Unit 12: Python Project

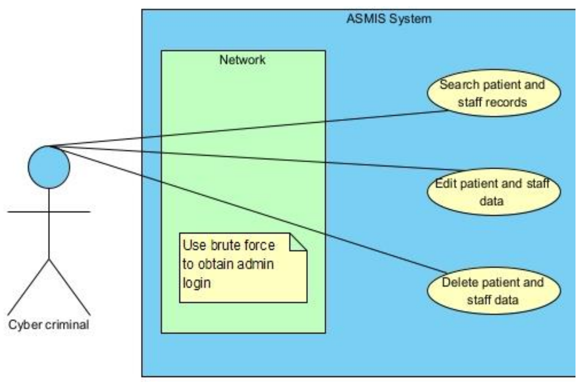
Contents:

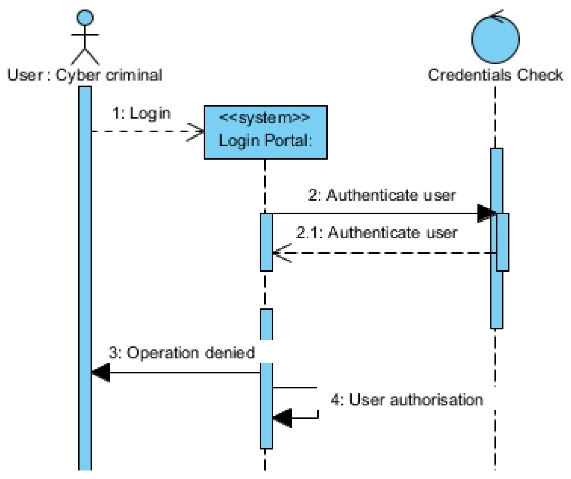
Abstract:

The following report project aims to demonstrate solutions with critical evaluation to solve or mitigate security issues that may arise from introducing ASMIS at Queen's Medical Centre. After initially providing context and use case scenario, solutions are offered in python code to implement the security measures. The code will then be tested and evaluated.

Context and use case 1:

The threat is that a cybercriminal could use brute force to obtain username and password login credentials by a negligent administrator or staff member who has high-level access privileges and has not used appropriate usernames or passwords.


ASMIS System
Network
Search patient and staff records
Edit patient and staff data
Delete patient and staff data
Use brute force to obtain admin login
Cyber criminal



Solution 1:

Allowing staff members to design their password will enable empowerment; however, criteria must be adhered to have secure passwords. It is more difficult for a cybercriminal to use brute force and obtain sensitive credential information if the password has alphanumeric characters, parameters of length and case sensitive.

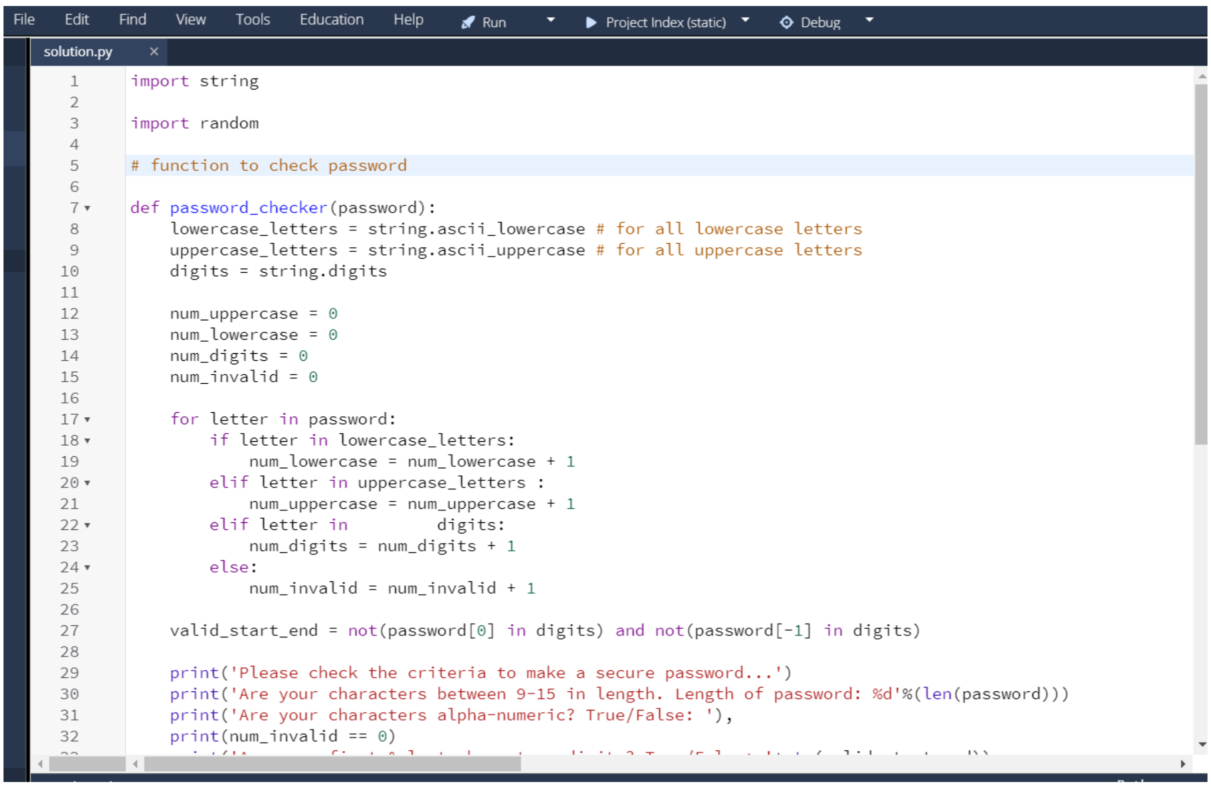
Implementation of the security measure 1:

```
import string

import random

# function to check password is suitable

def password_checker(password):
    lowercase_letters = string.ascii_lowercase # for all lowercase letters

    uppercase_letters = string.ascii_uppercase # for all uppercase letters

    digits = string.digits


    num_uppercase = 0

    num_lowercase = 0

    num_digits = 0

    num_invalid = 0


    for letter in password: #This checks that the criteria for the password can be met

        if letter in lowercase_letters:

            num_lowercase = num_lowercase + 1

        elif letter in uppercase_letters :

            num_uppercase = num_uppercase + 1

        elif letter in         digits:

            num_digits = num_digits + 1

        else:

            num_invalid = num_invalid + 1


    valid_start_end = not(password[0] in digits) and not(password[-1] in digits) #This
checks that the first and last characters are not digits


    print('Please check the criteria to make a secure password...') #This visually aids
the user in designing a better password.
```

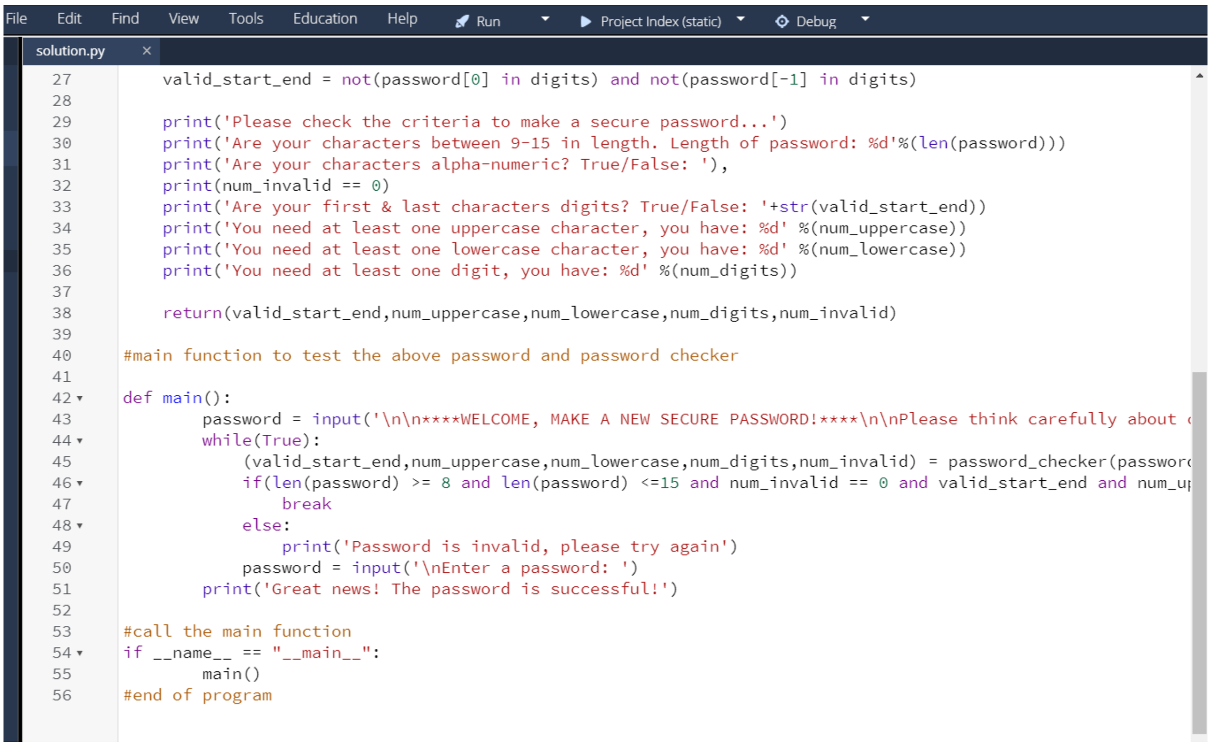
```
    print('Are your characters between 9-15 in length. Length of password: 
%d'%(len(password)))
    print('Are your characters alpha-numeric? True/False: '),
    print(num_invalid == 0)
    print(Do you hsve letter characters at the beginning and at the end? True/False: 
'+str(valid_start_end))
    print('You need at least one uppercase character, you have: %d' 
%(num_uppercase))
    print('You need at least one lowercase character, you have: %d' 
%(num_lowercase))
    print('You need at least one digit, you have: %d' %(num_digits))

    return(valid_start_end,num_uppercase,num_lowercase,num_digits,num_invalid)

#main function to test the above password and password checker

def main():
        password = input('\n\n****WELCOME, MAKE A NEW SECURE 
PASSWORD!****\n\nPlease think carefully about choosing a secure password.\nIt 
should be between 8 and 15 characters, include digits and it should be case 
sensitive.\nLetters should be at the beginning and end.\n\nEnter a password: ')
        while(True):
            (valid_start_end,num_uppercase,num_lowercase,num_digits,num_invalid) = 
password_checker(password)
```

```
            if(len(password) >= 8 and len(password) <=15 and num_invalid == 0 and
valid_start_end and num_uppercase > 0 and num_lowercase > 0 and num_digits >
0):
                break
            else:
                print('Password is invalid, please try again')
            password = input('\nEnter a password: ')
        print('Great news! The password is successful!')


#call the main function
if __name__ == "__main__":
        main()
#end of program
```

Testing the code 1:

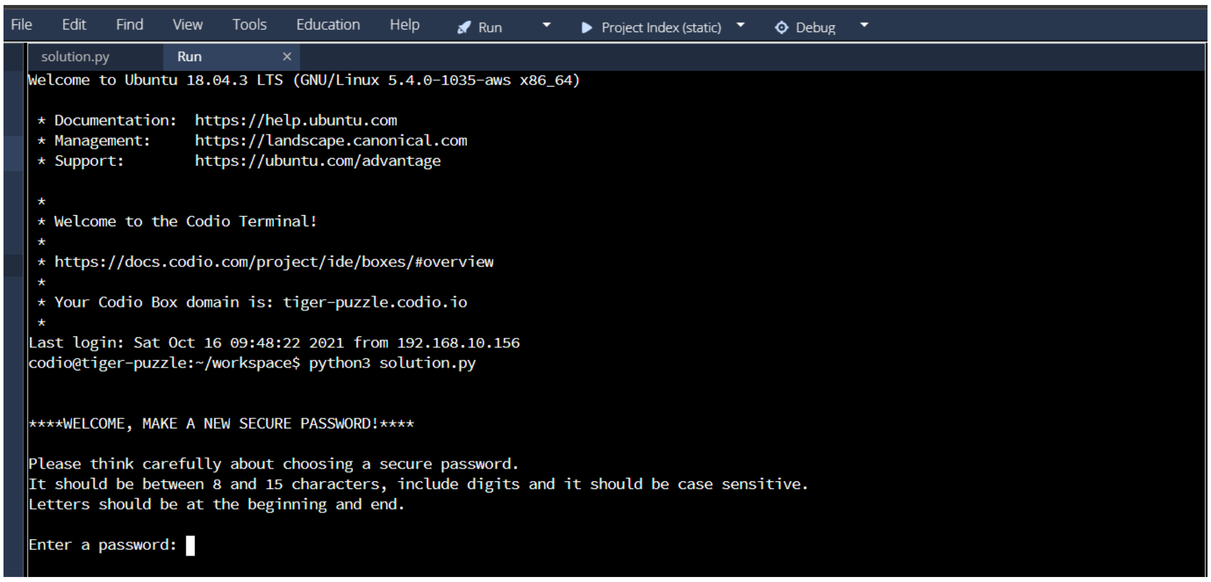


```python
import string

import random

# function to check password

def password_checker(password):
    lowercase_letters = string.ascii_lowercase # for all lowercase letters
    uppercase_letters = string.ascii_uppercase # for all uppercase letters
    digits = string.digits

    num_uppercase = 0
    num_lowercase = 0
    num_digits = 0
    num_invalid = 0

    for letter in password:
        if letter in lowercase_letters:
            num_lowercase = num_lowercase + 1
        elif letter in uppercase_letters :
            num_uppercase = num_uppercase + 1
        elif letter in         digits:
            num_digits = num_digits + 1
        else:
            num_invalid = num_invalid + 1

    valid_start_end = not(password[0] in digits) and not(password[-1] in digits)

    print('Please check the criteria to make a secure password...')
    print('Are your characters between 9-15 in length. Length of password: %d'%(len(password)))
    print('Are your characters alpha-numeric? True/False: '),
    print(num_invalid == 0)
```



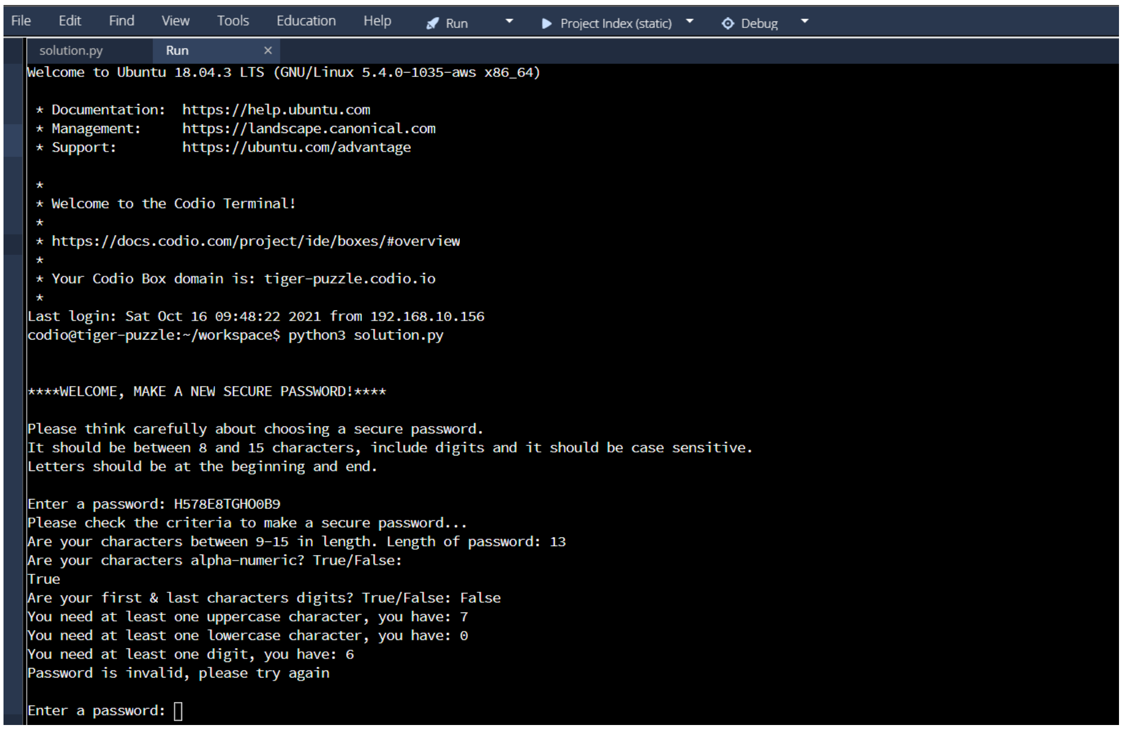
```python
    valid_start_end = not(password[0] in digits) and not(password[-1] in digits)

    print('Please check the criteria to make a secure password...')
    print('Are your characters between 9-15 in length. Length of password: %d'%(len(password)))
    print('Are your characters alpha-numeric? True/False: '),
    print(num_invalid == 0)
```

```python
    print('Are your first & last characters digits? True/False: '+str(valid_start_end))
    print('You need at least one uppercase character, you have: %d' %(num_uppercase))
    print('You need at least one lowercase character, you have: %d' %(num_lowercase))
    print('You need at least one digit, you have: %d' %(num_digits))

    return(valid_start_end,num_uppercase,num_lowercase,num_digits,num_invalid)

#main function to test the above password and password checker

def main():
        password = input('\n\n****WELCOME, MAKE A NEW SECURE PASSWORD!****\n\nPlease think carefully about 
        while(True):
            (valid_start_end,num_uppercase,num_lowercase,num_digits,num_invalid) = password_checker(passwor
            if(len(password) >= 8 and len(password) <=15 and num_invalid == 0 and valid_start_end and num_u
                break
            else:
                print('Password is invalid, please try again')
            password = input('\nEnter a password: ')
        print('Great news! The password is successful!')

#call the main function
if __name__ == "__main__":
        main()
#end of program
```



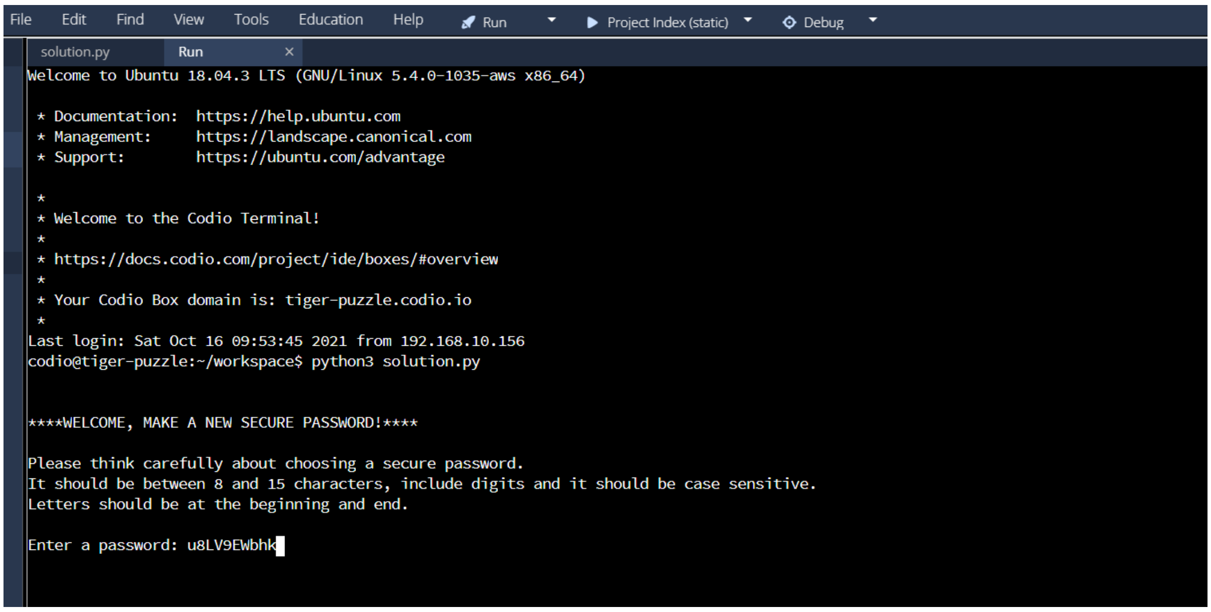
```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Sat Oct 16 09:48:22 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solution.py


****WELCOME, MAKE A NEW SECURE PASSWORD!****

Please think carefully about choosing a secure password.
It should be between 8 and 15 characters, include digits and it should be case sensitive.
Letters should be at the beginning and end.

Enter a password:
```

Following the on-screen instructions for the user to design their own password. By doing this, it would be hoped the password would be more memorable. If the user follows the password validation criteria, they are more educated and gain skills for designing better and more secure passwords in the future.

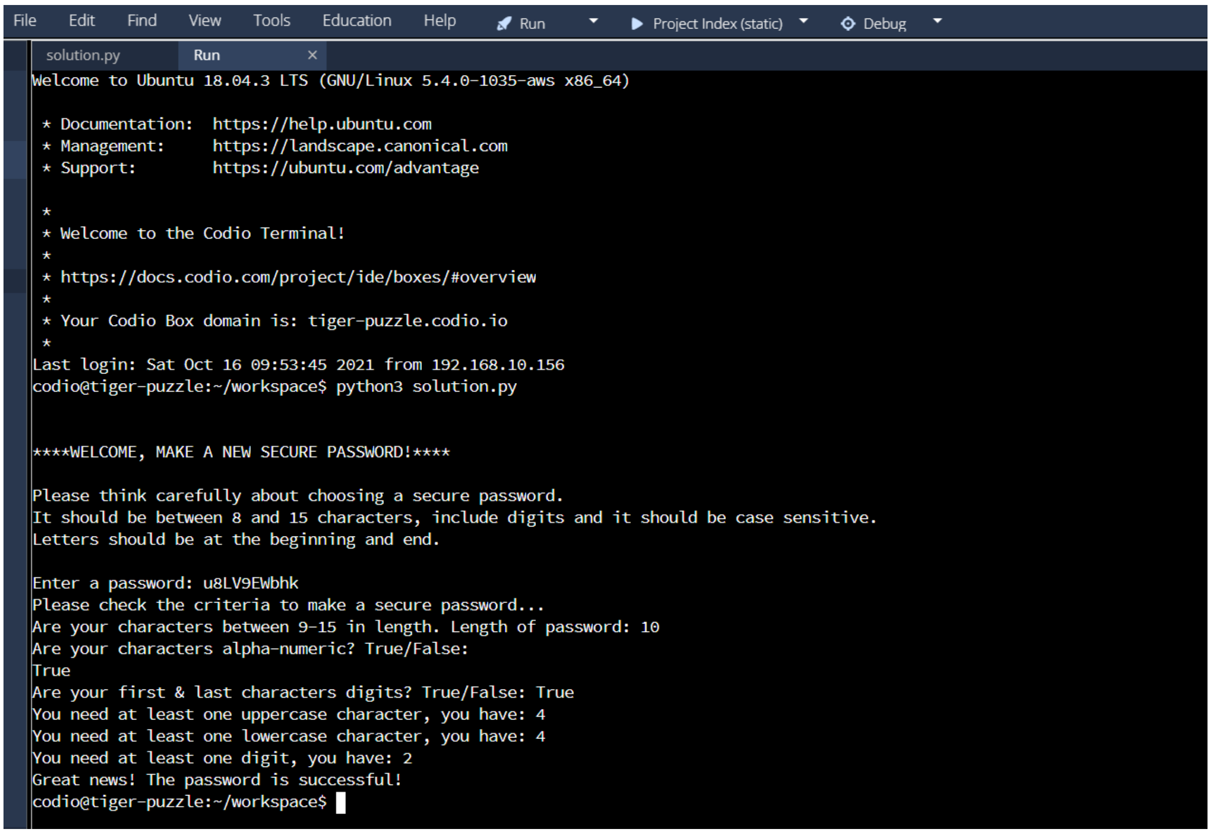


```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Sat Oct 16 09:48:22 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solution.py


****WELCOME, MAKE A NEW SECURE PASSWORD!****

Please think carefully about choosing a secure password.
It should be between 8 and 15 characters, include digits and it should be case sensitive.
Letters should be at the beginning and end.

Enter a password: H578E8TGHO0B9
Please check the criteria to make a secure password...
Are your characters between 9-15 in length. Length of password: 13
Are your characters alpha-numeric? True/False:
True
Are your first & last characters digits? True/False: False
You need at least one uppercase character, you have: 7
You need at least one lowercase character, you have: 0
You need at least one digit, you have: 6
Password is invalid, please try again

Enter a password:
```

In this situation, we see that the user has not had their password validated by not incorporating suggested criteria for validation. The while loop in python code flows to repeat the process and ask the user to enter a password again.



```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Sat Oct 16 09:53:45 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solution.py


****WELCOME, MAKE A NEW SECURE PASSWORD!****

Please think carefully about choosing a secure password.
It should be between 8 and 15 characters, include digits and it should be case sensitive.
Letters should be at the beginning and end.

Enter a password: u8LV9EWbhk
```



```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Sat Oct 16 09:53:45 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solution.py


****WELCOME, MAKE A NEW SECURE PASSWORD!****

Please think carefully about choosing a secure password.
It should be between 8 and 15 characters, include digits and it should be case sensitive.
Letters should be at the beginning and end.

Enter a password: u8LV9EWbhk
Please check the criteria to make a secure password...
Are your characters between 9-15 in length. Length of password: 10
Are your characters alpha-numeric? True/False:
True
Are your first & last characters digits? True/False: True
You need at least one uppercase character, you have: 4
You need at least one lowercase character, you have: 4
You need at least one digit, you have: 2
Great news! The password is successful!
codio@tiger-puzzle:~/workspace$
```

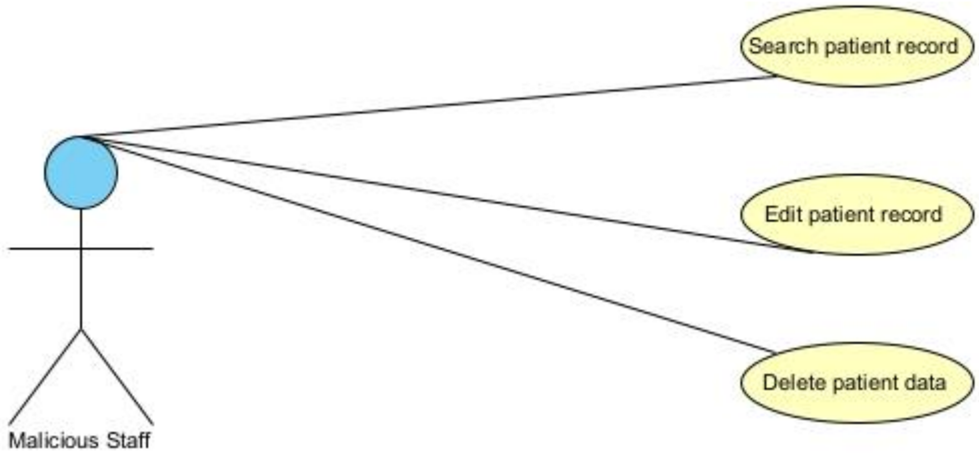
In this case, the user has followed the guidance and designed a validated password.

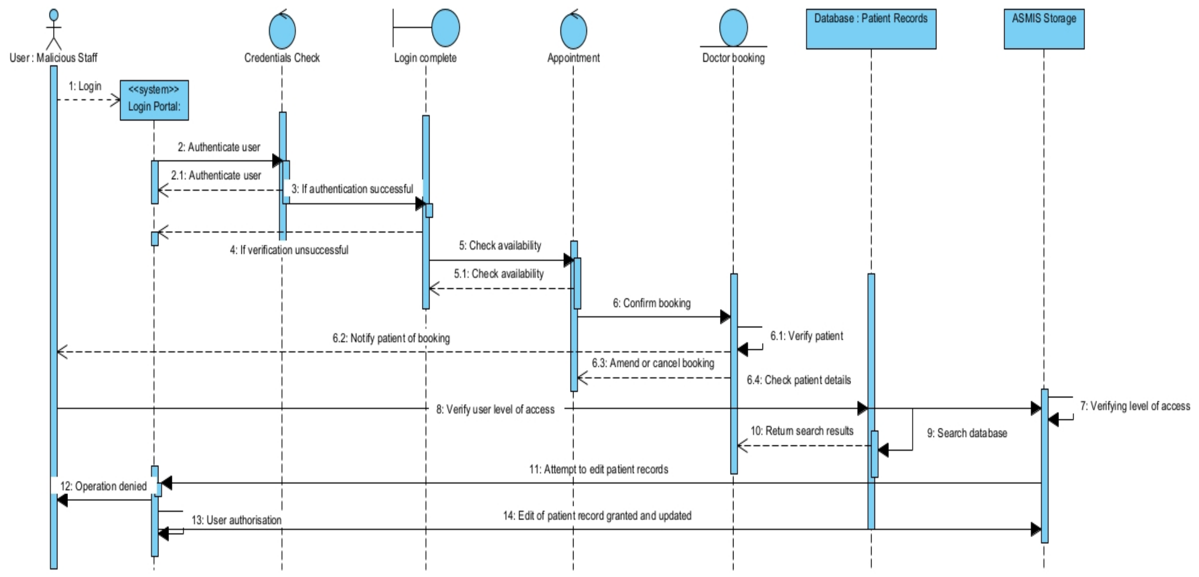
Evaluation 1:

This essential solution allows users to protect themselves and the ASMIS as a preventative measure against cybercrime. Password strength compromising of case-sensitive characters and greater length would be more difficult for a threat actor to use brute force to obtain password credentials (Anderson, 2008). The benefit of this is that the cybercriminal would take far longer to break the password through this security measure as smaller passwords of around eight lowercase characters could take approximately 58,000 hours to be identified (Raza et al., 2012). However, this measure does have limitations, and the inclusion of authentication, encryption and authorisation would offer a reduced threat risk and a more comprehensive method of preventative measures, as we can see in the next section.

Context and use case 2:

The threat is a staff member who may act maliciously to tamper, disclose information or delete data from the storage system.




User : Malicious Staff
Credentials Check
Login complete
Appointment
Doctor booking
Database : Patient Records
ASMIS Storage
<<system>> Login Portal:
1: Login
2: Authenticate user
2.1: Authenticate user
3: If authentication successful
4: If verification unsuccessful
5: Check availability
5.1: Check availability
6: Confirm booking
6.1: Verify patient
6.2: Notify patient of booking
6.3: Amend or cancel booking
6.4: Check patient details
7: Verifying level of access
8: Verify user level of access
9: Search database
10: Return search results
11: Attempt to edit patient records
12: Operation denied
13: User authorisation
14: Edit of patient record granted and updated

Solution 2:

Authenticate usernames and passwords with privilege permissions set with access controls at the login portal stage.

The user will then be allowed authorisation to the correct level of access, and malicious attempts to gain elevated privileges would be prevented.

Implementation of the security measure 2:

```
import hashlib
#The program allows the checking of the user with regard to authentication and
authorised permissions. Aspects of this code has been adapted from (Phillips, 2015)
class User: #This class will manage the user. A username and encypted password
can be stored and the user can login to see if the password is valid.
    def __init__(self, username, password):
        '''Create a new user object. The password
        will be encrypted before storing.'''
        self.username = username
        self.password = self._encrypt_pw(password)
        self.is_logged_in = False

    def _encrypt_pw(self, password):
        '''Encrypt the password with the username and return
        the sha digest.'''
        hash_string = (self.username + password)
        hash_string = hash_string.encode("utf8")
        return hashlib.sha256(hash_string).hexdigest() # SHA-256 algorithm is very
popular for authentication and encryption. Particularly for password verification as
the hash value is compared to a table and is a secure method of storing passwords
than plain text.

    def check_password(self, password): #This checks the password to see if it is the
correct one
```

```
        '''Return True if the password is valid for this
        user, false otherwise.'''
        encrypted = self._encrypt_pw(password)
        return encrypted == self.password

class AuthException(Exception): #The inclusion of exceptions is very important so
users to do not get assigned to usernames already created or the
username/password are not meeting security policy criteria.
    def __init__(self, username, user=None):
        super().__init__(username)
        self.username = username
        self.user = user

class UsernameAlreadyExists(AuthException):
    pass

class PasswordTooShort(AuthException):
    pass

class InvalidUsername(AuthException):
    pass

class InvalidPassword(AuthException):
    pass
```

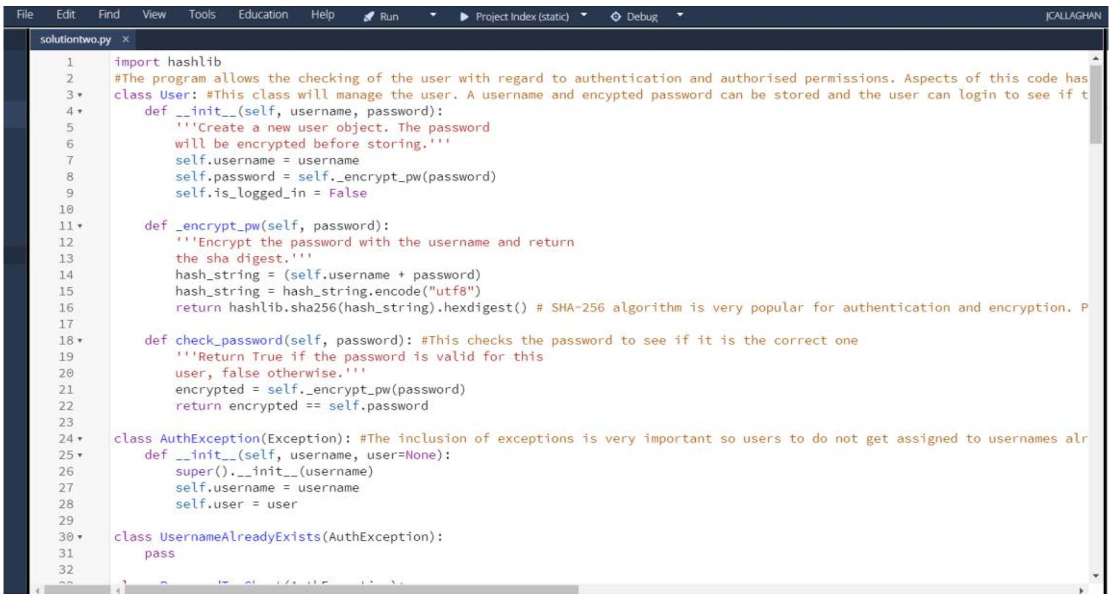
```
class PermissionError(Exception):
    pass


class NotLoggedInError(AuthException):
    pass


class NotPermittedError(AuthException):
    pass


class Authenticator: #This manages the user in terms of logging in and out. It will
authenticate the user.
    def __init__(self):
        '''Construct an authenticator to manage
        users logging in and out.'''
        self.users = (Mendez & Open, 2014)


    def add_user(self, username, password): #Here we see the user management
where checks are made to the credentials.
        if username in self.users:
            raise UsernameAlreadyExists(username) # This checks for two conditions,
one is existing users and the other is password length.
        if len(password) < 6:
            raise PasswordTooShort(username)
        self.users[username] = User(username, password)
    def login(self, username, password):
        try:
            user = self.users[username]
        except KeyError:
            raise InvalidUsername(username)

        if not user.check_password(password):
            raise InvalidPassword(username, user)

        user.is_logged_in = True
        return True

    def is_logged_in(self, username): #Here the process checks the validations of
user credentials and whether to proceed or not through boolean values.
        if username in self.users:
            return self.users[username].is_logged_in
        return False

class Authorizor: #This will map and check the permissions of the user to which
activities the user can perform.
    def __init__(self, authenticator):
        self.authenticator = authenticator
        self.permissions = {}

    def add_permission(self, perm_name): #adding a permission for the class
```

```
        '''Create a new permission that users
        can be added to'''
        try:
            perm_set = self.permissions[perm_name] #Use of set means we add
numerous permissions.
        except KeyError:
            self.permissions[perm_name] = set()
        else:
            raise PermissionError("Permission Exists")

    def permit_user(self, perm_name, username): #granting a permission for the class
        '''Grant the given permission to the user'''
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            raise PermissionError("Permission does not exist")
        else:
            if username not in self.authenticator.users:
                raise InvalidUsername(username)
            perm_set.add(username)

    def check_permission(self, perm_name, username): #checking a permission. The
condition is that the user must be logged in and authenticated and have the set
privilege to be able execute any activity.
        if not self.authenticator.is_logged_in(username):
            raise NotLoggedInError(username)
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            raise PermissionError("Permission does not exist")
        else:
            if username not in perm_set:
                raise NotPermittedError(username)
            else:
                return True



authenticator = Authenticator()
authorizor = Authorizor(authenticator)

import auth #This is imported from the auth.py file attached

# Set up a test user and permission
auth.authenticator.add_user("mistertester", "testerspassword") #mistertester is the
username and testerspassword is the password
auth.authorizor.add_permission("test program") #Permission is given to the user to
attempt to test the program
auth.authorizor.add_permission("change program") #Here the user can attempt to
change the program but does not have the permission to actually change anything
```

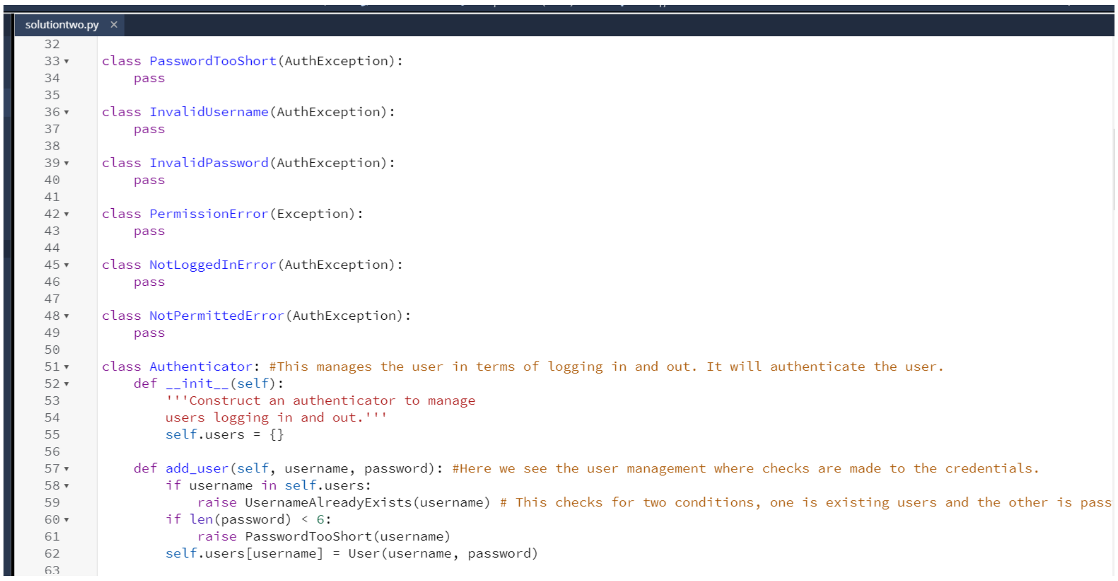
```
auth.authorizor.permit_user("test program", "mistertester") #This permits the user to
carry out the test of the program.
#In this case we can see the user can only login and test the program but permission
are set to not change the program.
class Editor:
    def __init__(self):
        self.username = None #These are the options available to the user
        self.menu_map = {
                "login": self.login,
                "test": self.test,
                "change": self.change,
                "quit": self.quit
                }

    def login(self):
        logged_in = False
        while not logged_in: #The user here can input a username and password
            username = input("username: ")
            password = input("password: ")
            try:
                logged_in = auth.authenticator.login(
                        username, password) #This authenticates the user and checks
credentials
            except auth.InvalidUsername:
                print("Sorry, that username does not exist")
            except auth.InvalidPassword:
                print("Sorry, incorrect password")
            else:
                self.username = username

    def is_permitted(self, permission):
        try:
            auth.authorizor.check_permission(
                permission, self.username) #Once authenticated, the user is checked for
authorisation permissions.
        except auth.NotLoggedInError as e:
            print("{} is not logged in".format(e.username))
            return False
        except auth.NotPermittedError as e:
            print("{} cannot {}".format(
                e.username, permission))
            return False
        else:
            return True

    def test(self):
        if self.is_permitted("test program"):
            print("Testing program now...") #For our test example 'mistertester' we should
see that the user can test the program.
```

```
    def change(self):
        if self.is_permitted("change program"):
            print("Changing program now...") #Our user should not be able to perform
this function as we have not set the appropriate permission to change the program.

    def quit(self):
        raise SystemExit()

    def menu(self):
        try:
            answer = ""
            while True:
                print("""
Please enter a command: #The command options the user will see
\tlogin\tLogin
\ttest\tTest the program
\tchange\tChange the program
\tquit\tQuit
""")
                answer = input("enter a command: ").lower()
                try:
                    func = self.menu_map[answer]
                except KeyError:
                    print("{} is not a valid option".format(
                        answer))
            else:
                func()
        finally:
            print("Thank you for testing the auth module") #The auth module is very
useful as it supports other modules who need to authenticate or authorise.



Editor().menu()
```

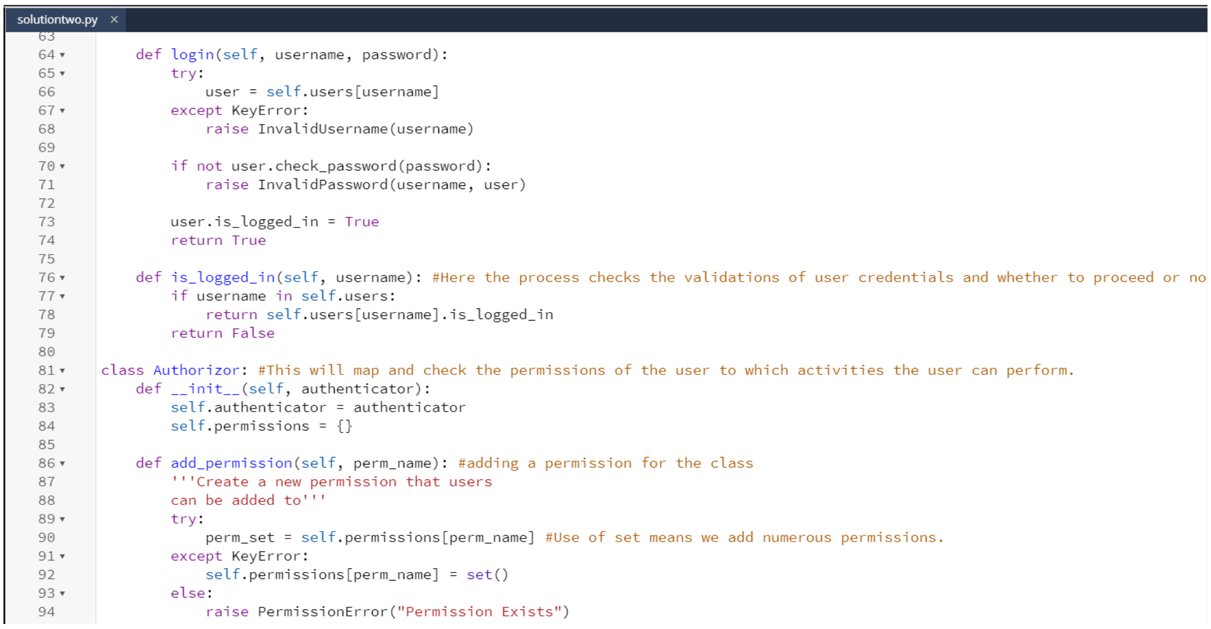
Testing the code 2:



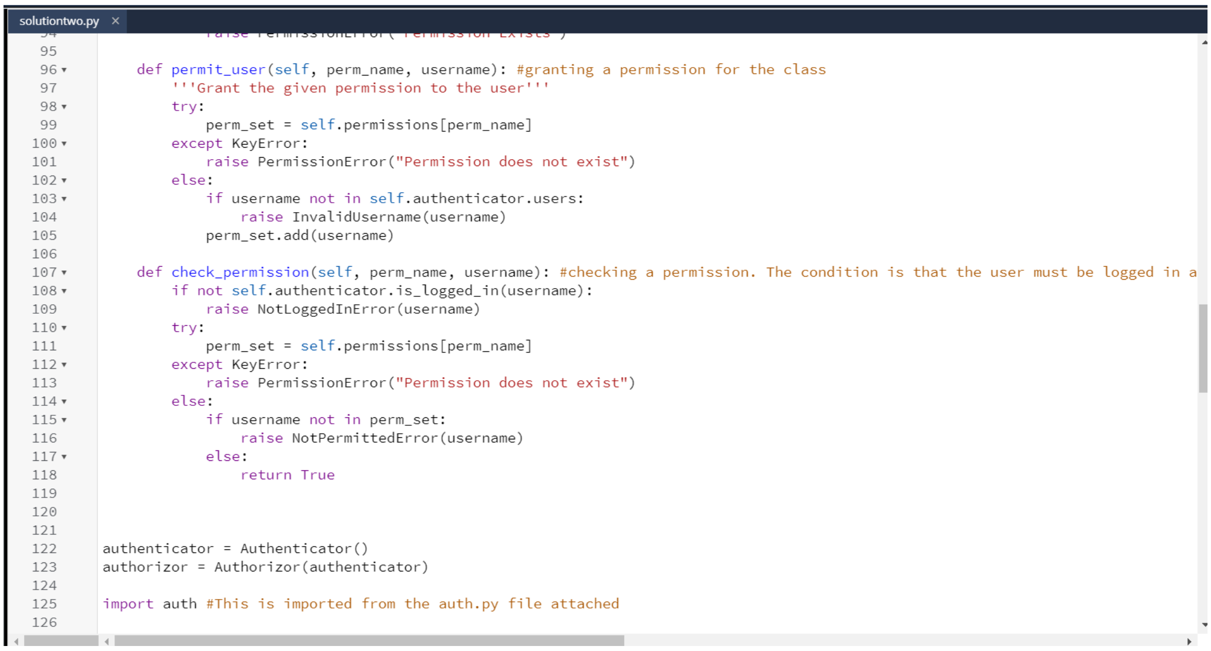
```python
import hashlib
#The program allows the checking of the user with regard to authentication and authorised permissions. Aspects of this code has
class User: #This class will manage the user. A username and encypted password can be stored and the user can login to see if t
    def __init__(self, username, password):
        '''Create a new user object. The password
        will be encrypted before storing.'''
        self.username = username
        self.password = self._encrypt_pw(password)
        self.is_logged_in = False

    def _encrypt_pw(self, password):
        '''Encrypt the password with the username and return
        the sha digest.'''
        hash_string = (self.username + password)
        hash_string = hash_string.encode("utf8")
        return hashlib.sha256(hash_string).hexdigest() # SHA-256 algorithm is very popular for authentication and encryption. P

    def check_password(self, password): #This checks the password to see if it is the correct one
        '''Return True if the password is valid for this
        user, false otherwise.'''
        encrypted = self._encrypt_pw(password)
        return encrypted == self.password

class AuthException(Exception): #The inclusion of exceptions is very important so users to do not get assigned to usernames alr
    def __init__(self, username, user=None):
        super().__init__(username)
        self.username = username
        self.user = user

class UsernameAlreadyExists(AuthException):
    pass

```



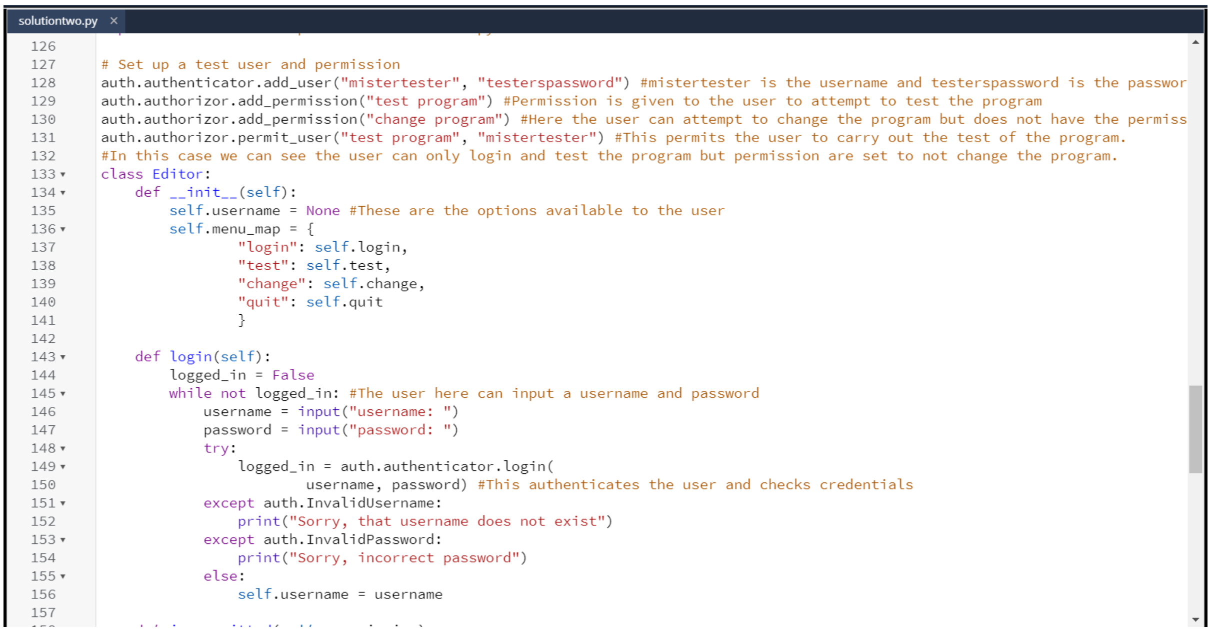
```python

class PasswordTooShort(AuthException):
    pass

class InvalidUsername(AuthException):
    pass

class InvalidPassword(AuthException):
    pass

class PermissionError(Exception):
    pass

class NotLoggedInError(AuthException):
    pass

class NotPermittedError(AuthException):
    pass

class Authenticator: #This manages the user in terms of logging in and out. It will authenticate the user.
    def __init__(self):
        '''Construct an authenticator to manage
        users logging in and out.'''
        self.users = {}

    def add_user(self, username, password): #Here we see the user management where checks are made to the credentials.
        if username in self.users:
            raise UsernameAlreadyExists(username) # This checks for two conditions, one is existing users and the other is pass
        if len(password) < 6:
            raise PasswordTooShort(username)
        self.users[username] = User(username, password)
```



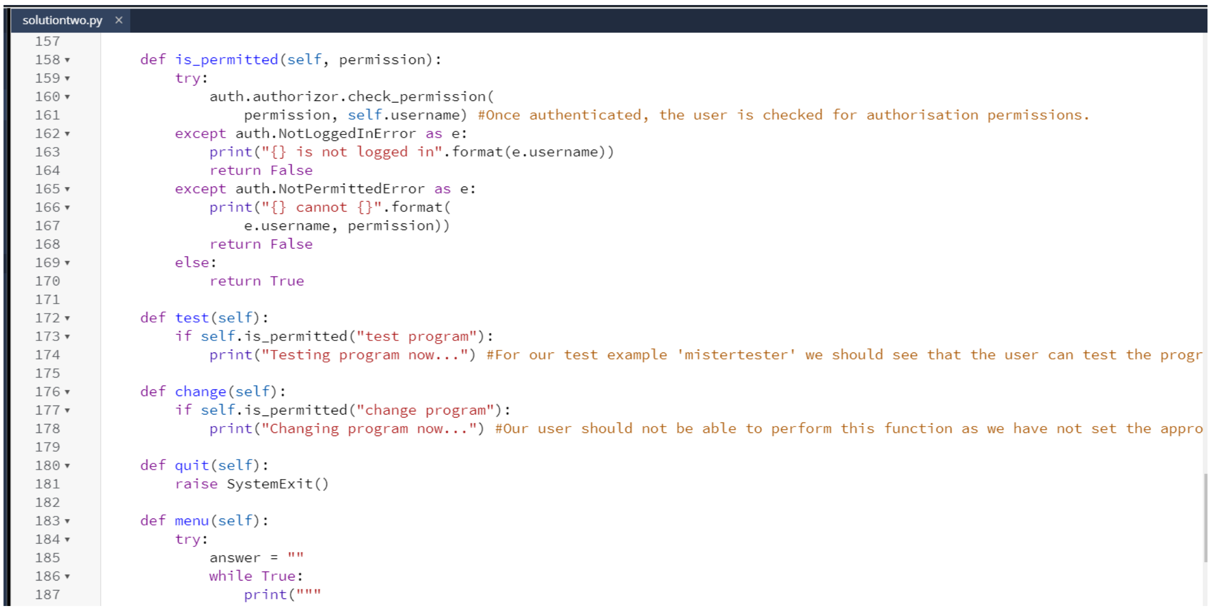
```python
    def login(self, username, password):
        try:
            user = self.users[username]
        except KeyError:
            raise InvalidUsername(username)

        if not user.check_password(password):
            raise InvalidPassword(username, user)

        user.is_logged_in = True
        return True

    def is_logged_in(self, username): #Here the process checks the validations of user credentials and whether to proceed or no
        if username in self.users:
            return self.users[username].is_logged_in
        return False

class Authorizor: #This will map and check the permissions of the user to which activities the user can perform.
    def __init__(self, authenticator):
        self.authenticator = authenticator
        self.permissions = {}

    def add_permission(self, perm_name): #adding a permission for the class
        '''Create a new permission that users
        can be added to'''
        try:
            perm_set = self.permissions[perm_name] #Use of set means we add numerous permissions.
        except KeyError:
            self.permissions[perm_name] = set()
        else:
            raise PermissionError("Permission Exists")
```



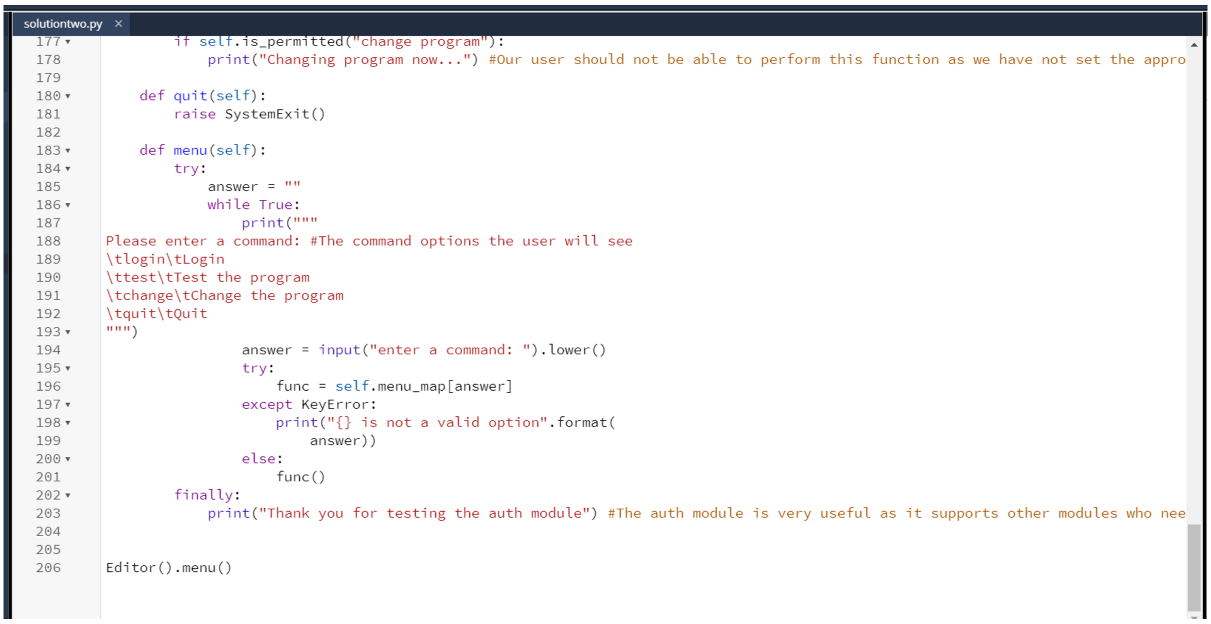
```python

    def permit_user(self, perm_name, username): #granting a permission for the class
        '''Grant the given permission to the user'''
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            raise PermissionError("Permission does not exist")
        else:
            if username not in self.authenticator.users:
                raise InvalidUsername(username)
            perm_set.add(username)

    def check_permission(self, perm_name, username): #checking a permission. The condition is that the user must be logged in a
        if not self.authenticator.is_logged_in(username):
            raise NotLoggedInError(username)
        try:
            perm_set = self.permissions[perm_name]
        except KeyError:
            raise PermissionError("Permission does not exist")
        else:
            if username not in perm_set:
                raise NotPermittedError(username)
            else:
                return True



authenticator = Authenticator()
authorizor = Authorizor(authenticator)

import auth #This is imported from the auth.py file attached

```



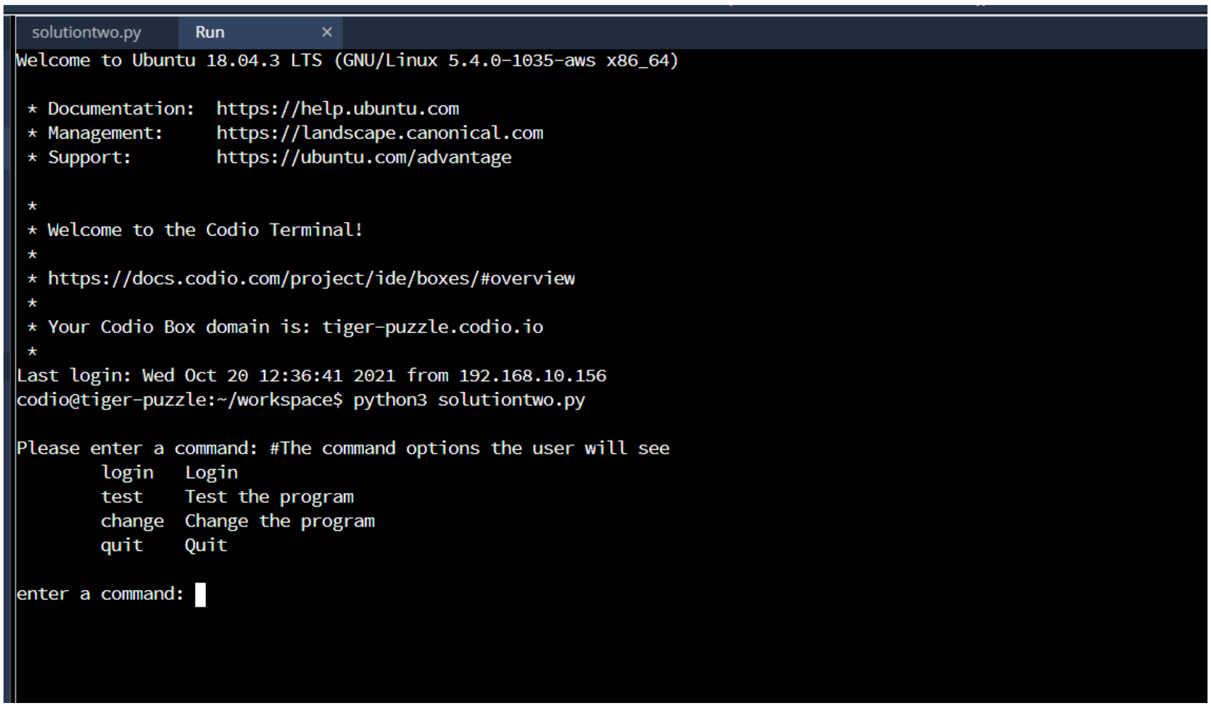
```python

# Set up a test user and permission
auth.authenticator.add_user("mistertester", "testerspassword") #mistertester is the username and testerspassword is the passwor
auth.authorizor.add_permission("test program") #Permission is given to the user to attempt to test the program
auth.authorizor.add_permission("change program") #Here the user can attempt to change the program but does not have the permiss
auth.authorizor.permit_user("test program", "mistertester") #This permits the user to carry out the test of the program.
#In this case we can see the user can only login and test the program but permission are set to not change the program.
class Editor:
    def __init__(self):
        self.username = None #These are the options available to the user
        self.menu_map = {
                "login": self.login,
                "test": self.test,
                "change": self.change,
                "quit": self.quit
                }

    def login(self):
        logged_in = False
        while not logged_in: #The user here can input a username and password
            username = input("username: ")
            password = input("password: ")
            try:
                logged_in = auth.authenticator.login(
                        username, password) #This authenticates the user and checks credentials
            except auth.InvalidUsername:
                print("Sorry, that username does not exist")
            except auth.InvalidPassword:
                print("Sorry, incorrect password")
            else:
                self.username = username

```



```python

    def is_permitted(self, permission):
        try:
            auth.authorizor.check_permission(
                permission, self.username) #Once authenticated, the user is checked for authorisation permissions.
        except auth.NotLoggedInError as e:
            print("{} is not logged in".format(e.username))
            return False
        except auth.NotPermittedError as e:
            print("{} cannot {}".format(
                e.username, permission))
            return False
        else:
            return True

    def test(self):
        if self.is_permitted("test program"):
            print("Testing program now...") #For our test example 'mistertester' we should see that the user can test the progr

    def change(self):
        if self.is_permitted("change program"):
            print("Changing program now...") #Our user should not be able to perform this function as we have not set the appro

    def quit(self):
        raise SystemExit()

    def menu(self):
        try:
            answer = ""
            while True:
                print("""
```

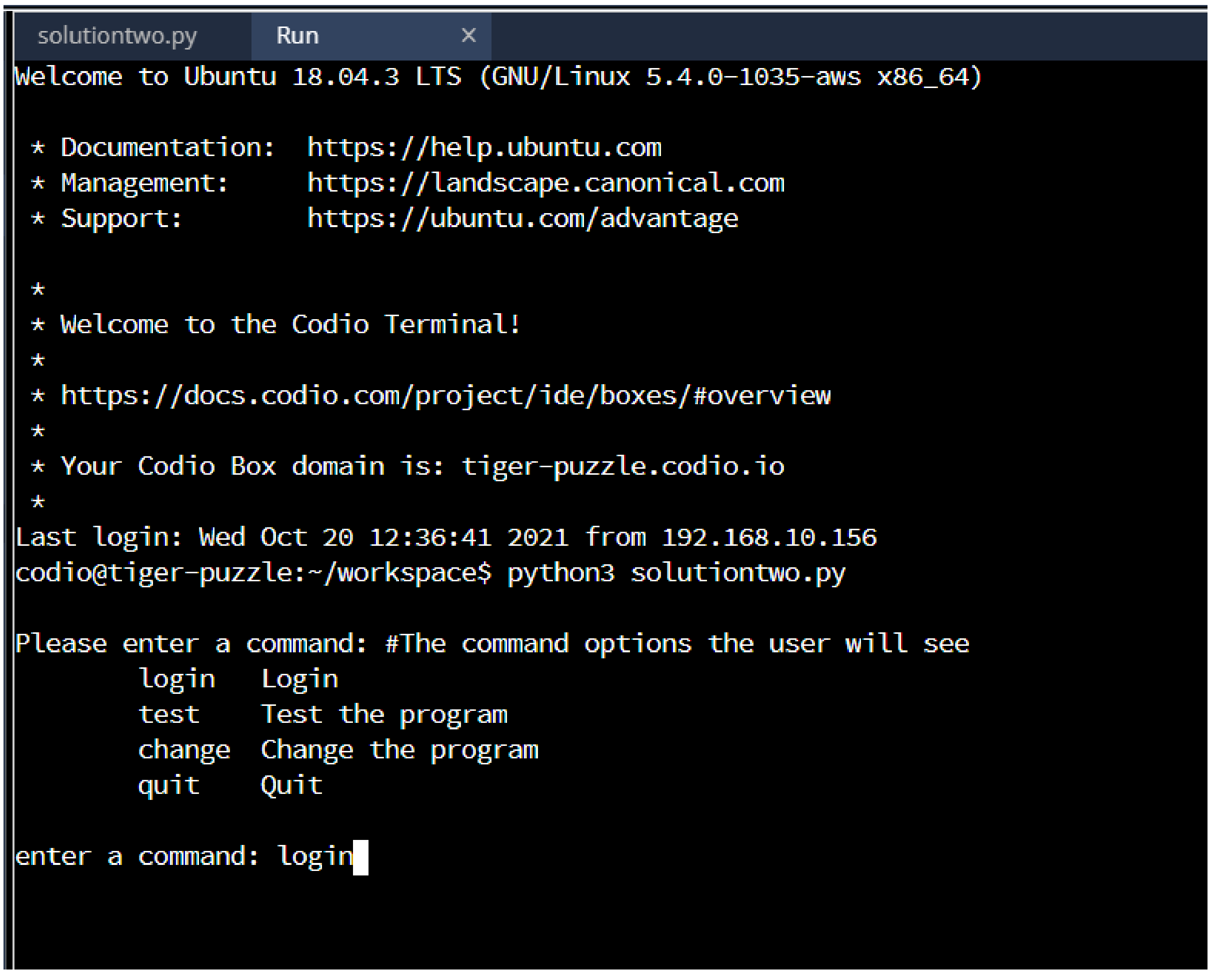

solutiontwo.py

```
        if self.is_permitted("change program"):
            print("Changing program now...") #Our user should not be able to perform this function as we have not set the appro

    def quit(self):
        raise SystemExit()

    def menu(self):
        try:
            answer = ""
            while True:
                print("""
Please enter a command: #The command options the user will see
\tlogin\tLogin
\ttest\tTest the program
\tchange\tChange the program
\tquit\tQuit
""")
                answer = input("enter a command: ").lower()
                try:
                    func = self.menu_map[answer]
                except KeyError:
                    print("{} is not a valid option".format(
                        answer))
                else:
                    func()
        finally:
            print("Thank you for testing the auth module") #The auth module is very useful as it supports other modules who nee


Editor().menu()
```


solutiontwo.py Run

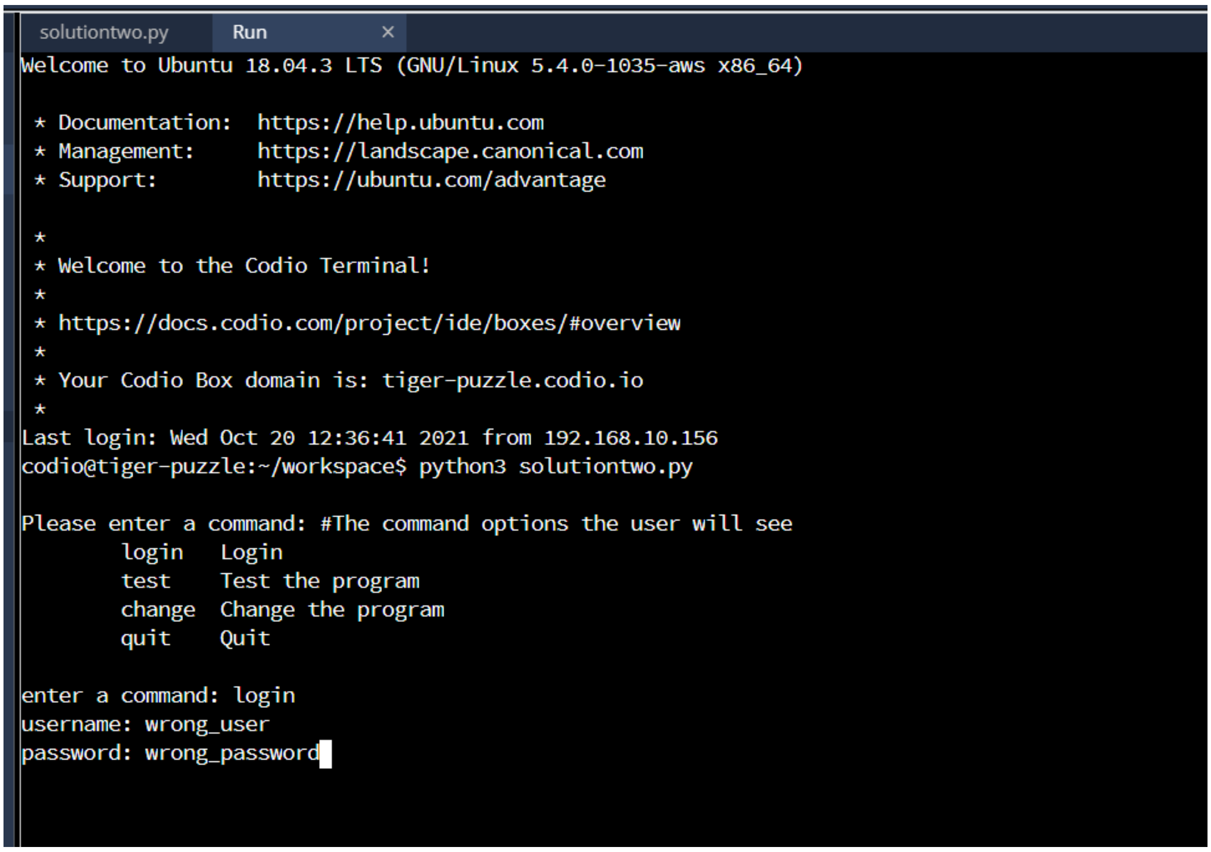
```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Wed Oct 20 12:36:41 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solutiontwo.py

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: 
```

The user should enter the 'login' command. This will then prompt the user to enter credentials.



solutiontwo.py Run

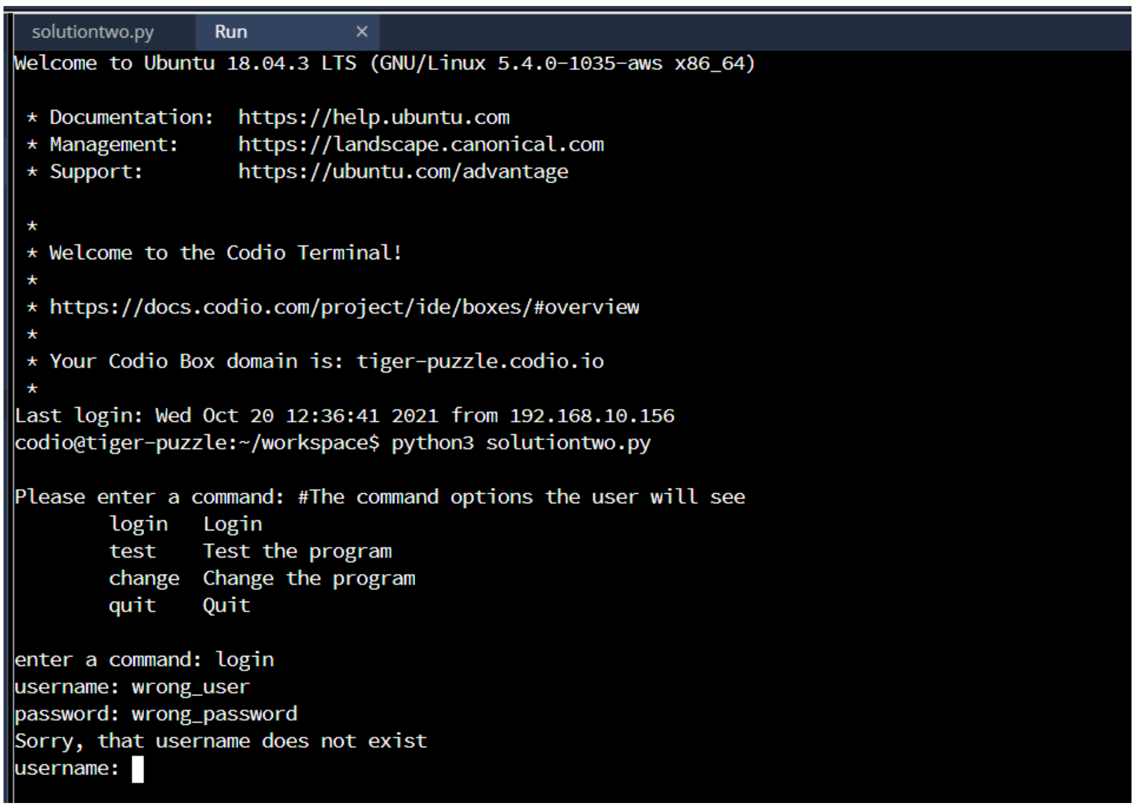
```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Wed Oct 20 12:36:41 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solutiontwo.py

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: login
```



solutiontwo.py Run

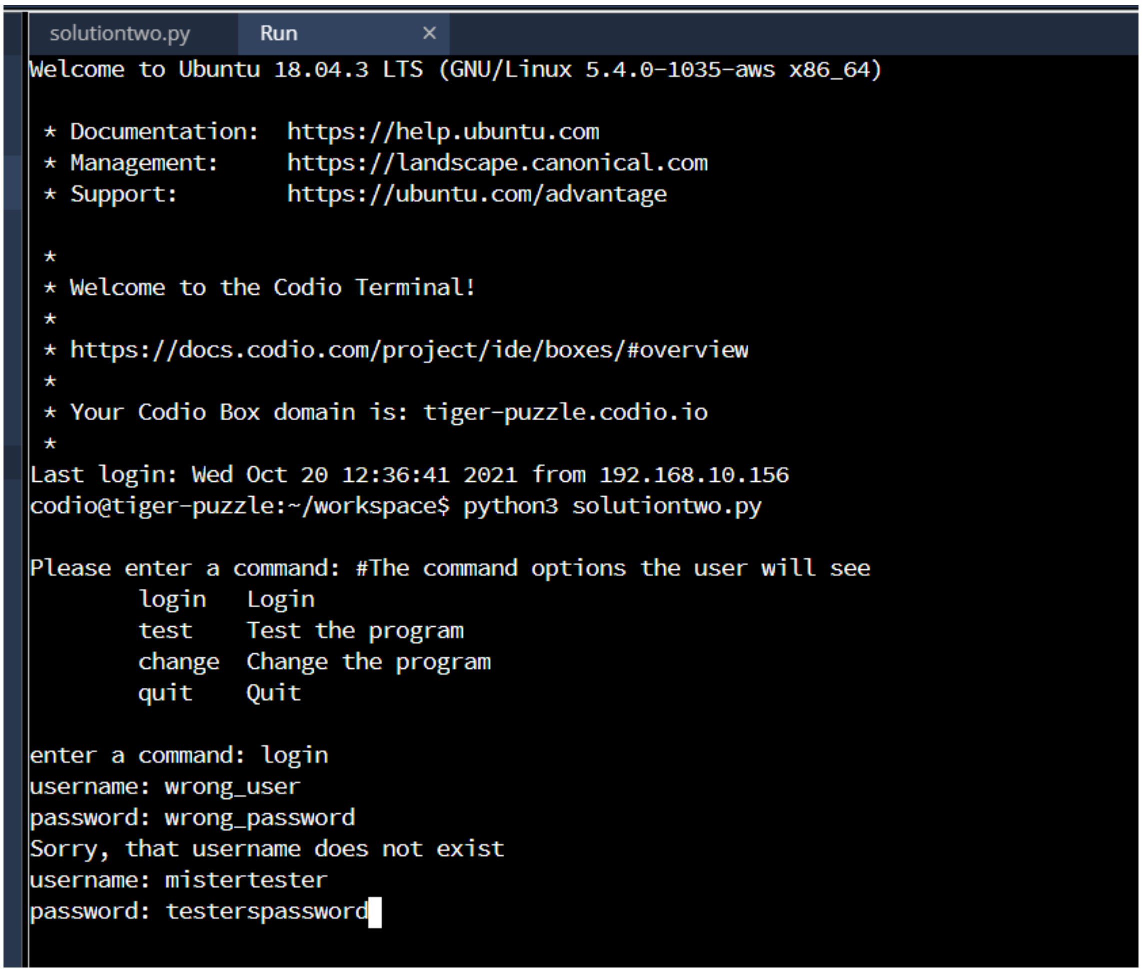
```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Wed Oct 20 12:36:41 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solutiontwo.py

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: login
username: wrong_user
password: wrong_password
```

In this case, we see the user enter an invalid username and password.


solutiontwo.py Run ×

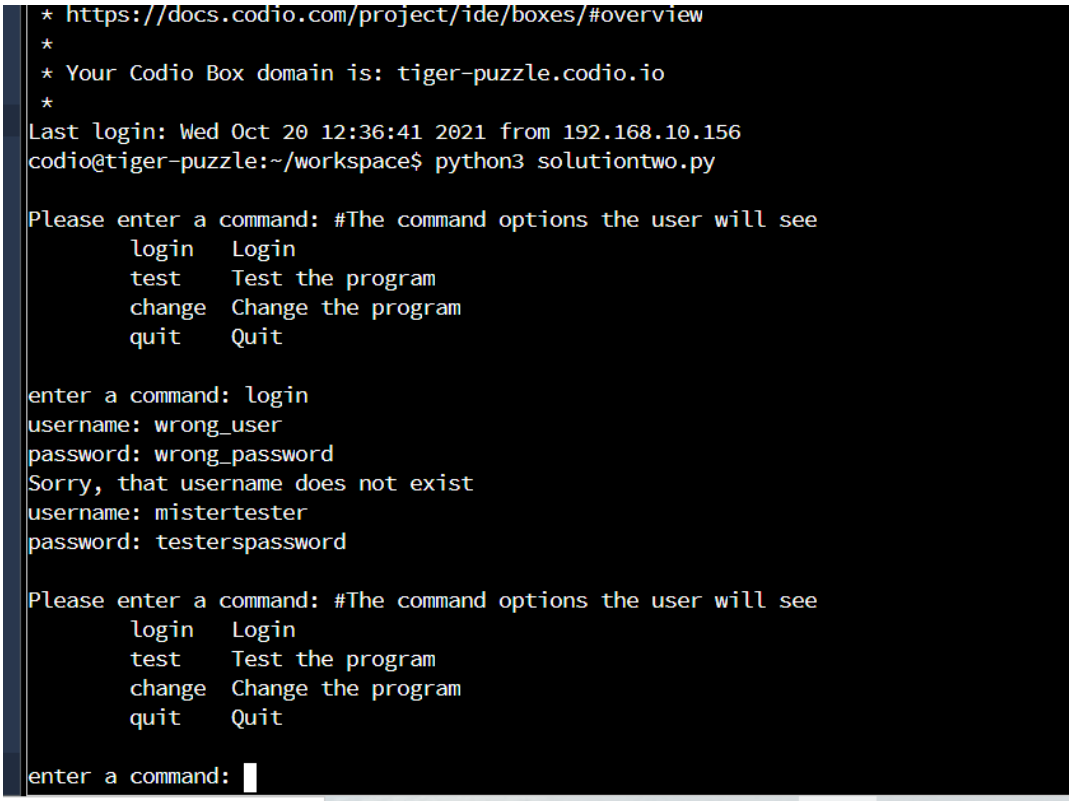
```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Wed Oct 20 12:36:41 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solutiontwo.py

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: login
username: wrong_user
password: wrong_password
Sorry, that username does not exist
username: 
```

The program recognises that the username does not exist and loops the process again. The user has another opportunity to log in for validation.



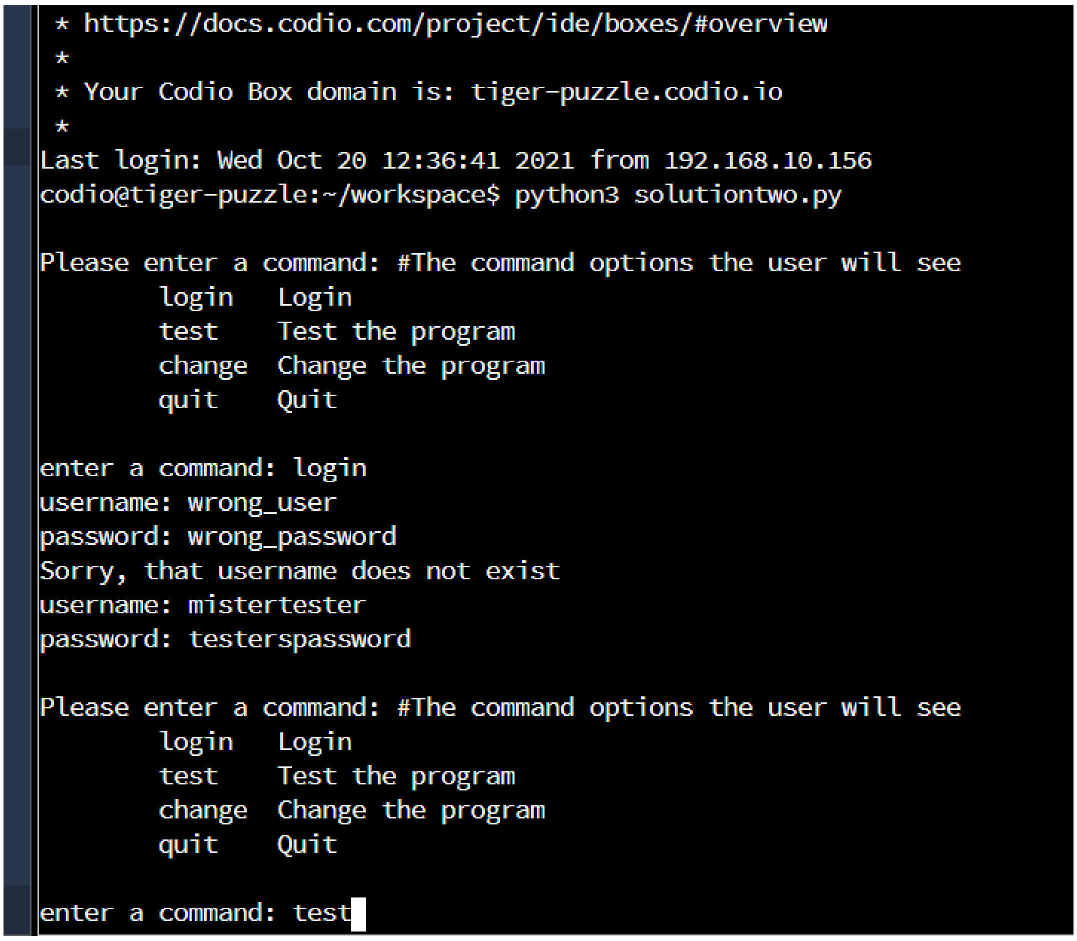
```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Wed Oct 20 12:36:41 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solutiontwo.py

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: login
username: wrong_user
password: wrong_password
Sorry, that username does not exist
username: mistertester
password: testerspassword
```

The user enters a username and password.



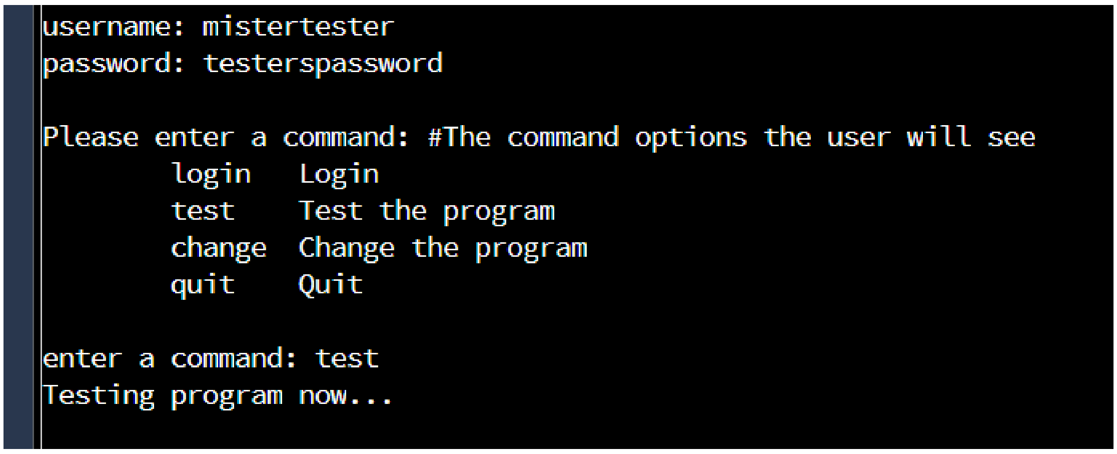
```
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Wed Oct 20 12:36:41 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solutiontwo.py

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: login
username: wrong_user
password: wrong_password
Sorry, that username does not exist
username: mistertester
password: testerspassword

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: 
```

The username and password are successfully validated and allow the user to the next stage, either testing the program, changing the program, or quitting.



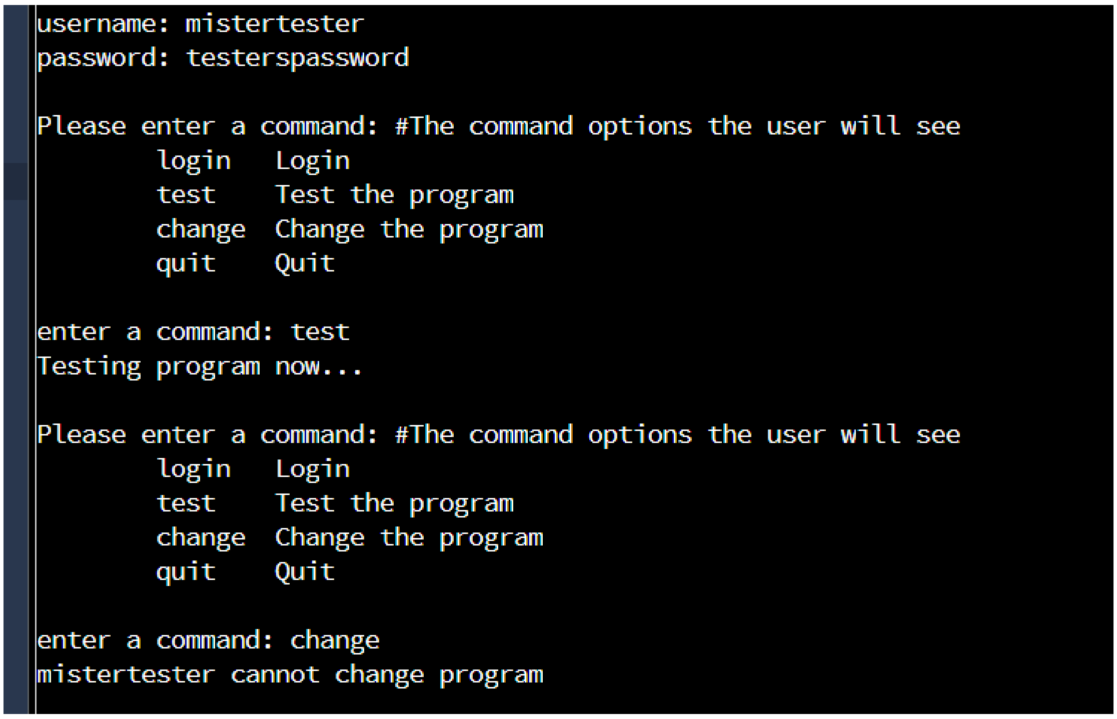
```
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: tiger-puzzle.codio.io
 *
Last login: Wed Oct 20 12:36:41 2021 from 192.168.10.156
codio@tiger-puzzle:~/workspace$ python3 solutiontwo.py

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: login
username: wrong_user
password: wrong_password
Sorry, that username does not exist
username: mistertester
password: testerspassword

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: test
```

Using the command 'test', the user tests the program.



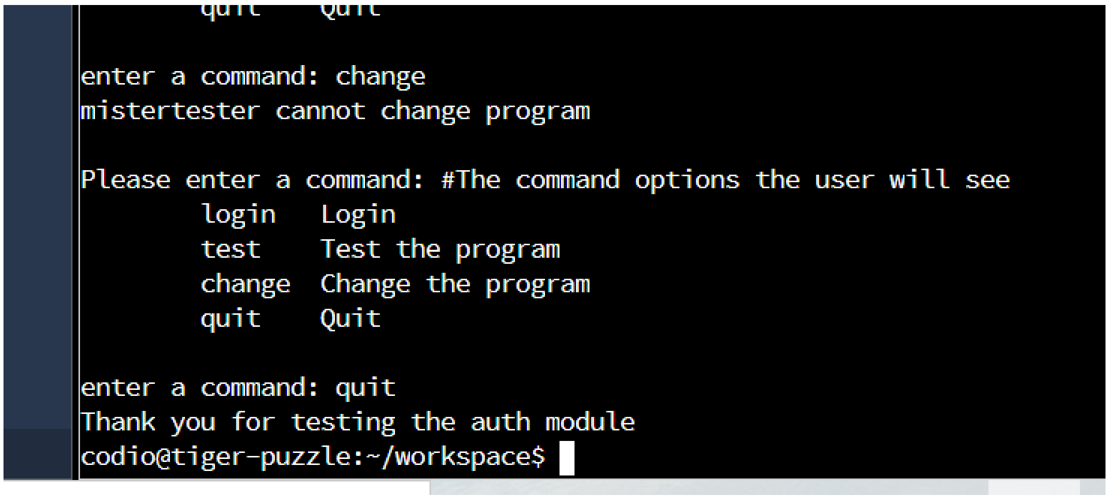
```
username: mistertester
password: testerspassword

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: test
Testing program now...
```

As we see the program being tested, this also explains that the user 'mistertester' has the privilege of doing this activity.



```
username: mistertester
password: testerspassword

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: test
Testing program now...

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: change
mistertester cannot change program
```

Once complete, the user returns to the command line and commands' change' to change the program. This could be alter privileges, or a malicious user could tamper with the program. However, the security policy is strict and has restricted 'mistertester' access and cannot perform any change to the program.



```
        quit    Quit

enter a command: change
mistertester cannot change program

Please enter a command: #The command options the user will see
        login   Login
        test    Test the program
        change  Change the program
        quit    Quit

enter a command: quit
Thank you for testing the auth module
codio@tiger-puzzle:~/workspace$ 
```

On return to the command line, the user commands 'quit', which exits the program.

Evaluation 2:

The security measure provides a practical method to allow users to store a username and encrypted password. Once the user has logged in, the password can be validated. After the authentication process, the permissions can be set to allow authorisation, and the user can have certain levels of access privilege. This program is very beneficial as malicious staff members can have restricted access to certain aspects of the ASMIS. This will mitigate threats such as information disclosure and tampering from the STRIDE threat model (Khan et al., 2017). As long as the security policy states that the correct staff has the right level of privilege, the program can work effectively. The encryption of the password would be it extremely difficult for a threat actor to find out the password, which would further strengthen the policy. The program can check to see if the username has already been assigned before and meets specific criteria to be valid. This process automates the authentication process allows the administrator to focus on security issues elsewhere. Through authorisation, the user can be comforted by knowing that they are working with the correct access level.

However, some limitations could affect this. Any form of negligent behaviour by staff, whether intentional or not intentional, could allow a breach and cause a more significant risk. Disgruntled employees could be an example of intentional, reckless behaviour that could cost the medical centre financially and potentially human life. The other limitation is that the users' credentials all need to be stored first in this program. Whilst this data can be secured through encryption, this means that the credentials need to be inputted by an administrator, which could take time away from other security tasks.

References:

Anderson, R. (2008) *Security Engineering: A Guide to Building Dependable Distributed Systems.* 2nd ed. Indianapolis, USA: Wiley.

Khan, R., Mclaughlin, K., Laverty, D. & Sezer, S. (2017) 'STRIDE-based threat modeling for cyber-physical systems', *IEEE PES Innovative Smart Grid Technologies Conference Europe.* Italy, 26-29 September 2017. ISGT Europe. Available from: https://ieeexplore.ieee.org/document/8260283 [Accessed 19 October 2021]

Phillips, D. (2015). *Python 3 Object Oriented Programming.* 2nd ed. Birmingham, UK: Packt Publishing.[Accessed 20 October 2021].

Raza, M., Iqbal, M., Sharif, M. & Haider, W. (2012). A Survey of Password Attacks and Comparative Analysis on Methods for Secure Authentication. *World Applied Sciences Journal* 19: 439-444. Available from: https://www.researchgate.net/publication/236898951_A_Survey_of_Password_Attacks_and_Comparative_Analysis_on_Methods_for_Secure_Authentication [Accessed 20 October 2021]