Password Check Program

<u>Brief Description:</u>
This project involves designing and implementing an authentication library for a website supporting local users. The library will enforce a password policy and manage access control for registered users. The requirements include writing a command-line program in a language like Python 3, with functionality to authenticate users and change passwords. The program must store user data persistently in a SHA1 hashed format, using a file named **password_check.dat**. The interfaces for authentication and password changes must adhere to specified command-line parameters, returning appropriate messages based on success or failure.

Before change to Susan in the .dat file

```
password_check.dat                                   Free Mode
~/Desktop/password_check.dat
1  peter,003d2065d2f391858754907036613884 14f195cf
2  susan,003d2065d2f391858754907036613884 14f195cf
3  edmond,003d2065d2f391858754907036613884 14f195cf
4
```

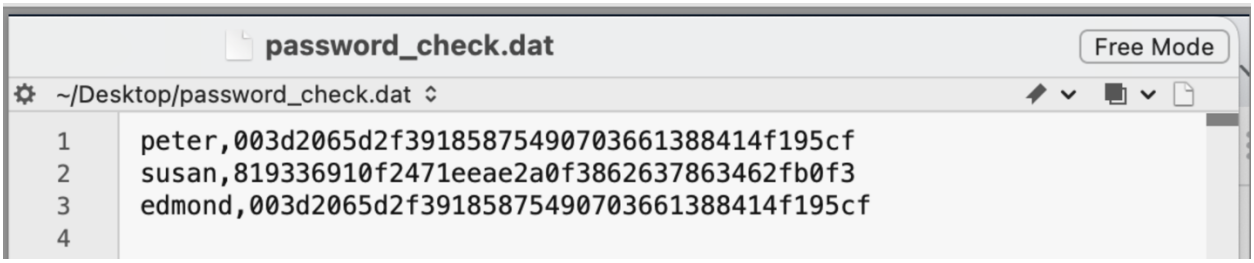
Testing both interfaces

```
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py Authenticate susan PASSword123
Success
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py Authenticate susan PASSword
Error: wrong password
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py Authenticate rachel PASSword123
Error: no such user
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword rachel PASSword123 PASSword456
Error: no such user
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword456 PASSword999
Error: wrong password
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword123 PASSword456
Success
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword456 PASSword123
Success
```

Testing password constraints

```
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword123 pass
Error: password policy failure: must be a minimum of  8  digits.
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword123 passwordistoolong
Error: password policy failure: exceeds maximum:  12  digits.
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword123 PASSword123!
Error: password policy failure: must not contain any special characters.
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword123 PASSwordis
Error: password policy failure: must contain a min 1 digit/s.
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword123 Password123
Error: password policy failure: must contain a min of 2 uppercase char.
joraelkayla@Joraels-MacBook-Air Desktop % python3 password_check_CSIA450_Jorael_Jamison.py ChangePassword susan PASSword123 PASSWORD123
Error: password policy failure: must contain a min of 3 lowercase char.
joraelkayla@Joraels-MacBook-Air Desktop %
```

Successful change to Susan in the .dat file



<u>Python Script</u>

```
import csv
import sys
import hashlib

# Coder: Jorael Jamison
# CSIA 450 - Cyber Security Capstone
# Professor Matt Boehnke
# May 5th, 2023


# Define Password Constants
min_password_length = 8
max_password_length = 12
min_lowercase = 3
min_uppercase = 2
min_digits = 1
#special_char = isalnum()

# Allow ability to call specific interface at command line
def main():
    if sys.argv[1] == "Authenticate":
        user = sys.argv[2]
        password = sys.argv[3]

        auth = Authenticate(user, password)
        auth.open_file()

    elif sys.argv[1] == "ChangePassword":
        user = sys.argv[2]
        old_password = sys.argv[3]
        new_password = sys.argv[4]
```

```
        auth = ChangePassword(user, old_password, new_password)
        auth.open_file()


# INTERFACE 1 - Authenticate(“user”, “password”)
class Authenticate:
    def __init__(self, user, password):
        self.user = user
        self.password = password

    def open_file(self):
        with open("password_check.dat", "r") as file:
            file_reader = csv.reader(file)
            self.user_find(file_reader)
            file.close()

# Searches file to verify user is found, if so, saves SHA1 HASH as correct_pass
    def user_find(self, file):
        for row in file:
            if len(row) >= 2 and row[0] == self.user:
                correct_pass = row[1]
                self.pass_check(correct_pass)
                break
            else:
                continue
                break
        else:
            print("Error: no such user")
            exit()

# Takes the user input and generates SHA1 HASH, checks against correct_pass
    def pass_check(self, correct_pass):
        user_input = self.password
        h = hashlib.new("SHA1")
        h.update(user_input.encode())
        input_hash = h.hexdigest()
        if correct_pass == input_hash:
            print("Success")
        else:
            print("Error: wrong password")


# INTERFACE 2 - ChangePassword(“user”, “old_password”, "new_password")
```

```
class ChangePassword:
  def __init__(self, user, old_password, new_password):
    self.user = user
    self.old_password = old_password
    self.new_password = new_password
    self.password_constraints()

  def open_file(self):
    with open("password_check.dat", "r") as file:
      file_reader = csv.reader(file)
      self.user_find(file_reader)
      file.close()

# Searches file to verify user is found, if so, saves SHA1 HASH as correct_pass
  def user_find(self, file):
    for row in file:
# Confirms password file has two rows user and password hash
      if len(row) >= 2 and row[0] == self.user:
        correct_pass = row[1]
        self.input_hash = self.pass_check(correct_pass)
        self.new_pass_hash = self.new_pass()
        self.change_password(self.new_pass_hash, correct_pass, self.input_hash)
        break
    else:
      print("Error: no such user")
      exit()

# Checks new_password to ensure follows all password constraints
  def password_constraints(self):
    if len(self.new_password) < min_password_length:
      print("Error: password policy failure: must be a minimum of ", min_password_length, "
digits.")
      exit()
    if len(self.new_password) > max_password_length:
      print("Error: password policy failure: exceeds maximum: ", max_password_length, "
digits.")
      exit()
    lowercase_count = 0
    uppercase_count = 0
    for char in self.new_password:
        if char.islower():
            lowercase_count += 1
        elif char.isupper():
            uppercase_count += 1
```

```
        if lowercase_count < min_lowercase:
            print("Error: password policy failure: must contain a min of", min_lowercase,
"lowercase char.")
            exit()
        if uppercase_count < min_uppercase:
            print("Error: password policy failure: must contain a min of", min_uppercase,
"uppercase char.")
            exit()
        if not any(char.isdigit() for char in self.new_password):
            print("Error: password policy failure: must contain a min", min_digits, "digit/s.")
            exit()
# Checks if any characters are not alphanumeric.
        if not self.new_password.isalnum():
            print("Error: password policy failure: must not contain any special characters.")
            exit()

# Takes the user input and generates SHA1 HASH, checks against correct_pass
    def pass_check(self, correct_pass):
        user_input = self.old_password
        h = hashlib.new("SHA1")
        h.update(user_input.encode())
        input_hash = h.hexdigest()
        if correct_pass == input_hash:
            print("Success")
            return input_hash
        else:
            print("Error: wrong password")
            exit()

    def new_pass(self):
        user_input = self.new_password
        h = hashlib.new("SHA1")
        h.update(user_input.encode())
        new_pass_hash = h.hexdigest()
        return new_pass_hash


# Replaces the old password hash with the new password hash in the digest
    def change_password(self, new_pass_hash, correct_pass, input_hash):
        if correct_pass == input_hash:
            with open("password_check.dat", "r") as file:
                file_reader = csv.reader(file)
                rows = list(file_reader)
```

```
# Find the row that corresponds to the user
        for row in rows:
            if row[0] == self.user:
# Update the row with the new password hash
                row[1] = new_pass_hash
                break
        else:
            print("Error: no such user")
            return

# Write the updated password to the file digest
        with open("password_check.dat", "w") as file:
            file_writer = csv.writer(file)
            file_writer.writerows(rows)

main()
```