COMP 636: Python Assessment

Due: **5pm Friday 12th December 2025**

Worth **40%** of COMP636 grade

Submit via Akoraka | Learn

Important note

This is an **individual** assessment. You must not collaborate or discuss your work with others (e.g., telling others exactly what to do, or how to do it, or sharing or debugging others' code, or using ghost writers, etc), but the discussion of general concepts (e.g., how loops work **in general**, not specific to this assessment) is allowed. You may seek clarification and advice from staff.

You MAY use AI tools in the completion of this assignment, this includes but is not limited to: VS Code & Github Copilot ChatGPT, Microsoft Co-Pilot, Google Gemini.

Note that you are expected to be able to describe and explain your program code and the solution you produce.

Ensure you are familiar with the University guidelines and policies on Academic Integrity (see here).

Introduction

Selwyn Dance School provides dance classes to children of different ages and abilities; they have an existing system written in Python that they use to manage the dance school however it has not been well documented and contains some errors. The dance school would also like some new features to be added. Students are in classes based on age or grade. Grades are exams that students have completed.

Note: *The information provided is not exhaustive. You will need to make assumptions, and we expect you to ask clarifying questions.*

The School offers the following Dance classes:

- Glowworms: Students aged 5 years or under.
- Fireflies: Students who are grade 1 or at least 6 years old.
- Butterflies: Students who are grade 2 or at least 7 years old.
- Piwakawaka: Students who are grade 3 or at least 8 and a half years old.
- Robins: Students who are grade 4 or at least 10 years old.

The problems that the School have described include:

1. Students that are new to the school and who should be put into **Piwakawaka** sometimes end up in **Butterflies**.
2. The system doesn't check information input, so sometimes it crashes and sometimes information is recorded with errors. Users should be able to re-enter information until they get it right.

The School has also recently expanded and requested some new features be added to the system:

- A new class called Bellbirds, this is for students who are grade 5 or at least 12 years old.
- A new class called Senior Dance, this is only for students who are grade 6 or above (No entry based on age, only students who are Grade 6 or above are permitted in this class).
- A new report "Dancers Ages" should be added that prints the name of each student and their age today.
- A new report that displays which students are in which class, this report should be grouped by class and students and should be ordered by family name.

Tasks

Complete the following in `sds_admin_your_name.py`:

1. **New Features:** Add the new features requested to the code.
2. **Fix Problems:** Fix the problems identified by the School staff
3. **Comment the Code:** Write comments for each function that describes what it does. You should use the Python # format and comment above the function definition.
4. **Create a README.MD**: In this file include test data that can be entered to test that the system works correctly. This should be presented in a table, one column for each field plus a column to describe the feature that is being tested.
5. **Add a reflection**: In the README.MD file, include a reflection on what was the most difficult aspect of the assessment and why.

Notes:

- The provided `sds_admin_your_name.py` Python file contains a menu structure and partially completed functions. **These functions must not be deleted or renamed**, but you may add arguments/parameters to these functions. You may also add additional functions of your own. Remember to **rename the file to include your name**.
- Data and additional standard functions are provided in `sds_data.py`.
    - **Do not add any functions in `sds_data.py`**
    - **You may alter the data in `sds_data.py`**
    - To view `sds_data.py` in a separate window while editing `sds_admin_your_name.py`, in VS Code choose File > Duplicate Workspace and then display `sds_data.py`.
- One function for menu Option 1 has already been provided for you: `list_all_students()`. This lists details of all students. This is an example of how to produce basic output using the `display_formatted_row()` function in `sds_data.py`.
- Any changes to data will be stored only while the program is running – there is no need to permanently store your changes. When the program restarts, the data will revert to the original data in `sds_data.py`.
- The quality of the user experience will be taken into account, for each of the tasks. Full marks for any item will require validation of data entered – for data type and sensible values (e.g., valid dates) and details in the interface that demonstrate some consideration of what would work well for the user (within the limitations of the terminal window output in VS Code), such as conveniently displaying information when needed or for confirmation of a change.
- You are expected to apply problem solving skills to practically solve issues as they arise.
- You must add comments to your code. These should not be on every line of code but should be written in enough detail so that if you came back to the code in 12 months' time, you could quickly work out what the code is doing. The functions in `sds_data.py`

give an idea of the level of commenting that is expected (excluding the long instructions for `display_formatted_row()`).

Data details

The following array variables are available in `sds_data.py`. These have been imported at the start of `sds_admin_your_name.py`, so the variable name can be used directly, without the `sds_data.` prefix (e.g., just `students`, rather than `sds_data.students`).

`classes` is a dictionary of dance classes indexed by a key value (classname), the entry contains a list that has the id numbers of students in that class.

students is a list of students where each student is represented by a list containing ID, First Name, Family Name, Date of Birth, Grade, Contact Email.

File Download:

Download the following files from the COMP636 **Assessment** block on Akoraka | Learn:

- `sds_admin_your_name.py` – This is the initial code to begin from.
  Include your **own name** in the filename (e.g., `sds_admin_Anna_Lee.py`), and your name and student ID in a comment at the start of the file. **Do not change the menu numbering or existing function names**, although you may add arguments to the function calls and create additional functions of your own in the program overall.
- `sds_data.py` – The student and class data and provided functions. **Do not change the *structure*** of this data, although, you may add extra data.
- Two functions are provided in `sds_data.py`:
  - `unique_id()`: returns the next ID number – to use when adding a new student.
  - `display_formatted_row()`: formats each row of output into columns for consistent output. This must be used for output.

  These functions can be accessed in your program without needing the `'sds_data.'` prefix because they are imported at the top of your code. **Do not change these functions.**

Submission:

Submit (upload) **all** Python `.py` files plus your README.MD for marking as a ZIP file: `sds_admin_your_name.py` (with your name in the file name). Include the `sds_data.py` file.Submit your ZIP file via the submission link on the COMP636 **Assessment** page.

Python Oral

In the Python oral following your submission, you will be asked TWO questions. For each question you will have two minutes to complete your answer.

Question 1 – Explain one new feature in detail

The first question will ask you to explain one of the new features you have implemented in the assessment. The feature to be explained will be selected randomly by you selecting a card.

Question 2 – Explain a Python concept

The second question will ask you to explain one Python concept. You will be asked to randomly select a card relating to one of the following concepts:

1. For loop
2. While loop
3. If statement
4. Lists
5. Dictionaries
6. Function

Marking Rubric

Grade Band	New Features	Fix Problems	Comments	README.MD	Reflection	Oral
A	All features working to an excellent standard	All problems fixed with appropriate error checking and modification of existing code and data structures. No additional problems created.	Detailed comments that clearly explain the function provided for all functions	Extensive test data provided correctly formatted that covers all test scenarios, including the new features	A nuanced a personal reflection that provides insight into the issues encountered comprehensi vely	An excellent explanation of work done and comprehensive understanding of Python concept
B	All features functionally correct	All problems functionally solved	Adequate commenting that describes functionality	Adequate test data provided to test all functions	A personal reflection provided that highlights some issues encountered	A good explanation of work done and a strong understanding of Python concept
C	Majority of features working correctly, only minor issues outstanding	Majority of features working correctly with minor issues outstanding	Comments that do not fully describe or explain functionality	Limited Test data provided	A reflection provided that has little personal insight	A basic understanding of work done or basic understanding of Python concept
D	Many features not working correctly or significant issues outstanding	Many features not working correctly or significant issues outstanding	Limited Comments	Small amount of test data provided	Weak reflection that is generic or lacking in personal insight	Poor understanding of work done or unable to adequately explain Python concept
E	No new Features added	Problems not solved	No comments provided	No test data provided	No reflection provided	Unable to answer questions posed

+/- Movement within the grade band depending on both quality and level of achievement across all components

General requirements

- All data inputs validated to prevent incorrect data entry or program crashes.
- Labels, headings, etc. in user-friendly format, e.g. “Student list” rather than “student_list”
- Helpful prompts for input and error messages.
- `display_formatted_row()` and other provided functions used, and used correctly.

Excellent work would include:

- Full validation of data entered.
- Extra effort for tidy display and user-friendly interface for the user (attention to small details – doesn’t need to be complicated).
- Displaying information useful to the user when they need it and to confirm that updates have been successful (can use functions you have already created for this).