FIT9136 Algorithm and programming foundation in Python

Assignment 1

Lecturer in Charge: Shirin Ghaffarian Maghool

March 2022

Table of Contents

1. Key Information

Purpose	This assessment is related to the following learning objectives (LO): ● **LO1:** Apply best practice Python programming constructs for solving computational problems
Your task	It is Individual assignment, where you will write Python code for a simple emulation of collecting user information from the console. Your application should be able to ask the user to input username, password, email and postcode and save the information into a collection type.
Value	**15**% of your total marks for the unit
Due Date	**[Friday 1st April 2022, week 5] 4:30 pm**
Submission	● Via Moodle Assignment Submission. ● Turnitin will be used for similarity checking of all submissions.
Assessment Criteria	See Moodle Assessment page
Late Penalties	● 10% deduction of the available mark per calendar day or part thereof for up to one week ● Submissions more than 7 calendar days after the due date will receive a mark of zero (0) and no assessment feedback will be provided.
Support Resources	See Moodle Assessment page
Feedback	Feedback will be provided on student work via: ● general cohort performance ● specific student feedback ten working days post submission

1.1. Do and Do NOT

Do	Do NOT
• Maintain academic integrity[1] • Get support early from this unit and other services in the university • Apply for special consideration for extensions[2] if needed (optional)	• Leave your assignment in draft mode (assignment in draft mode will not be marked) • Submit late (10% daily penalty applies)[3] • Submission is not accepted after 7 days of the due date, unless you have special consideration.

1.2. Documentation

Commenting your code is essential as part of the assessment criteria (refer to Marking rubrics).

You should also include comments at the beginning of your program file, which specify your name, your Student ID, the start date and the last modified date of the program, as well as with a high-level description of the program. In-line comments within the program are also part of the required documentation.

1.3. Submission

Your have to submit your assignment via the assignment submission link (i.e., "Assignment 1 Submission") on the Moodle site by the deadline specified in Section 1, i.e. **1st April 2022 (Friday) by 4:30 pm**:

- There will be NO hard copy submission required for this assignment.
- You are required to submit your assignment as a .zip file named with your Student ID. For example, if your Student ID is 12345678, you would submit a zipped file named 12345678.zip
- Do not include any unnecessary file in this folder
- Note that marks will be deducted if this requirement is not strictly complied with.

---

1 https://www.monash.edu/rlo/research-writing-assignments/referencing-and-academic-integrity/academic-integrity

2 https://www.monash.edu/exams/changes/special-consideration (**All the Special Consideration should be applied no later than two University working days after the due date of the affected assessment**).

3 eg: original mark was 70/100, submitting 2 days late results in 50/100 (10 marks off). This includes weekends

- No submission accepted via email.
- Please note we **cannot mark any work on the FITGitLab Server**, you need to ensure that you submit via Moodle where you need to accept the student declaration. It is your responsibility to **ENSURE** that the submitted files are the correct files. We strongly recommend after uploading a submission, and prior to actually submitting in Moodle, that you download the submission and double-check its contents. Marks will be deducted for any of the requirements that are not complied with.

1.4. Deliverables

Your submission should contain the following documents:

- One **.ipynb file**.
- One **.pdf file** (The pdf file cannot be an image pdf file. Make sure all the text in the pdf file can be selected and copied).
- **Marks will be deducted for any of these requirements that are not strictly complied with.**

2. Getting help

2.1. English language skills

if you don't feel confident with your English.

- Talk to English Connect: https://www.monash.edu/english-connect

2.2. Study skills

If you feel like you just don't have enough time to do everything you need to, maybe you just need a new approach

- Talk to a learning skills advisor: https://www.monash.edu/library/skills/contacts

2.3. Things are tough right now

Everyone needs to talk to someone at some point in their life, no judgement here.

- Talk to a counsellor: https://www.monash.edu/health/counselling/appointments (friendly, approachable, confidential, free)

2.4. Things in the unit don't make sense

Even if you're not quite sure what to ask about, if you're not sure you won't be alone, it's always better to ask.

- Ask in Ed: https://edstem.org/au/courses/7429/discussion/
- Attend a consultation: https://lms.monash.edu/course/view.php?id=135703&section=21

2.5. I don't know what I need

Everyone at Monash University is here to help you. If things are tough now they won't magically get better by themselves. Even if you don't exactly know, come and talk with us and we'll figure it out. We can either help you ourselves or at least point you in the right direction.

3. Key tasks (100 marks)

This assignment is a simple emulation of collecting user information from the console. Your application should be able to ask the user to input username, password, email and postcode and save the information into a collection type.

This is an individual assignment and must be your own work.

The allowed libraries are **random and math**. You will receive penalties if you use any other libraries.

3.1. 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".

3.2. 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.
- Use the 3 characters to encode the input string. The first_character is used one time, the second_character is used 2 times and the third_character is used 3 times. Repeat the encryption process every 3 letters of input string until the end.
- Start character “^^^” and End character “$$$” for the final encrypted string.

Example input string: “password”
Example first_character: “)”
Example second_character: “$”
Example third_character: “)”
Encrypted result: “^^^)p)$$a$$)))s))))s)$$w$$)))o))))r)$$d$$$$$”

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

3.3. 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.

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.

3.5. 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.

For example, the username=”aaaa”, password=”12333”, user_dict={“12345”: [“aaaa”, “^^^&1&!!2!!&&&3&&&&3&!!3!!$$$”, “aa@gmail.com”, “3151”], “34567”:[“bbbbb”, “^^^%1%%%2%%%%%2%%%%2%$$$”, “bb@gmail.com”, “3000”]}, the authentication result will be True. If the username=”bbbbb”, password=”12333”, the authentication result will be False.

3.6. 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*.

3.7. 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.

1. Define a user id list
2. 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]...].
3. Add several users by calling add user to list function
4. Convert the user id list and user list to a dictionary
5. Call the authentication of username and password function
6. When a user enters “q”, the program can quit. Otherwise, keep asking the user to input and do authentication.
7. Print out “username password correct” or “username or password incorrect” according to the authentication result.

Important Notes:

- If any exception/errors happens when running each function, you will lose 50% of allocated function logic marks. For example, if the get user input function returns any error, then maximum mark you can get is 5% instead of 10% in the function logic.
- Add proper validations and output messages to make your code more user friendly to end users.
- Run the test function at the end of your file to show that your program works fine.