



FIT9136 Algorithms and Programming Foundations in Python

Assignment 2

Apr 2023

Table of Contents

[1. Key Information](#)

[2. Instruction](#)

[2.1. Unit Class](#)

[2.2. User Class](#)

[2.3. UserAdmin Class](#)

[2.4. UserTeacher Class](#)

[2.5. UserStudent Class](#)

[2.6. Main File](#)

[2.7. User Manual](#)

[3. Group Communication](#)

[3.1. Effective team size and team size factor](#)

[4. Git Management](#)

[5. Do and Do NOT](#)

[5.1. Important NOTES](#)

[6. Submission Requirements](#)

[7. Academic Integrity](#)

[8. Marking Guide](#)

[9. Getting help](#)

[9.1. English language skills](#)

1. Key Information

Purpose	<p>This assignment will develop your skills in designing, constructing, testing, and documenting a small Python program according to specific programming standards. This assessment is related to the following learning outcome (LO):</p> <ul style="list-style-type: none">• LO2 - Restructure a computational program into manageable units of modules and classes using the object-oriented methodology• LO3 - Demonstrate Input/Output strategies in a Python application and apply appropriate testing and exception handling techniques
Your task	<p>This assignment is a group task where students will work in randomly selected groups of two or three. Your task is to develop a simple student information management system using Python code, based on the provided specifications.</p>
Value	<p>30% of your total marks for the unit.</p>
Due Date	<p>Task Submission: Week 9 - Friday, 5th May 2023, 4:30 pm (AEDT) Self and Group Evaluation (Feedback Fruit): Friday, 12th May 2023, 4:30:55 pm (AEST)- (no late submission permitted)</p>
Submission	<ul style="list-style-type: none">• Via Moodle Assignment Submission.• FIT GitLab check-ins will be used to assess the history of development• JPlag will be used for similarity checking of all submissions.
Assessment Criteria	<p>This assessment includes a compulsory interview with your tutor following the submission date. At the interview you will be asked to explain your code/design/testing, modify your code, and discuss your design decisions and alternatives. Marks will not be awarded for any section of code/design/functionality that you cannot explain satisfactorily. Failure to attend the interview will result in your assessment not being marked. You will be provided with the timing of the interviews at a later date.</p> <p>The following aspects will be assessed:</p> <ol style="list-style-type: none">1. Program functionality in accordance to the requirements2. Code Architecture and Adherence to Python coding standards3. The comprehensiveness of documented code
Late Penalties	<ul style="list-style-type: none">• 10% deduction 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.• 0 marks for peer evaluation component (see marking guide) if the Self and Group Evaluation is not completed by the due date (no late submission permitted)
Support Resources	<p>See Moodle Assessment page and Section 7 in this document</p>
Feedback	<p>Feedback will be provided on student work via</p> <ul style="list-style-type: none">• general cohort performance• specific student feedback ten working days post submission

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	<ul style="list-style-type: none">• N/A (You can add if you need)																						
Required Methods	<table><tr><td colspan="2">2.1.1. __init__() Constructs a unit object.</td></tr><tr><td>Arguments</td><td><ul style="list-style-type: none">• <i>unit_id</i> (must be unique integer. E.g., 1111111)• <i>unit_code</i> (must be unique. E.g., FIT9136)• <i>unit_name</i>• <i>unit_capacity</i> (Each unit has a maximum enrol capacity)</td></tr><tr><td>Returns</td><td><ul style="list-style-type: none">• N/A</td></tr><tr><td colspan="2"><i>*All arguments of the constructor must have a default value.</i></td></tr><tr><td colspan="2">2.1.2. __str__() Return the unit information as a formatted string.</td></tr><tr><td>Arguments</td><td><ul style="list-style-type: none">• N/A</td></tr><tr><td>Returns</td><td><ul style="list-style-type: none">• a string in the following format: "unit_id, unit_code, unit_name, unit_capacity"</td></tr><tr><td colspan="2">2.1.3. generate_unit_id() Return a unique unit id (7 digits number)</td></tr><tr><td>Arguments</td><td><ul style="list-style-type: none">• N/A</td></tr><tr><td>Returns</td><td><ul style="list-style-type: none">• an integer number consisting of 7 digits</td></tr><tr><td colspan="2">Note:<ul style="list-style-type: none">• All the units are saved in the file "data/unit.txt".</td></tr></table>	2.1.1. __init__() Constructs a unit object.		Arguments	<ul style="list-style-type: none">• <i>unit_id</i> (must be unique integer. E.g., 1111111)• <i>unit_code</i> (must be unique. E.g., FIT9136)• <i>unit_name</i>• <i>unit_capacity</i> (Each unit has a maximum enrol capacity)	Returns	<ul style="list-style-type: none">• N/A	<i>*All arguments of the constructor must have a default value.</i>		2.1.2. __str__() Return the unit information as a formatted string.		Arguments	<ul style="list-style-type: none">• N/A	Returns	<ul style="list-style-type: none">• a string in the following format: "unit_id, unit_code, unit_name, unit_capacity"	2.1.3. generate_unit_id() Return a unique unit id (7 digits number)		Arguments	<ul style="list-style-type: none">• N/A	Returns	<ul style="list-style-type: none">• an integer number consisting of 7 digits	Note: <ul style="list-style-type: none">• All the units are saved in the file "data/unit.txt".	
2.1.1. __init__() Constructs a unit object.																							
Arguments	<ul style="list-style-type: none">• <i>unit_id</i> (must be unique integer. E.g., 1111111)• <i>unit_code</i> (must be unique. E.g., FIT9136)• <i>unit_name</i>• <i>unit_capacity</i> (Each unit has a maximum enrol capacity)																						
Returns	<ul style="list-style-type: none">• N/A																						
<i>*All arguments of the constructor must have a default value.</i>																							
2.1.2. __str__() Return the unit information as a formatted string.																							
Arguments	<ul style="list-style-type: none">• N/A																						
Returns	<ul style="list-style-type: none">• a string in the following format: "unit_id, unit_code, unit_name, unit_capacity"																						
2.1.3. generate_unit_id() Return a unique unit id (7 digits number)																							
Arguments	<ul style="list-style-type: none">• N/A																						
Returns	<ul style="list-style-type: none">• an integer number consisting of 7 digits																						
Note: <ul style="list-style-type: none">• All the units are saved in the file "data/unit.txt".																							

2.2. User Class

Contains all the operations related to a user.	
Required	<ul style="list-style-type: none">• N/A (You can add if you need)

Class Variables																													
Required Methods	<table> <tr> <td colspan="2"> 2.2.1. __init__() Constructs a user object. </td></tr> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> • <i>user_id</i> (must be unique integer) • <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores) • <i>user_password</i> (must be encrypted) • <i>user_role</i> (can only be one of the following three options: 'AD' for 'admin', 'TA' for 'teacher', or 'ST' for 'student') • <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> • N/A </td></tr> <tr> <td colspan="2"> <i>*All arguments of the constructor must have a default value.</i> </td></tr> <tr> <td colspan="2"> 2.2.2. __str__() Return the user information as a formatted string. </td></tr> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> • N/A </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> • a string in the following format: "user_id, user_name, user_password, user_role, user_status" </td></tr> <tr> <td colspan="2"> 2.2.3. generate_user_id() Return a unique user id (5 digits number) </td></tr> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> • N/A </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> • an integer number consisting of 5 digits </td></tr> <tr> <td colspan="2"> 2.2.4. check_username_exist() Return a boolean value to indicate username existence. </td></tr> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> • <i>user_name</i> </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> • True (exist) / False (not exist) </td></tr> <tr> <td colspan="2"> 2.2.5. encrypt() </td></tr> </table>	2.2.1. __init__() Constructs a user object.		Arguments	<ul style="list-style-type: none"> • <i>user_id</i> (must be unique integer) • <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores) • <i>user_password</i> (must be encrypted) • <i>user_role</i> (can only be one of the following three options: 'AD' for 'admin', 'TA' for 'teacher', or 'ST' for 'student') • <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) 	Returns	<ul style="list-style-type: none"> • N/A 	<i>*All arguments of the constructor must have a default value.</i>		2.2.2. __str__() Return the user information as a formatted string.		Arguments	<ul style="list-style-type: none"> • N/A 	Returns	<ul style="list-style-type: none"> • a string in the following format: "user_id, user_name, user_password, user_role, user_status" 	2.2.3. generate_user_id() Return a unique user id (5 digits number)		Arguments	<ul style="list-style-type: none"> • N/A 	Returns	<ul style="list-style-type: none"> • an integer number consisting of 5 digits 	2.2.4. check_username_exist() Return a boolean value to indicate username existence.		Arguments	<ul style="list-style-type: none"> • <i>user_name</i> 	Returns	<ul style="list-style-type: none"> • True (exist) / False (not exist) 	2.2.5. encrypt()	
2.2.1. __init__() Constructs a user object.																													
Arguments	<ul style="list-style-type: none"> • <i>user_id</i> (must be unique integer) • <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores) • <i>user_password</i> (must be encrypted) • <i>user_role</i> (can only be one of the following three options: 'AD' for 'admin', 'TA' for 'teacher', or 'ST' for 'student') • <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) 																												
Returns	<ul style="list-style-type: none"> • N/A 																												
<i>*All arguments of the constructor must have a default value.</i>																													
2.2.2. __str__() Return the user information as a formatted string.																													
Arguments	<ul style="list-style-type: none"> • N/A 																												
Returns	<ul style="list-style-type: none"> • a string in the following format: "user_id, user_name, user_password, user_role, user_status" 																												
2.2.3. generate_user_id() Return a unique user id (5 digits number)																													
Arguments	<ul style="list-style-type: none"> • N/A 																												
Returns	<ul style="list-style-type: none"> • an integer number consisting of 5 digits 																												
2.2.4. check_username_exist() Return a boolean value to indicate username existence.																													
Arguments	<ul style="list-style-type: none"> • <i>user_name</i> 																												
Returns	<ul style="list-style-type: none"> • 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.

Encryption steps:

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	<ul style="list-style-type: none">• <i>user_password</i>
Local Variables	<ul style="list-style-type: none">• <i>str_1</i> = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"• <i>str_2</i> = "!#\$%&()*+,-./:;<=>?@\^_`{ }~"
Returns	<ul style="list-style-type: none">• an encrypted password string.

* Examples:

- 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	<ul style="list-style-type: none">• <i>user_name</i>• <i>user_password</i>
Returns	<ul style="list-style-type: none">• a user information string (obtained from the "user.txt" file).

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

Note:

- 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	<ul style="list-style-type: none">• N/A (You can add if you need)																										
Required Methods	<table><tr><td colspan="2">2.3.1. __init__() Constructs an admin object.</td></tr><tr><td>Arguments</td><td><ul style="list-style-type: none">• <i>user_id</i> (must be unique integer)• <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores)• <i>user_password</i> (must be encrypted)• <i>user_role</i> (can only be 'AD')• <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in)</td></tr><tr><td>Returns</td><td><ul style="list-style-type: none">• N/A</td></tr><tr><td colspan="2"><i>*All arguments of the constructor must have a default value.</i></td></tr><tr><td colspan="2">2.3.2. __str__() Return the admin information as a formatted string.</td></tr><tr><td>Arguments</td><td><ul style="list-style-type: none">• N/A</td></tr><tr><td>Returns</td><td><ul style="list-style-type: none">• a string in the following format: "user_id, user_name, user_password, user_role, user_status"</td></tr><tr><td colspan="2">2.3.3. admin_menu() Display a list of all operations that can only be performed by an admin.</td></tr><tr><td>Arguments</td><td><ul style="list-style-type: none">• N/A</td></tr><tr><td>Returns</td><td><ul style="list-style-type: none">• N/A</td></tr><tr><td colspan="2">2.3.4. search_user() Display the information of the user found by the search.</td></tr><tr><td>Arguments</td><td><ul style="list-style-type: none">• <i>user_name</i></td></tr><tr><td>Returns</td><td><ul style="list-style-type: none">• N/A</td></tr></table>	2.3.1. __init__() Constructs an admin object.		Arguments	<ul style="list-style-type: none">• <i>user_id</i> (must be unique integer)• <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores)• <i>user_password</i> (must be encrypted)• <i>user_role</i> (can only be 'AD')• <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in)	Returns	<ul style="list-style-type: none">• N/A	<i>*All arguments of the constructor must have a default value.</i>		2.3.2. __str__() Return the admin information as a formatted string.		Arguments	<ul style="list-style-type: none">• N/A	Returns	<ul style="list-style-type: none">• a string in the following format: "user_id, user_name, user_password, user_role, user_status"	2.3.3. admin_menu() Display a list of all operations that can only be performed by an admin.		Arguments	<ul style="list-style-type: none">• N/A	Returns	<ul style="list-style-type: none">• N/A	2.3.4. search_user() Display the information of the user found by the search.		Arguments	<ul style="list-style-type: none">• <i>user_name</i>	Returns	<ul style="list-style-type: none">• N/A
2.3.1. __init__() Constructs an admin object.																											
Arguments	<ul style="list-style-type: none">• <i>user_id</i> (must be unique integer)• <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores)• <i>user_password</i> (must be encrypted)• <i>user_role</i> (can only be 'AD')• <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in)																										
Returns	<ul style="list-style-type: none">• N/A																										
<i>*All arguments of the constructor must have a default value.</i>																											
2.3.2. __str__() Return the admin information as a formatted string.																											
Arguments	<ul style="list-style-type: none">• N/A																										
Returns	<ul style="list-style-type: none">• a string in the following format: "user_id, user_name, user_password, user_role, user_status"																										
2.3.3. admin_menu() Display a list of all operations that can only be performed by an admin.																											
Arguments	<ul style="list-style-type: none">• N/A																										
Returns	<ul style="list-style-type: none">• N/A																										
2.3.4. search_user() Display the information of the user found by the search.																											
Arguments	<ul style="list-style-type: none">• <i>user_name</i>																										
Returns	<ul style="list-style-type: none">• N/A																										

2.3.5. list_all_users()

Display the information of all users currently stored in the system

Arguments	<ul style="list-style-type: none">• N/A
Returns	<ul style="list-style-type: none">• N/A

2.3.6. list_all_units()

Display the information of all units currently stored in the system.

Arguments	<ul style="list-style-type: none">• N/A
Returns	<ul style="list-style-type: none">• N/A

2.3.7. enable_disable_user()

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	<ul style="list-style-type: none">• <i>user_name</i>
Returns	<ul style="list-style-type: none">• N/A

2.3.8. add_user()

Add a user to the system. All the users should be persisted to the *user.txt* file.

Argument	<ul style="list-style-type: none">• <i>user_obj</i> (An instance of the <i>UserTeacher</i> or <i>UserStudent</i> class)
Returns	<ul style="list-style-type: none">• N/A

2.3.9. delete_user()

Delete the user that was found by the search.

Arguments	<ul style="list-style-type: none">• <i>user_name</i>
Returns	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none"> N/A (You can add if you need) 												
Required Methods	<div> <div> 2.4.1. __init__() Constructs a teacher object. </div> <table> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> <i>user_id</i> (must be unique integer) <i>username</i> (must be unique and can only consist of numbers, letters, and underscores) <i>user_password</i> (must be encrypted) <i>user_role</i> (can only be 'TA') <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) <i>teach_units</i> (A list of units code taught by the teacher) </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> N/A </td></tr> </table> <p><i>*All arguments of the constructor must have a default value.</i></p> <div> <div> 2.4.2. __str__() Return the teacher information as a formatted string. </div> <table> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> N/A </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> a string in the following format: "user_id, user_name, user_password, user_role, user_status, teach_units" </td></tr> </table> <div> <div> 2.4.3. teacher_menu() Display a list of all operations that can only be performed by a teacher. </div> <table> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> N/A </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> N/A </td></tr> </table> <div> 2.4.4. list_teach_units() </div> </div> </div> </div>	Arguments	<ul style="list-style-type: none"> <i>user_id</i> (must be unique integer) <i>username</i> (must be unique and can only consist of numbers, letters, and underscores) <i>user_password</i> (must be encrypted) <i>user_role</i> (can only be 'TA') <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) <i>teach_units</i> (A list of units code taught by the teacher) 	Returns	<ul style="list-style-type: none"> N/A 	Arguments	<ul style="list-style-type: none"> N/A 	Returns	<ul style="list-style-type: none"> a string in the following format: "user_id, user_name, user_password, user_role, user_status, teach_units" 	Arguments	<ul style="list-style-type: none"> N/A 	Returns	<ul style="list-style-type: none"> N/A
Arguments	<ul style="list-style-type: none"> <i>user_id</i> (must be unique integer) <i>username</i> (must be unique and can only consist of numbers, letters, and underscores) <i>user_password</i> (must be encrypted) <i>user_role</i> (can only be 'TA') <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) <i>teach_units</i> (A list of units code taught by the teacher) 												
Returns	<ul style="list-style-type: none"> N/A 												
Arguments	<ul style="list-style-type: none"> N/A 												
Returns	<ul style="list-style-type: none"> a string in the following format: "user_id, user_name, user_password, user_role, user_status, teach_units" 												
Arguments	<ul style="list-style-type: none"> N/A 												
Returns	<ul style="list-style-type: none"> N/A 												

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*

	<table> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> N/A </td></tr> </table>	Returns	<ul style="list-style-type: none"> N/A
Returns	<ul style="list-style-type: none"> N/A 		

2.5. UserStudent Class

Contains all the operations related to a student. This class inherits from the User class.																	
Required Class Variables	<ul style="list-style-type: none"> N/A (You can add if you need) 																
Required Methods	<table> <tr> <td colspan="2"> 2.5.1. __init__() Constructs a student object. </td></tr> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> <i>user_id</i> (must be unique integer) <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores) <i>user_password</i> (must be encrypted) <i>user_role</i> (can only be 'ST') <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) <i>enrolled_units</i> (list of tuples (<i>unit_code</i>, <i>score</i>)). The score default is -1. </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> N/A </td></tr> </table> <p><i>*All arguments of the constructor must have a default value.</i></p> <table> <tr> <td colspan="2"> 2.5.2. __str__() Return all the student's attributes as a formatted string. </td></tr> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> N/A </td></tr> <tr> <td>Returns</td><td> <ul style="list-style-type: none"> a string in the following format: "user_id, user_name, user_password, user_role, user_status, enrolled_units" </td></tr> </table> <table> <tr> <td colspan="2"> 2.5.3. student_menu() Display a list of all operations that can only be performed by a student. </td></tr> <tr> <td>Arguments</td><td> <ul style="list-style-type: none"> N/A </td></tr> </table>	2.5.1. __init__() Constructs a student object.		Arguments	<ul style="list-style-type: none"> <i>user_id</i> (must be unique integer) <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores) <i>user_password</i> (must be encrypted) <i>user_role</i> (can only be 'ST') <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) <i>enrolled_units</i> (list of tuples (<i>unit_code</i>, <i>score</i>)). The score default is -1. 	Returns	<ul style="list-style-type: none"> N/A 	2.5.2. __str__() Return all the student's attributes as a formatted string.		Arguments	<ul style="list-style-type: none"> N/A 	Returns	<ul style="list-style-type: none"> a string in the following format: "user_id, user_name, user_password, user_role, user_status, enrolled_units" 	2.5.3. student_menu() Display a list of all operations that can only be performed by a student.		Arguments	<ul style="list-style-type: none"> N/A
2.5.1. __init__() Constructs a student object.																	
Arguments	<ul style="list-style-type: none"> <i>user_id</i> (must be unique integer) <i>user_name</i> (must be unique and can only consist of numbers, letters, and underscores) <i>user_password</i> (must be encrypted) <i>user_role</i> (can only be 'ST') <i>user_status</i> (must be either 'enabled' or 'disabled'. Only users with the status 'enabled' are allowed to log in) <i>enrolled_units</i> (list of tuples (<i>unit_code</i>, <i>score</i>)). The score default is -1. 																
Returns	<ul style="list-style-type: none"> N/A 																
2.5.2. __str__() Return all the student's attributes as a formatted string.																	
Arguments	<ul style="list-style-type: none"> N/A 																
Returns	<ul style="list-style-type: none"> a string in the following format: "user_id, user_name, user_password, user_role, user_status, enrolled_units" 																
2.5.3. student_menu() Display a list of all operations that can only be performed by a student.																	
Arguments	<ul style="list-style-type: none"> N/A 																

Returns	<ul style="list-style-type: none"> • N/A
----------------	---

2.5.4. list_available_units()

Display all the units that can be enrolled by the current student.

Arguments	<ul style="list-style-type: none"> • N/A
Returns	<ul style="list-style-type: none"> • N/A

2.5.5. list_enrolled_units()

Display all the units that the student enrolled.

Arguments	<ul style="list-style-type: none"> • N/A
Returns	<ul style="list-style-type: none"> • N/A

2.5.6. enrol_unit()

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	<ul style="list-style-type: none"> • <i>unit_code</i>
Returns	<ul style="list-style-type: none"> • N/A

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

2.5.7. drop_unit()

Remove the unit from the list of units in which the student is currently enrolled

Argument	<ul style="list-style-type: none"> • <i>unit_code</i>
Returns	<ul style="list-style-type: none"> • N/A

2.5.8. check_score()

Display the unit score.

Argument	<ul style="list-style-type: none"> • <i>unit_code</i>
Returns	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • <i>unit_code</i>
Returns	<ul style="list-style-type: none"> • 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:

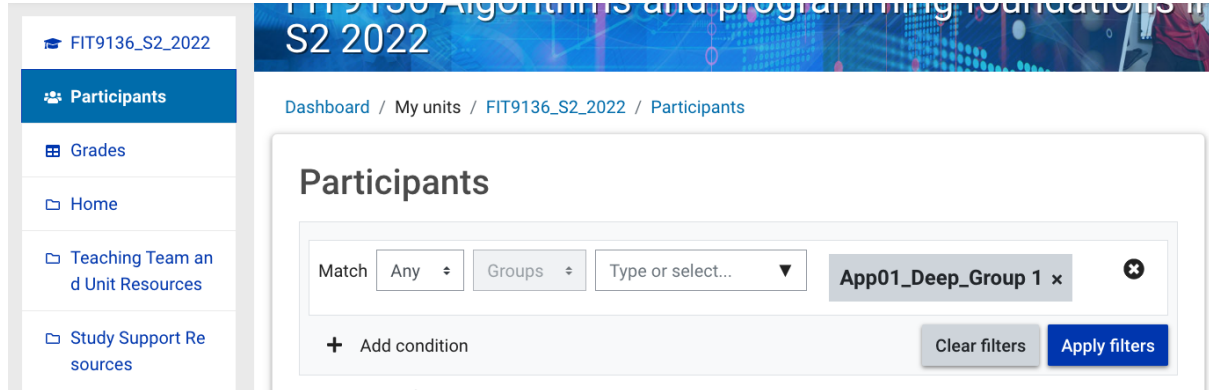
- In terms of validation, it is essential to perform all validation in the main file outside of any methods. It is not acceptable to include any validation inside of methods unless it is required locally. Additionally, all operations involving data should be updated to files immediately.
- It is also 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

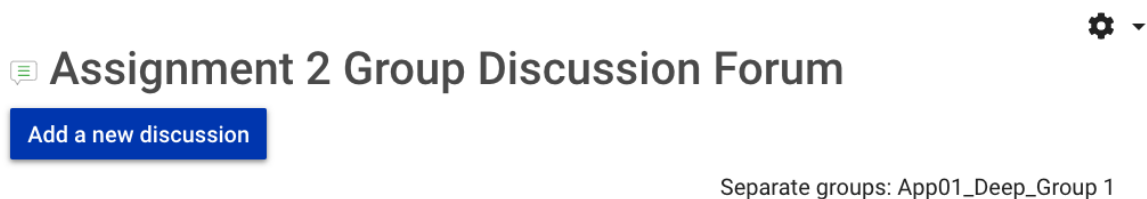
Please provide user instructions in a file named **userManual_group#.pdf**, which should explain how to run your application. Your marker will use these instructions to run the application, so please ensure they are clear and easy to follow. Please be concise in your writing and aim to keep the README.pdf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

3. Group Communication

This is a group assignment. You are going to work in a group of 2-3. You can find your group number, group members, and allocated tutor through Moodle> Participant:



You can communicate with your group members and your allocated tutors using Assignment 2 Group Discussion Forum on Moodle> Assessments> Assignment 2 Group Discussion Forum:



3.1. Effective team size and team size factor

Some teams may have fewer than 3 members. Furthermore, some teammates might be inactive. So the effective size of your team might not be 3. Your TA will determine this with evidence such as the team chat. If the effective team size is:

3, then the team size factor is 100%.

2, then the team size factor is 110%.

1, then the team size factor is 120%.

Your mark will be multiplied by this factor. Inactive students receive 0 for this assignment.

4. Git Management

ENSURE your group number and members names are shown on each .py file as a part of header comment at the top of the file you submit.

GIT STORAGE

Your work **MUST** be saved in your group local working directory (repo) in the Assignment 2 folder and **regularly pushed to the FIT GitLab server** to build a clear history of development of your application. Any submission with less than twelve pushes to the FIT GitLab server will incur a grade penalty. Please note **twelve** pushes is a minimum, in practice we would expect significantly more. **This number of pushes must be evenly distributed amongst group members.** All commits must include a meaningful commit message which clearly describes what the particular commit is about.

Groups must regularly check that their pushes have been successful by logging in to the web interface of the FIT GitLab server; you must not simply assume they are working. Before submission, via Moodle, you must log in to the [web interface of the GitLab server](#) and ensure your submission files are present on the GitLab server.

GIT automatically maintains a history of all files pushed to the server, you do not need to, and **MUST** not, add a version name to your various versions, please ensure you use the same name for all versions of a particular file.

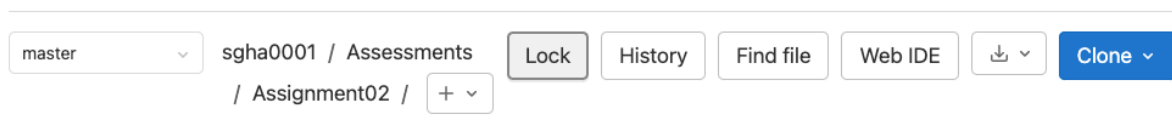
If you have problems in pushing to the remote group repo, you should move your current local group repo out of the way (to a new folder) and then reclone your group repo.

If multiple students work on a .py file at the same time, merging these changes can be quite difficult. For this reason, you are required to take a simple approach to working on the .py file - **lock the remote repo when making changes.**

Whenever a particular student wishes to work on the model, they should go to the Git Server web interface and check if the assignment 2 folder has been locked by another member of the group.

If it has, you must not carry out any work on the assignment task.

If it has not been locked, you can proceed to lock the folder by selecting "Lock":



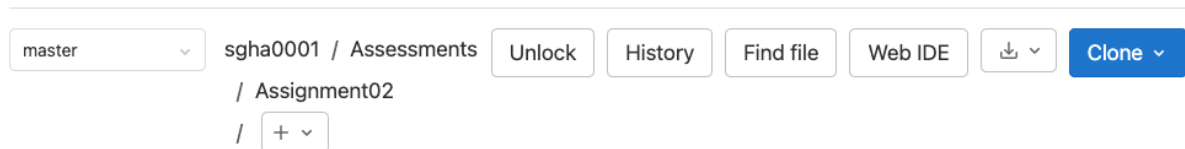
Ensure you are in the correct folder when this lock is applied.

You will know the items are locked as each will have a lock icon attached to it:



If you hover over the padlock icon, you will be able to see who currently has the folder locked.

When you have completed your work, and pushed it to Git, you should return to the Git web interface and unlock the folder:



It is our expectation that all members of the group will contribute to complete the program, just one member must not complete it. In assessing your group's work, we will examine the commit log to ensure all members of the group have participated.

5. Do and Do NOT

Do	Do NOT
<ul style="list-style-type: none">• Maintain appropriate citing and referncing¹,• Get support early from this unit and other services within the university,• Apply for special consideration or for extensions² early if needed.	<ul style="list-style-type: none">• Leave your assignment in draft mode (assignments in draft mode will not be marked),• Submit late (10% daily penalty applies)³,• Attempt to submit after 7 days of the due date (they will not be accepted), unless you have special consideration.

5.1. Important NOTES

- DO NOT use absolute paths and follow the exact structure for the path as provided in the examples in each section. All the path issues that cause your program crash or exception will lead to no mark for functionality.
- You must implement all required methods but you may add additional methods if you require.
- Please make sure your file reading/writing operations include encoding type **utf8**. Changing the application running environment could cause issues if you do not have the encoding type. Any character encoding issues/exceptions will cause serious mark deduction.
- If one method is not working and it hinders the program from continuing running to show other functionalities, the following functionalities will get no mark. For example, if your system can register but cannot login, the functionality that needs to be marked after login will get no mark and you will only get marks for the register part. Therefore, it is important to finish the methods one by one and make sure they can work properly.
- If any exception/errors happen when running your program, you will lose 50% of allocated function logic marks. For example, if the main menu function returns any error, then the maximum mark you can get is 5% instead of 10% in the function logic.
- Add correct validation and output messages to make your code more user-friendly to users.
- The assignment must be done using the **Pycharm, Python Version 3.9**.

¹<https://www.monash.edu/library/help/citing-and-referencing/citing-and-referencing-tutorial>

² <https://www.monash.edu/exams/changes/special-consideration>

³ e.g.: The original mark was 70/100, submitting 2 days late results in 50/100 (20 mark deduction). This includes weekends and public holidays.

- The Python code for this assignment must be implemented according to the [PEP 8-Style Guide for Python Code](#).
- The allowed libraries are **random**, **math**, **os**, **string**, **re**. You will not receive marks for the components if you use any other libraries apart from the mentioned library.
- Commenting on your code is an essential part of the assessment criteria. In addition to inline and function commenting on your code, you should include comments at the beginning of your program file which specify your name, Student ID, the creation date, and the last modified date of the program, as well as a high-level description of the program.
- This assignment cannot be completed in a few days and requires students to apply what we learn each week as we move closer to the submission date. Please remember to show your progress weekly to your tutor.
- You must keep up to date with the Moodle Ed Assignment 2 forum where further clarifications may be posted (this forum is to be treated as your client).
- Please be careful to ensure you do not publicly post anything which includes your reasoning, logic or any part of your work to this forum, doing so violates Monash plagiarism/ collusion rules and has significant academic penalties. Use private posts or email your allocated tutor to raise questions that may reveal part of your reasoning or solution.
- In this Assessments, you must NOT use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task.

6. Submission Requirements

The assignment must be submitted by **Friday, 5th May 2023, 4:30 PM (AEST)**.

The following files are to be submitted and must exist in your Group FITGITLab server repo:

- A series of **.py** files (i.e., unit.py, user.py, user_admin.py, user_student.py, user_teacher.py,, and main.py).
 - A template, A2_student_template.zip, is available on the Moodle Assessments page. **You have to use this template.**
- A **userManual_group#.pdf** file.
- A PDF document of your Group Diary named as Group##_Diary.pdf (replace ## with your group number eg. Group01_Diary.pdf for Group01). A template is available on the Moodle Assessments page to provide a suggested structure for your group diary.

The above files must be compress to a .zip file named **ass2_group#.zip** and submitted via Moodle. **The files only need to be submitted by one member of the group after the group has agreed that the submission is complete and ready to be graded.**

The **.py** files must also have been pushed to your group FIT GitLab server repo with an appropriate history as you developed your solutions. Please ensure your committed comments are meaningful. **Do NOT** need to push the history of userManual_group#.pdf file to the group FIT GitLab server. Only push the final version of PDF file. **DO NOT** push .zip file.

- No submissions will be accepted via email,
- Please note we **cannot mark any work on the GitLab Server**, you need to ensure that you submit correctly via Moodle since it is only in this process that you complete the required student declaration without which work cannot be assessed.
- 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.
- Please **carefully** read the documentation under the “**Special Consideration**” and “**Assignment Task Submission**” on the Moodle Assessments page which covers things such as extensions, correct submission, and resubmission.
- **Please note, if you need to resubmit, you cannot depend on your tutors' availability, for this reason, please be VERY CAREFUL with your submission. It is strongly recommended that you submit several hours before due to avoid such issues.**
- **Marks will be deducted for any of these requirements that are not strictly complied with.**

7. Academic Integrity

Students are expected to be familiar with the [University Academic Integrity Policy](#) and are particularly reminded of the following:

Section 1.9:

Students are responsible for their own good academic practice and must:

- undertake their studies and research responsibly and with honesty and integrity;
- credit the work of others and seek permission to use that work where required;
- not plagiarise, cheat or falsify their work;
- ensure that their work is not falsified;
- not resubmit any assessment they have previously submitted, without the permission of the chief examiner; appropriately acknowledge the work of others;
- take reasonable steps to ensure that other students are unable to copy or misuse their work; and
- be aware of and comply with University regulations, policies and procedures relating to academic integrity.

and **Section 2.9:**

Unauthorised distribution of course-related materials: Students are not permitted to share, sell or pass on to another person or entity external to Monash:

2.9.1 any course material produced by Monash University (such as lecture slides, lecture recordings, class handouts, assessment requirements, examination questions; excluding Handbook entries) as this is a breach of the Copyright Compliance Policy and such conduct may be a copyright law infringement subject to legal action; or

2.9.2 any course-related material produced by students themselves or other students (such as class notes, past assignments), nor to receive such material, without the permission of the chief examiner. The penalties for breaches of academic misconduct include

- a zero mark for the assessment task
- a zero mark for the unit
- suspension from the course
- exclusion from the University.

Where a penalty or disciplinary action is applied, the outcome is recorded and kept for seven years, or for 15 years if the penalty was excluded.Fi

8. Marking Guide

Your work will be marked as per the following:

- Application Functionality - 35 Marks
 - Your tutor will run your main.py to check all the basic operations listed in **section 2. Instruction**.
 - If your program crashes/ any operation won't be implemented correctly, then 0 mark for this criteria.
- Classes Implementation - 10 Marks
 - Unit Class - 2 Marks
 - User Class - 2 Marks
 - UserAdmin Class - 2 Marks
 - UserTeacher Class - 2 Marks
 - UserStudent Class - 2 Marks
- Main File Design - 13 Marks
- User Manual - 2 Marks
- Code Architecture, Style and Documentation - 10 Marks
- Interview - 20 Marks
- Peer Evaluation - 10 Marks
 - Contribution and Participation in your group:
 - Communication
 - Project Management
 - Quality of contribution
 - Quantity of contribution
 - Support for the group's working environmentas assessed by self-evaluation and group members (peer) evaluation.
- Penalty - up to 20 Marks
 - Missing submission requirements

If the effective team size is:

3, then the team size factor is 100%.

2, then the team size factor is 110%.

1, then the team size factor is 120%.

Your mark will be multiplied by this factor. Inactive students receive 0 for this assignment.

9. Getting help

9.1. English language skills

if you don't feel confident with your English.

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

9.2. 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>

9.3. 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)

9.4. 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/8843/discussion/>
- Attend a consultation:
<https://lms.monash.edu/course/view.php?id=141449§ion=21>

9.5. 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.