state)
   elif lock_state == 4 and char == "6":
       #now we are in the state q4 if the key is correct:6 then incement the state
and move to state q5
       lock_state += 1
       #print (lock_state
   elif lock_state == 5 and char == "1":
       #now we are in the state q5 if the key is correct:6 then incement the state
and move to state q6UNLOCK
       lock_state = 0
       #print (lock_state)
       if current_state == 0:
        change_lock() #a function that changes the current_state and print the the
change in state
       else:
        print ("unlock")

   elif lock_state == 5 and char == "4":
       #now we are in the state q5 if the key is correct:6 then incement the state
and move to state q6UNLOCK
       lock_state = 0
       #print (lock_state)
       if current_state == 1:
        change_lock() #a function that changes the current_state and print the the
change in state
       else:
        print ("lock")

   else:
       #else then the key is wrong and the state is reset to q0
       lock_state = 0
       #print (lock_state)

#A function that changes the current_State and print the the change in state
def change_lock ():
   global current_state
   if current_state == 0:
       current_state = 1
       print ("unlock")
   else:
```

```
        current_state = 0
        print ("lock")


##############################################
#THE KEYPAD
##############################################

#A list that keeps track of actual inputs
input_list = []
#a list that keep track of number of inputs till 6 and prints *s
print_list = []

#A loop that is always working
while True:
    #getting the user keys
    user_input = getpass.getpass('')

    #ignoring input other than keys from 1 to 9
    if user_input in ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]:
        input_list.append(user_input)
        print_list.append("*")
        print (*print_list)
        transition(user_input)
    else:
        #in case of a wrong input print wrong character
        print ("Wrong Character")

    #if the 6 digits are entrered but they are wrong then print wrong passcode
    if len(input_list) == 6:
        if input_list != ["5","6","1","7","6","1"]:
            if input_list != ["5","6","1","7","6","4"]:
                print ("wrong passcode")
        input_list = []
        print_list = []
```

Test Table:

Test Cases	Example	Output
Testing a correct unlock code	561761	unlock
Testing a correct lock code	561764	lock
Testing an invalid key	agdjeb	Wrong Character
Testing a valid key but with an incorrect code	123456	Wrong Passcode
Testing an invalid passcode	253hds	Wrong Passcode
Testing an invalid passcode	53kd67	Wrong Passcode
Testing an invalid passcode	364932	Wrong Passcode
Testing an invalid passcode	34dfg3	Wrong Passcode
Testing all zeros	000000	Wrong Passcode
Testing all 1s	111111	Wrong Passcode
Testing the unlock code twice	561761 then 561761	Unlock then unlock
Testing the lock code twice	561764 then 561764	Lock then lock
Testing Unlock then lock	561761 then 561764	Unlock then lock
Testing lock the unlock	561764 then 561761	Lock the unlock
Testing Unlock with invalid characters	561h7u61	Unlock (wrong character messages in the middle)
Testing lock with invalid characters	561h7u64	lock (wrong character messages in the middle)