assignment1_template

April 1, 2022

1 FIT9136 Assignment 1

2 1. Get user input function

This function has one positional argument "requirement" which is str type. The variable "requirement" can only be the value of ("letter", "number", "letter_or_number_or_underscore", "email"). According to the "requirement" value, this function asks the user to input a corresponding value and return this input.

If "requirement" is "letter", user input can only be letters from [a-zA-Z].

If "requirement" is "number", user input can only be [0-9].

If "requirement" is "letter_or_number_or_underscore", user input can only be [a-zA-z0-9_].

If "requirement" is "email", the user input must contain "@" and ".com".

If user input cannot match the "requirement", a loop should be applied to keep asking user to input until a valid result is obtained. Finally, the valid value should be returned.

For example, when calling this function and giving "requirement" value "letter", user input like "abc123" will receive an error message printed out. Then, your system should print out messages to ask user re-input until an all letter input is made like "abcde".

[ ]:
```
"""NAME: ABDULLAH MANSUR MALEK
STUDENT ID: 32646941
START DATE: 18/03/2022
LAST MODIFIED DATE: 24/03/2022

PROGRAM DESCRIPTION:
This is a program to manage user log-in to a system.
It accepts inputs from users to securely store their log-in credentials for
future log-in's. When a user tries to log-in by entering a username and
password, the program authenticates the user based on the existing users
present in the system.
"""

# define your function here

"""This is a generic function that is used to accept and validate inputs of
various kinds from the user e.g. username, password, email, postcode.
```

```
The constraints on user input e.g. letters, numbers, etc. are specified
by the requirement parameter.
The function validates the user input as per the given specifications
and returns it.
"""
def get_user_input(requirement):
    userInput = input("Enter input: ")
    # Validate input as per requirement variable
    if requirement == "letter":
        # Keep asking for new input until valid input is obtained
        while(userInput.isalpha() == False):
            userInput = input("Invalid input. Enter all letters only: ")
    elif requirement == "number":
        while(userInput.isdigit() == False):
            userInput = input("Invalid input. Enter all numbers only: ")
    elif requirement == "letter_or_number_or_underscore":
        # Replacing only underscores with a letter, no other special symbols␣
↪allowed
        while(userInput.replace("_", "X").isalnum() == False):
            userInput = input("Invalid input. Enter letters/numbers/underscores␣
↪only: ")
    elif requirement == "email":
        """
        Email validity criteria:
        1. Must be alphanumeric and can have "_"
        2. Must also have, but not start with "@" and ".com"
        3. "@" and ".com" must be in that specific serial order, and cannot be␣
↪adjacent
        4. "@" and ".com" must also appear exactly once.
        5. Can't start with an "_" or "."
        """
        while(userInput.replace("_", "X").replace("@", "X").replace(".", "X").
↪isalnum() \
            == False or userInput.find("@") < 1 or userInput.find(".com") < 1 \
            or userInput.find(".com") - userInput.find("@") <= 1 \
            or userInput.count("@") > 1 or userInput.count(".com") > 1 \
            or userInput.find(".") == 0 or userInput.find("_") == 0):
            userInput = input("Invalid input. Enter a valid email id: ")
    return userInput

# get_user_input("number")
# get_user_input("letter_or_number_or_underscore")
# get_user_input("email")
```

2.1 2. Encryption function

This function has a string type positional argument. This function is used to encrypt user input passwords. When we use a web application and enter our password. Our password values will not be stored directly as plain text into the application's database. Because if an attacker get the database information, they can obtain the user password text. Commonly, users' passwords will be encrypted with some algorithms(like MD5) to avoid further loss when database leakage happens. Our function emulates a password encryption process. The final encrypted password will follow the requirements listed below.

One variable all_punctuation is provided whose value is all_punctuation =“ ““!”#$%&’()*+,-./:;<=>?@[]^_‘{|}~“ “ “.

get the character of all_punctuation at input string length module all_punctuation length as the first_character.

The second_character position in all_punctuation is the input string length module 5.

The third_character position in all_punctuation is the input string length module 10.

Start character “^^^” and End character “$$$” for the final encrypted string.

Example:

input string: “password”

first_character: “)”

second_character: “$”

third_character: “)”

Encrypted result: “^^^)p)$$a$$)))s))))s)$$w$$)))o))))r)$$d$$$$$”

The encrypted string will be returned at the end of this function.

[ ]:
```
"""This function is used to encrypt the user password to enhance data security.
It has one parameter which is the password in plain text.
The function encrypts this argument string using the given logic and returns
it.
"""
def encryption(input_str):
    all_punctuation = """!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""
    # your answer
    # Obtaining the three encoding characters
    first_character = all_punctuation[len(input_str) % len(all_punctuation)]
    second_character = all_punctuation[len(input_str) % 5]
    third_character = all_punctuation[len(input_str) % 10]
    # making a list of these three characters for use below
    encodingList = [first_character, second_character, third_character]
    # print(second_character)
    # Building the encrypted password
    i = 1
    output_str = ""
```

```
    for j in input_str:
        """Multiple the code character by its index i and
        append this string before and after the password
        character "j"
        """
        j = encodingList[i-1]*i + j + encodingList[i-1]*i
        output_str += j
        # Update index i in encodingList, reset to 0 if end of list reached
        i = (i+1) % 4
        if i == 0:
            i = 1
    # Add the final head and tail strings to the encrypted password
    output_str = "^^^" + output_str + "$$$"
    # print(output_str)
    return output_str

# encryption("12333")
```

2.2 3. Generate user id function

This function contains two positional arguments that are number_of_digits(int type), number_list(list type, a list of str). Based on the number_of_digits, you are required to generate an all digit string and all the string in the number_list should be unique.

For example, the number_of_digits = 7, the generated string should only contain 7 digits. If the number_list = [“1234567”, “2345678”], the newly generated id cannot be the same as any element in the given list. The generated string id should be returned.

[ ]:
```
# define your function here
import random

"""This function generates a unique random numeric string and returns it.
It has two parameters:
1. number of digits that random string must have.
2. the number list containing the previously generated random strings.
"""
def generate_user_id(number_of_digits, number_list):
    """
    Keep generating a new id until a unique id is
    obtained
    """
    while True:
        user_id = ""
        # Generate a random id string
        for i in range(number_of_digits):
            user_id += str(random.randint(0,9))
            # print("-")
            # user_id = "443322"
```

```
        # Check if generated id is unique
        flag = 0
        for i in number_list:
            if user_id == i: # not unique, stop
                flag = 1
                break
        if flag == 0:
            # print("Unique ==> added")
            break
    return user_id

# generate_user_id(6, ["123456", "223344", "443322"])
```

2.3 4. Check username exist function

This function contains two positional arguments that are username(str type) and user_list(list type, a list of list). The user_list looks like [[username1, password1, email1, postcode1],[username2, password2, email2, postcode2]...]. This function should check whether the username string exists in the user_list or not and return the boolean result.

For example, given user_list=[[“aaaaa”, “bbbbb”, “aaa@gmail.com”, “3000”], [“eeeee”, “fffff”, “eee@gmai.com”, “4000”]], if the given username is “aaaaa”, return True.

[ ]:
```
# define your function here

"""This function checks if the argument username exists in
an argument list of users.
It has two parameters:
1. username, which is to be checked for uniqueness
2. user list, the list of existing users
The function returns a boolean value; True if username is
found in the user list, False otherwise.
"""
def check_username_exists(username, user_list):
    for i in user_list:
        if (i[0] == username): #match found
            return True
    #match not found
    return False

# check_username_exists("aaaaa", [["aaaaa", "bbbbb", "aaa@gmail.com", "3000"],␣
↪["eeeee", "fffff", "eee@gmail.com", "4000"]])
```

2.4 5. Authenticate username and password function

This function contains three positional arguments that are username(str type), password(str type) and user_dict(dict type). The user_dict looks like {user_id1: [username1, password1, email1,

postcode1], user_id2: [username2, password2, email2, postcode2].....}. You are required to check whether the given username and password can match one item in the user_dict.

[ ]:
```
# define your function here

"""This functions authenticates an argument
username-password pair.
It has three parameters:
1. username string
2. password string(plain text)
3. user dictionary
The function takes the username-password pair and scans
the user dictionary for it.
If a match is found, True is returned, and False otherwise
"""
def authenticate(username, password, user_dict):
    #scan through all pairs of dictionary
    for i in user_dict.items():
        if (i[1][0] == username and \
            i[1][1] == encryption(password)): #match found
            return True
    #match not found
    return False

# authenticate("bbbbb", "1222", {"12345": ["aaaa", "^^^%1%!!2!!%%%3%%%3%!!3!!
 ↪$$$"\
#     , "aa@gmail.com", "3151"], "34567": ["bbbbb", "^^^%1%%%2%%%%%2%%%%2%$$$",\
#                                          "bb@gmail.com", "3000"]})
```

2.5 6. Add user to list function

This function has two positional arguments that are user_id_list(list type) and user_list(list type). The user_id_list looks like ['1234', '5123', '62345',.....] and the user_list looks like [[username1, password1, email1, postcode1],[username2, password2, email2, postcode2]...]. In this function, you should call the get user input function several times to ask the user to input username(only contains letters), password(contains letter or number or underscore), email(email format) and postcode(only contains numbers). Username cannot have duplicates in the user_list(call check username exist function here). After getting the postcode, you are required to generate a unique user_id for this user.
The rules are listed below.

1000 <= postcode < 2000 → generate a 7 digits user id

2000 <= postcode < 3000 –> generate a 8 digits user id

3000 <= postcode < 4000 –> generate a 9 digits user id

4000 <= postcode < 5000 –> generate a 10 digits user id

After generating the unique user_id, it should be added into the user_id_list.
Once getting all the necessary information from user input, a new user(format: [username, password, email, postcode]) should be added to the user_list. The password should be encrypted when adding user info into user_list.

For example, after getting user input, a user like ["aaaaa", "~~~%1%%%2%%%%%2%%%%2%$$$", "aa@gmail.com", "3131"] can be added into the user_list and a user id "123456789" can be added into the user_id_list.

[ ]:
```
# define your function here

def add_user_to_list(user_id_list, user_list):
    #Loop until unique username is entered
    while True:
        print("USERNAME", end = "\n")
        username = get_user_input("letter")
        if (check_username_exists(username, user_list) == False):
            break
        else:
            print("Username already exists. Try again!", end = "\n")
    #Accept password
    print("PASSWORD", end = "\n")
    password = get_user_input("letter_or_number_or_underscore")
    password = encryption(password)
    #Accept email
    print("EMAIL", end = "\n")
    email = get_user_input("email")
    #Accept postcode
    print("POST CODE", end = "\n")
    #Validate postcode
#     while True:
#         postcode = int(get_user_input("number"))
#         if (postcode >= 1000 and postcode < 5000):
#             break
#         else:
#             print("Post code not recognized. Try again!", end = "\n")
# The above validation was added based on an assumption.
# But the updated assignment specification invalidates any assumptions
# here, so if postcode is not between 1000 and 5000, generate an 11-digit
# user id.
    # Determine length of user_id based on accepted postcode
    postcode = int(get_user_input("number"))
    if (1000 <= postcode < 2000):
        user_id = generate_user_id(7, user_id_list)
    elif (2000 <= postcode < 3000):
        user_id = generate_user_id(8, user_id_list)
    elif (3000 <= postcode < 4000):
        user_id = generate_user_id(9, user_id_list)
```

```
    elif (4000 <= postcode < 5000):
        user_id = generate_user_id(10, user_id_list)
    else:
        user_id = generate_user_id(11, user_id_list)
    # Add new user id and user list to existing lists
    user_id_list.append(user_id)
    user_list.append([username, password, email, postcode])
    # print(user_id_list)
    return None

    # print(user_id_list, "\n", user_list)

# add_user_to_list([], [])
```

2.6 7. Test function

This function contains the test code using previous defined functions. The test function steps are listed below. You can also add more steps if you need.

Define a user id list.

Define a user list. Each user is also a list which contains username(str type), encrypted password(str type), email(str type) and postcode(str type). The format is like [[username1, password1, email1, postcode1],[username2, password2, email2, postcode2]...].

Add several users by calling add user to list function.

Convert the user id list and user list to a dictionary.

Call the authentication of username and password function.

When a user enters "q", the program can quit. Otherwise, keep asking the user to input and do authentication.

Print out "username password correct" or "username or password incorrect" according to the authentication result.

[ ]:
```
# define your function here

import random

def test():
    # 1. Define user_id_list
    user_id_list = []
    # 2. Define user_list
    user_list = []
    # 3. Add several users
    limit = random.randint(5, 8)
    for i in range(3): #can use limit variable as argument for range method
        print("\nUSER {} DETAILS".format(i+1))
        add_user_to_list(user_id_list, user_list)
```

```
    # 4. Convert the user_id_list and user_list to a dictionary
    user_dict = dict(zip(user_id_list, user_list))
    # print(user_dict)
    """
    5. Authenticating user log in
    6. Keep asking user to input credentials until
        q is entered or correct credentials are received
    """
    print("\nLogIn username", end="\n")
    username = get_user_input("letter")
    print("LogIn password", end="\n")
    password = get_user_input("letter_or_number_or_underscore")
    while(authenticate(username, password, user_dict) == False ):
        flag = input\
        ("username or password incorrect. Press enter to try again or press q␣
↪to quit: ")
        while(flag != "q" and len(flag) != 0): # neither enter nor q is pressed
            flag = input("Invalid input! Press", \
            " enter to try again or press q to quit: ")
        # q is pressed
        if(flag == "q"):
            print("Goodbye!")
            return None
        # enter is pressed ==> length of flag is 0
        elif(len(flag) == 0):
            print("LogIn username", end="\n")
            username = get_user_input("letter")
            print("LogIn password", end="\n")
            password = get_user_input("letter_or_number_or_underscore")
    # Authentication successful
    print("username password correct. Welcome, {}!".format(username))
    # print(user_dict)
    return None
```

[ ]:
```
# run the test function here

if __name__ == "__main__":
    test()
```