SYSC 4810 Assignment

All functional code is in assignment.py. All test code is in assignment_test.py

Problem 1

a) The access control model that I will use is RBAC, since the access control that a given user has is determined purely by their role

b) See permissions matrix below based on justInvest's access control policy:

Operations

0. Access system outside of business hours
1. View account balance
2. View investment portfolio
3. Modify investment portfolio
4. View Financial Advisor contact info
5. View Financial Planner contact info
6. View money market instruments
7. View private consumer instruments

T if user with that role can access, otherwise they can't

Role/ Operation	0	1	2	3	4	5	6	7
Client	T	T	T		T			
Premium Client	T	T	T	T	T	T		
Financial Advisor	T	T	T	T				T
Financial Planner	T	T	T	T			T	T
Teller		T	T					

c) See access_control() and ACCESS_CONTROL_POLICY and see Access_Control_TestCase in assignment_test. Test methodology:

i) test_access_granted(): make sure that user with each role could access the operations they were allowed to access

ii) test_access_denied(): make sure that user with each role could not access the operations they were not allowed to access

Problem 2

a) I decided to use argon2 as the hash function, since in my research it is incredibly secure. It won the Password Hashing Competition and comes with secure defaults for hash length and salt length. It also generates the salt itself and the documentation recommends allowing it to do that instead of using a custom salt. The hash length I will be using is 32 bytes (256 bits) with a salt length of 16 bytes (128 bits)
b) The password file will have the following data: username, hash, role
c) See functions write_user_to_file() and get_user_from_file()
d) See Write_and_Get_File_TestCase. One simple test was enough to demonstrate the functions correctly write to and get from the file

Problem 3

a) See launch_signup()
b) See proactive_password_checker()
c) I wasn't able to do a unit test for launch_signup() because mocking user input for inputting password was especially difficult, since it takes in keyboard events rather than characters. However I tested launch_signup() many times manually when I ran the system with different usernames, passwords and roles.
   For proactive_password_checker(), see Proactive_Password_Checker_TestCase. I tested one case where the input was valid and 7 different cases where the input was invalid based on the 7 disqualifying characteristics:
   1. Password too short
   2. Password was too long
   3. Password is a common password
   4. Password does not contain a number
   5. Password does not contain a lower-case character
   6. Password does not contain an upper-case character
   7. Password is the same as the username

Problem 4

a) See authenticate user()
b) See display_access()
c) I wasn't able to do a unit test for authenticate user() because mocking user input for inputting password was especially difficult, since it takes in keyboard events rather than characters. However I tested authenticate user() many times manually when I ran the system with different usernames and passwords
   For display_access(), see Display_Access_TestCase. I tested that the output was valid for all the system roles