## 2. Instruction

For this assignment, your goal is to create a **Student Information Management System** that utilizes two files named "user.txt" and "unit.txt" located in the "data" folder. The application will feature multiple user roles (i.e., Admin, Teacher, Student), each with varying levels of access to different operations, including but not limited to:

- For all users:
    - Login to the application, After a user logs in, the available sets of operations will vary depending on the user's assigned role.
    - Exit the application
- For admin:
    - Search user information
    - List all users' information
    - List all units' information
    - Enable/Disable user
    - Add/Delete user
    - Log out
- For teacher:
    - List all teaching units information
    - Add/Delete a unit
    - List all students' information and scores of one unit
    - Show the avg/max/min score of one unit
    - Log out
- For student:
    - List all available units information
    - List all enrolled units, each students can enrol maximum 3 units
    - Enrol/Drop a unit
    - Check the score of a unit
    - Generate score
    - Log out

Please note that logging out under Admin, Teacher, or Student accounts will not exit the program. After logging out, the program will return to the main menu and prompt the user to log in again. The program will continue to run until the user chooses to exit the application.

To develop this application, you will need to create five classes, each defined in a separate Python file (i.e. *.py), and one main Python file to execute the program. All the required files can be found in the **A2_student_template.zip** file, which is located in the Assessments section on Moodle. **It is important to note that the addition of supplementary files/classes exceeding the prescribed six may incur a penalty in the form of mark deduction.**

## 2.1. Unit Class

| Contains all the operations related to a unit. | |
|---|---|
| **Required Class Variables** | ● N/A (You can add if you need) |
| **Required Methods** | |

| **2.1.1. __init__()**<br>Constructs a unit object. | |
|---|---|
| **Arguments** | ● *unit_id* (must be unique integer. E.g., 1111111)<br>● *unit_code (*must be unique. E.g., FIT9136*)*<br>● *unit_name*<br>● *unit_capacity* (Each unit has a maximum enrol capacity) |
| **Returns** | ● N/A |

**All arguments of the constructor must have a default value.*

| **2.1.2. __str__()**<br>Return the unit information as a formatted string. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● a string in the following format:<br>"unit_id, unit_code, unit_name, unit_capacity" |

| **2.1.3. generate_unit_id()**<br>Return a unique unit id (7 digits number) | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● an integer number consisting of 7 digits |

***Note:***
- *All the units are saved in the file "data/unit.txt".*

## 2.2. User Class

| Contains all the operations related to a user. | |
|---|---|
| **Required** | ● N/A (You can add if you need) |

| | |
|---|---|
| **Class Variables** | |
| **Required Methods** | |

| **2.2.1. __init__()**<br>Constructs a user object. | |
|---|---|
| **Arguments** | ● *user_id* (must be unique integer)<br>● *user_name* (must be unique and can only consist of numbers, letters, and underscores)<br>● *user_password* (must be encrypted)<br>● *user_role* (can only be one of the following three options: 'AD' for 'admin', 'TA' for 'teacher', or 'ST' for 'student')<br>● *user_status* (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) |
| **Returns** | ● N/A |

**All arguments of the constructor must have a default value.*

| **2.2.2. __str__()**<br>Return the user information as a formatted string. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● a string in the following format:<br>"user_id, user_name, user_password, user_role, user_status" |

| **2.2.3. generate_user_id()**<br>Return a unique user id (5 digits number) | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● an integer number consisting of 5 digits |

| **2.2.4. check_username_exist()**<br>Return a boolean value to indicate username existence. | |
|---|---|
| **Arguments** | ● *user_name* |
| **Returns** | ● True (exist) / False (not exist) |

| **2.2.5. encrypt()** |
|---|

Encode a user-provided password. Use the two provided strings *str_1* and *str_2* as encryption character pools.
<u>Encryption steps:</u>
For each letter in the user- provided password:

1. Get the ASCII code number of the letter using ord().
2. Get the remainder of the ASCII code number divided by the length of the *str_1*.
3. Use the remainder as an index to locate a character in *str_1*.
4. Get the remainder of the letter index in the user- provided password divided by the length of the *str_2*.
5. Use the remainder as an index to locate a character in str_2.
6. The characters obtained from step 3 and step 5 are used to encrypt the letter.

Finally, add "^^^" at the beginning and "$$$" at the end of the encrypted password to indicate its beginning and ending respectively.

| | |
|---|---|
| **Arguments** | • *user_password* |
| **Local Variables** | • *str_1* = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"<br>• *str_2* = "!#$%&()*+-./:;<=>?@\^_`{\|}~" |
| **Returns** | • an encrypted password string. |

***<u>Examples:</u>***

- *User-provided password: "password"*
  *Encrypted password string: "^^^Y!J#2$2%6&X(1)M*$$$"*
- *User-provided password: "abcd1234"*
  *Encrypted password string: "^^^J!K#L$M%X&Y(Z)1*$$$"*

**2.2.6. login()**
Authenticate a user login attempt.

| | |
|---|---|
| **Arguments** | • *user_name*<br>• *user_password* |
| **Returns** | • a user information string (obtained from the “user.txt” file). |

**If the user does not exist or their status is 'disabled', then return None.*

***<u>Note:</u>***

- *All the users are saved in the file "data/user.txt".*

## 2.3. UserAdmin Class

|  |  |
|---|---|
| Contains all the operations related to an admin. This class inherits from the User class. | |
| **Required Class Variables** | • N/A (You can add if you need) |
| **Required Methods** | |

|  |  |
|---|---|
| **2.3.1. __init__()**<br>Constructs an admin object. | |
| **Arguments** | • *user_id* (must be unique integer)<br>• *user_name* (must be unique and can only consist of numbers, letters, and underscores)<br>• *user_password* (must be encrypted)<br>• *user_role* (can only be 'AD')<br>• *user_status* (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) |
| **Returns** | • N/A |

**All arguments of the constructor must have a default value.*

|  |  |
|---|---|
| **2.3.2. __str__()**<br>Return the admin information as a formatted string. | |
| **Arguments** | • N/A |
| **Returns** | • a string in the following format:<br>"user_id, user_name, user_password, user_role, user_status" |

|  |  |
|---|---|
| **2.3.3. admin_menu()**<br>Display a list of all operations that can only be performed by an admin. | |
| **Arguments** | • N/A |
| **Returns** | • N/A |

|  |  |
|---|---|
| **2.3.4. search_user()**<br>Display the information of the user found by the search. | |
| **Arguments** | • *user_name* |
| **Returns** | • N/A |

| **2.3.5. list_all_users()**<br>Display the information of all users currently stored in the system | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● N/A |

| **2.3.6. list_all_units()**<br>Display the information of all units currently stored in the system. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● N/A |

| **2.3.7. enable_disable_user()**<br>Update a user's status by changing it from enabled to disabled or from disabled to enabled, depending on the user's current status. | |
|---|---|
| **Argument** | ● *user_name* |
| **Returns** | ● N/A |

| **2.3.8. add_user()**<br>Add a user to the system. All the users should be persisted to the *user.txt* file. | |
|---|---|
| **Argument** | ● *user_obj (An instance of the UserTeacher or UserStudent class)* |
| **Returns** | ● N/A |

| **2.3.9. delete_user()**<br>Delete the user that was found by the search. | |
|---|---|
| **Arguments** | ● *user_name* |
| **Returns** | ● N/A |

***Note:***

- *Output a message that indicates that the user cannot be found if they are not found in the system.*

## 2.4. UserTeacher Class

| Contains all the operations related to a teacher. This class inherits from the User class. | |
|---|---|
| **Required Class Variables** | ● N/A (You can add if you need) |
| **Required Methods** | |

| **2.4.1. __init__()**<br>Constructs a teacher object. | |
|---|---|
| **Arguments** | ● *user_id* (must be unique integer)<br>● *username* (must be unique and can only consist of numbers, letters, and underscores)<br>● *user_password* (must be encrypted)<br>● *user_role* (can only be 'TA')<br>● *user_status* (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in)<br>● *teach_units* (A list of units code taught by the teacher) |
| **Returns** | ● N/A |

**All arguments of the constructor must have a default value.*

| **2.4.2. __str__()**<br>Return the teacher information as a formatted string. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● a string in the following format:<br>"user_id, user_name, user_password, user_role, user_status, teach_units" |

| **2.4.3. teacher_menu()**<br>Display a list of all operations that can only be performed by a teacher. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● N/A |

| **2.4.4. list_teach_units()** |
|---|

| Display the information of all units that are taught by the current teacher. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● N/A |

**You can use regex to extract information from the data/user.txt .*
**Output a message indicating that no units taught by the teacher are found in the system, if applicable.*

| **2.4.5. add_teach_unit()** Add a new unit information to the data/unit.txt and add the unit_code in the current teacher's 'teach_units' list | |
|---|---|
| **Arguments** | ● *unit_obj (An instance of the Unit class)* |
| **Returns** | ● N/A |

*Both the information for the units and the information for the users,which are stored in the 'unit.txt' and 'user.txt' files, respectively, require updating.

| **2.4.6. delete_teach_unit()** Delete a unit from the current teacher's 'teach_units' list. If this unit has been enrolled by students, remove all associated enrollment records as well. | |
|---|---|
| **Arguments** | ● *unit_code* |
| **Returns** | ● N/A |

*Both the information for the units and the information for the users,which are stored in the 'unit.txt' and 'user.txt' files, respectively, require updating.

| **2.4.7. list_enrol_students()** Display the information of all students currently enrolled in the unit. | |
|---|---|
| **Arguments** | ● *unit_code* |
| **Returns** | ● N/A |

**If no students are found enrolled in the unit within the system, please output a message to that effect.*

| **2.4.8. show_unit_avg_max_min_score()** Display the unit’s average, maximum and minimum score. | |
|---|---|
| **Argument** | ● *unit_code* |

| | |
|---|---|
| | **Returns** \| ● N/A |

## 2.5. UserStudent Class

| Contains all the operations related to a student. This class inherits from the User class. | |
|---|---|
| **Required Class Variables** | ● N/A (You can add if you need) |
| **Required Methods** | see below |

| **2.5.1. __init__()**<br>Constructs a student object. | |
|---|---|
| **Arguments** | ● *user_id* (must be unique integer)<br>● *user_name* (must be unique and can only consist of numbers, letters, and underscores)<br>● *user_password* (must be encrypted)<br>● *user_role* (can only be 'ST')<br>● *user_status* (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in)<br>● *enrolled_units (list of tuples (unit_code, score)). The score default is -1.* |
| **Returns** | ● N/A |

**All arguments of the constructor must have a default value.*

| **2.5.2. __str__()**<br>Return all the student's attributes as a formatted string. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● a string in the following format:<br>"user_id, user_name, user_password, user_role, user_status, enrolled_units" |

| **2.5.3. student_menu()**<br>Display a list of all operations that can only be performed by a student. | |
|---|---|
| **Arguments** | ● N/A |

| | |
|---|---|
| **Returns** | ● N/A |

| **2.5.4. list_available_units()**<br>Display all the units that can be enrolled by the current student. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● N/A |

| **2.5.5. list_enrolled_units()**<br>Display all the units that the student enrolled. | |
|---|---|
| **Arguments** | ● N/A |
| **Returns** | ● N/A |

| **2.5.6. enrol_unit()**<br>Enrol the current student into a unit. One student can enrol a maximum of 3 units and each unit has its own capacity. After enrollment, initialise the score as -1. | |
|---|---|
| **Argument** | ● *unit_code* |
| **Returns** | ● N/A |

** If enrollment is unsuccessful, display some messages to guide users.*

| **2.5.7. drop_unit()**<br>Remove the unit from the list of units in which the student is currently enrolled | |
|---|---|
| **Argument** | ● *unit_code* |
| **Returns** | ● N/A |

| **2.5.8. check_score()**<br>Display the unit score. | |
|---|---|
| **Argument** | ● *unit_code* |
| **Returns** | ● N/A |

*If the input 'unit_code' field is empty, then display the scores for all units in which the student is enrolled

**2.5.9. generate_score()**
A random score between 0 and 100 (inclusive) should be generated for a unit. This resulting score should then be added to the student's list of enrolled units in the 'user.txt' file.

| | |
|---|---|
| **Arguments** | ● *unit_code* |
| **Returns** | ● N/A |

## 2.6. Main File

In this file, you will be creating three functions: main_menu(), generate_test_data(), and main().

- The main_menu() function will display all available operations for users to choose from.
- The generate_test_data() function will generate test data for the program, including one admin user (with the username "admin" and password "password"), three units, three teachers (each allocated to one unit), and ten students (all of whom are enrolled in these three units). Note that students can enrol in more than one unit. This function will be called when the program starts and all the files will be overwritten by the newly generated test data.
- The main() function will handle all program logic, including user input, calling class methods, and handling validations.

The design and implementation is up to you but must include the menu items outlined in section 2 using the classes and methods outlined.

**You must ensure that your menu and control logic handles exceptions appropriately.**

You can break down your code to several functions if you wish but you still need to call the extra defined functions in the main function. In the if __name__==”__main__” part, only call main() function. Your tutor will only run your main.py file. For each operation that the user performs, try to give enough instructional messages.

In regards to the tasks listed above, it is acceptable to modify the function and method names as well as the variable names to adhere to your preferred naming conventions. Additionally, you are permitted to add more class variables, methods in classes and functions in the main file. However, it is crucial to ensure that all necessary methods and functions have been implemented and that they have been invoked in the application. Any unused code present in the application will lead to mark deductions.

**Please keep in mind:**

- **In terms of validation, it is essential to perform all validation in the main file outside of any methods. It is not acceptable to include any validation inside of methods unless it is required locally. Additionally, all operations involving data should be updated to files immediately.**
- **It is also important to understand the concept of inheritance. The child class inherits all the methods and attributes from the parent class, and the idea of overriding or adding new methods comes into play when additional methods or attributes are required in the child class.**
- **Furthermore, the format of the data saved in *user.txt* and *unit.txt* should conform to the format specified in the** str() **method for that specific object.**

## 2.7. User Manual

Please provide user instructions in a file named **README.pdf**, which should explain how to run your application. Your marker will use these instructions to run the application, so please ensure they are clear and easy to follow. Please be concise in your writing and aim to keep the README.pdf document to no more than one page in length.