COMP 636: Python Assessment

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.

DO NOT use any AI tools in the completion of this assignment, this includes but is not limited to: ChatGPT, Microsoft Co-Pilot, Google Bard.

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

Introduction

Te Waihora farm management system (FMS) is used to keep a record of the cattle ('stock') on Te Waihora farm.

Cattle are grouped into 'mobs' – in this case called 'R1-2', 'R3' and 'Bulls' – and the FMS records which animals are in which mob. New animals must be added to a mob. The number of mobs may change in future.

The FMS contains details of different paddocks on Te Waihora farm, including the area (size) in hectares (ha), pasture (grass) growth, and the mob and number of cattle in each paddock. The number of paddocks may also change in future.

Mobs are assigned to a paddock, with only one mob allowed in one paddock at a time. The paddock a mob is in can be changed.

The FMS also monitors the pasture (grass) quantities in each paddock – measured as kilograms (kg) of 'dry matter' (DM, the weight of dried grass without its water content). This grows each day and is also eaten by the stock. The total dry matter (DM) is tracked by adding the growth and subtracting the consumption by the stock.

The FMS has an internal 'current date', which can be advanced by one day to forecast pasture levels. Pasture growth and consumption are recalculated when the day is advanced.

Data details

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

- **`stock`** is a list of lists. Each internal list is for a single animal, including: ID number, name of the mob it is in, birthdate (in datetime format), age in years, and weight in kg. For example: `[816, 'R3', datetime(2022,7,15), 2, 558.2]`. Adding a stock animal, adds a new list to the stock list. Each stock ID must be unique.
- **`mobs`** is a dictionary, where the key is the mob name and the value is a list of all stock IDs in that mob, e.g. `"Bulls": [121, 196]`. When a new animal is added, its ID must also be added to the relevant mob.
- **`paddocks`** is a dictionary, where the paddock name is the key and the value is another dictionary, e.g.: `"Corner": {'area':1.4, 'dm/ha': 1850, 'total dm':2590, 'mob':None, 'stock num': 0}`. The dictionary for each paddock contains information relating to paddock size (`'area'`, in hectares (ha)), dry matter per hectare (`'dm/ha'`), total dry matter (`'total dm'`), the mob assigned to that paddock (`'mob'`), and the number of stock in that paddock (`'stock num'`, which is calculated from the number of stock in the mob in that paddock).

*Remember that we will use a different set of data for marking in `farm_data.py` (but it will have the same structure as your data). So **do not hard-code values** because we will mark with different stock, mobs and paddocks.*

File Download:

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

- `farm_mgmt_your_name.py` – This is the initial code to begin from.
  Include your **own name** in the filename (e.g., `farm_mgmt_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.
- `farm_data.py` – The farm data and provided functions. **Do not change the *structure*** of this data. Although, you may add extra data. We will use our own copy of farm_data.py with different data, when marking, but with the same structure as provided.
- Three functions are provided in `farm_data.py`:
  - `next_id()`: returns the next ID number – to use when adding a new animal to the farm
  - `display_formatted_row()`: formats each row of output into columns for consistent output. This must be used for output.
  - `pasture_levels()`: calculates `'total dm'` and `'dm/ha'` values for a paddock

  These functions can be accessed in your program without needing the `'farm_data.'` prefix because they are renamed locally at the top of your code.

Tasks

Add the following features to the system:

1. **Menu enhancement:** Modify the code so that the user can enter an upper- *or* lower-case X (i.e., X or x) to exit the program. The current date should be displayed in the main menu (see Task 6 below)
2. **List stock by mob:** List all of the stock and their details, and group the rows for the stock into mobs. There will be a heading with the mob name, followed by column headers for each column of output, and then by the details of each animal in that mob. Birth date should be presented in an appropriate human readable NZ date format. Mobs will be ordered by mob name.
3. **List paddocks and details:** List all paddocks in alphabetical order by name, showing all of the details with appropriate column headings.
4. **Move mobs between paddocks:** The user specifies a mob and then the paddock that that mob is to be moved into. Mobs can be moved only to empty paddocks and the system should allow a move only if the destination paddock is empty.
5. **Add new stock:** Ask the user to enter details of a new stock animal, which must be saved in the `stock` list:
    a. The new cattle ID must be unique (using provided `next_id()` function)
    b. Birth date should be entered in an appropriate NZ date format and no earlier than the value in `earliest_birth_date` variable.
    c. Age must be calculated from the birth date entered (as whole years as at `current_date`).
    d. Animal weight must be between the values in `weight_range` variable.
    e. The system will keep asking the user to add new animals until the user enters an option to return to the main menu.
    f. After adding each new stock animal, update the stock ID numbers in `mobs` and the `'stock num'` in each paddock.
6. **Move to next day:** The date in the system is set by the `current_date` variable. (Note that this is not the real current date on your computer.) Selecting this option from the menu moves `current_date` forward by one day. The current date should be displayed and updated in the main menu. Stock ages must all be updated for the new date. As the grass grows and is eaten by the stock, the pasture dry matter (`'total dm'`) and dry matter per hectare (`'dm/ha'`) values in each paddock must be updated using the `pasture_levels()` function (refer to `farm_data.py`). Familiarise yourself with the `pasture_levels()` parameters, what the function does, and what the function returns. The first 3 arguments are available via `paddocks`. The other 2 should use the appropriate global variables assigned near the top of your main program (do not change those values). Do not alter this function.

Notes:

- The provided `farm_mgmt_your_name.py` Python file contains a menu structure and partially completed functions. These 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 `farm_data.py`. (To view this in a separate window while editing `farm_mgmt_your_name.py`, in VS Code choose File > Duplicate Workspace and then display `farm_data.py`.)
- One function for menu option 1 has already been provided for you: `list_all_stock()`. This lists details of all stock (except for birth date). This is an example of how to produce basic output using `display_formatted_row()` function.
- 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) 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 do not need to 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 `farm_data.py` give an idea of the level of commenting that is expected (excluding the long instructions for `display_formatted_row()`).
- 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 `farm_data.py`.

Submission:

Submit (upload) **only** your main Python .py file for marking: `farm_mgmt_your_name.py` (with your name in the file name). Do not upload the `farm_data.py` file.

- Submit your file via the submission link on the COMP636 **Assessment** page.

Indicative mark allocation

This *indicates* where to spend your time. (Allocation may change if one part is harder than expected, for example.)

40 marks available in total:

Item + ***Requirements***	Approximate marks
Menu enhancement *Current date shows in menu options. Code exits with upper or lower case 'x'.*	**2**
List stock by mob *Rows for animals in same mob grouped, with mob name and appropriate column headings above each group. Mobs in correct order. Appropriate date formatting.*	**4**
List paddock details *Details and headings displayed appropriately, in correct order.*	**2**
Move mobs between paddocks *Sensible interface that ensures correct values are entered. Paddock data updated correctly. Can only move to empty paddock. Can't overwrite an occupied paddock.*	**10**
Add new stock *New animal successfully added to stock and mob; stock numbers updated. Correct ID using next_id(). Birth date entered in appropriate format with valid value. Age calculated correctly. Valid weight range. Continues asking to add animals until exit option entered.*	**12**
Move to next day *Current date increased by 1. Stock ages updated correctly. Pasture growth values correctly updated. Current date updated in main menu (marked under 'Menu enhancement' above.)*	**6**
Code quality *Appropriate commenting. Well-structured code with sensible names. Good practice, no unnecessary duplication of code, etc.*	**4**
TOTAL	**40**

General requirements

- All data inputs validated to prevent incorrect data entry or program crashes
- Labels, headings, etc. in user-friendly format, e.g. "Stock list" rather than "stock_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)

Note : Further amendment

There is no requirement to update the cow weights. You can also ignore stock_growth_rate