Master assignment BaseCamp

Period: 9-13 January 2023

Mountaineering: climbing high!

Introduction

In recent weeks you have been preparing for the climb to the top in Basecamp. In the master assignment, we're going to take this climb to showcase the skills you've acquired over the past weeks.

Based on the techniques you have learned in Python, you will perform an assignment that has the Mountaineering theme. These are expeditions to well-known mountains, where climbers have tried to reach the top.

The application you will write for this will extract information from files and write it to other sources, generate reports and perform calculations to provide answers to questions we propose.

Beyond that, you are free to further expand the application to your own satisfaction in order to do justice to all the knowledge you have acquired. Finally, the master assignment is a showcase for your knowledge and work!

You can expand the application as far as you want, as long as the mandatory parts (which are listed later in this document) remain available. This way you can give it a Command Line Interface or maybe even a Graphical User Interface if you want! Make sure that you first perform the parts specified by us before you start working on extensions.

Topics that will be covered:

- Basic Python (datatypes, functions, loops, if/else, collections)
- Classes
- SQL
- JSON
- CSV
- Unit tests

Database

We have already made an empty sqlite3 database for you (climbersapp.db) and a JSON-file (expeditions.json) with information about climbers and their expeditions. With empty we mean that the tables are already created and only must be filled with the data from the JSON file Filling the database should only be done if the database is empty and when starting your application.

Schema database

The database consists of 4 tables

Table **climbers** has the following fields:
id (integer), first_name (string), last_name (string), nationality (string), date_of_birth (datetime)

Table **mountain** has the following fields:
id (integer), name (string), country (string), rank (integer), height (integer), prominence (integer), range (string)

Table **expeditions** has the following fields:
*id (integer), name (string), mountain_id (integer) <- id of row in table mountains, start_location (string), date (datetime), country (string) *, duration (integer), success (boolean)*

Table **expedition_climbers** is a table to connect climbers and expeditions based on their **id:**
climber_id (integer) <- id of row in table climbers, expedition_id (integer) <- id of row in table expeditions

**: in `expeditions.json` you will find a list of countries (so 1 or more).*
Use the first country as the country to store when extracting data for storing in the database

Classes

You have to make three classes in three separate files (*climber.py, mountain.py, expedition.py*)
Every class should have an __init__() with the attribute fields so they can be passed when initializing and a __repr__() for repr/printing (@dataclass style).

Class **Climber** has the following attributes:
id (int), first_name (str), last_name (str), nationality (str), date_of_birth (datetime, format: %Y-%M-%D)

The class should also have at least these methods:
get_age() (returns the age as integer), get_expeditions()
(returns a list of Expedition's)

Climber
- id : int - first_name : str - last_name : str - nationality : str - date_of_birth : datetime
+ get_age() -> int + get_expeditions() -> list(Expedition)

Class **Mountain** has the following attributes:
id (int), name (str), country (str), rank (int), height (int), prominence (int), range (str)

The class should also have at least these methods:
height_difference() (returns the difference between height and prominence as integer), get_expeditions() (returns a list of Expedition's for this mountain)

Mountain
- id : int - name : str - country : str - rank : int - height : int - prominence : int - range : str
+ height_difference() -> int + get_expeditions() -> list(Expedition)

Class **Expedition** has the following attributes:
id (int), name (str), mountain_id (int), start (str),
date (datetime), country (str), duration (int),
success (boolean)

The class should also have at least these methods:
add_climber(climber: Climber) (adds climber to expedition table expedition_climbers via the id of the passed climber object and the id of this expedition)
get_climbers() (returns a list of Climber's
search in table expedition_climbers for all climber_id's on this expedition, search all climbers with those id's)
convert_date(to_format: string) (returns converted date of this expedition in the provided datetime format)
convert_duration(to_format: string) (returns converted duration in the provided datetime format)
(example: %H:%M will return hours:minutes, options are: %D = days, %H = hours, %M = minutes)

Expedition
- id : int - name : str - mountain_id: int - start: str - date : datetime - country : str - duration : int (minutes) - success : bool
+ add_climber(climber: Climber) + get_climbers() -> list(Climber) + get_mountain() -> Mountain + convert_date(to_format: string) - > string + convert_duration(to_format: string) -> string

JSON example

Below is an example of the JSON (expeditions.json) that is supplied. Based on this JSON, you will synchronize the data with the database when you start your application.


```
[
  {
    "id": 1,
    "name": "Up to the Mount Everest",
    "mountain": {
      "name": "Mount Everest",
      "rank": 1,
      "range": "Mahalangur Himalaya",
      "prominence": 8848,
      "height": 8848,
      "countries": [
          "Nepal",
          "China"
      ]
    },
    "date": "1952-12-31",
    "country": "Czech Republic",
    "start": "Nepal",
    "duration": "18H39",
    "success": true,
    "climbers": [
      {
        "id": 1,
        "first_name": "Marion",
        "last_name": "Lissandre",
        "nationality": "Czech Republic",
        "date_of_birth": "24-08-1928"
      },
      {
        "id": 2,
        "first_name": "Jacquelynn",
        "last_name": "Terzza",
        "nationality": " Czech Republic ",
        "date_of_birth": "08-11-1926"
      }
    ]
  },
  …
]
```

Functionality

A template file is provided with a Reporter class with the following questions.
The template will also have the methods and potential example output specified.

Questions asked in the class Reporter:

1. How many climbers are there? -> int
   # Return: integer of amount
2. What is the highest mountain? -> Mountain
   # Return: object of type Mountain
3. What is the longest and shortest expedition? -> tuple[Expedition, Expedition]
   # Return: tuple(longest, shortest), both objects of type Expedition
4. Which expeditions have the most climbers -> tuple[Expedition, ...]
   # Return: tuple of all expeditions with most climbers (objects of type Expedition each)
5. Which climbers have made the most expeditions -> tuple[Climber, ...]
   # Return: tuple of all climbers with the most expeditions (objects of type Climber each)
6. Which mountains have the most expeditions -> tuple[Mountain, …]
   # Return: tuple of all mountains with the most expeditions (objects of type Mountain each)
7. Which expedition was the first expedition? -> Expedition
   # Return: object of type Expedition
8. Which expedition was the first successful expedition? -> Expedition
   # Return: object of type Expedition
9. Which expedition is the latest? -> Expedition
   # Return: object of type Expedition
10. Which successful expedition is the latest? -> Expedition
    # Return: object of type Expedition
11. Which climbers have climbed mountain Z between period X and Y? -> tuple[Climber, ...]
    # Return if `to_csv = False`: tuple of all climbers (objects of type Climber each)
    # Return if `to_csv = True`: None, but generate a CSV file:
    -`Climbers Mountain Z between X and Y.csv`
    with fields: "id, first_name, last_name, nationality, date_of_birth (format: %Y-%M-%D)""
12. Which mountains are located in country X? ->tuple[Mountain, ...]
    # Return if `to_csv = False`: tuple of all mountains (objects of type Mountain each)
    # Return if `to_csv = True`: None, but generate a CSV file:
    - `Mountains in country X.csv`
    with fields: "id, name, country, rank, height, prominence, range"
13. Which climbers are from country X? -> tuple[Climber, ...]
    # Return if `to_csv = False`: tuple of all climbers (objects of type Climber each)
    # Return if `to_csv = True`: None, but generate a CSV file:
    - `Climbers in country X.csv`
    with fields: "id, first_name, last_name, nationality, date_of_birth (format: %Y-%M-%D)"

Testing

The following unit tests need to be written:

1. test_age_of_climber():
   Test to check if the age of a climber is correct, based on the date_of_birth
2. test_amount_of_expeditions_of_climber():
   Test to check if the amount of expeditions for a specific climber is returned correctly
3. test_height_difference_mountain():
   Test to check the difference in height and prominence of a mountain
4. test_amount_of_expeditions_of_mountain():
   Test to check if the amount of expeditions for a specific mountain is returned correctly
5. test_expedition_date_conversion():
   Test to check if the returned date matches the specified format for that expedition date
6. test_expedition_duration_conversion():
   Test to check if the duration is converted from 1H19 to the specified format
7. test_add_climber_to_expedition():
   Test to check if a climber is added correctly to a specified expedition
8. test_amount_of_climbers_on_expidtion():
   Test to check the amount of climbers on a specified expedition
9. test_mountain_on_expedition():
   Test to validate if the given mountain of a specified expedition is correct

Technical requirements

- Code standard PEP8
- Imports of useful libraries allowed
- Python version 3.10
- Unit tests with Pytest

Filenames to submit in CodeGrade:

- *climber.py*
- *mountain.py*
- *expedition.py*
- *climbersapp.py*
- *climbersreporter.py*
- *test_climbersapp.py*

Plagiarism

The code has to be your code!

All submissions will be tested for plagiarism *(so don't copy code from your fellow students)* and abuse will result in a plagiarism report to the Exam committee. Your work will no longer be checked and will no longer count for the assessment.

Submissions in CodeGrade

The number of hand-ins is limited to 1 time per hour,
so test your code locally first before you upload it to CodeGrade!

Deadlines

Ma-Wo groups will start at Monday morning and have until Thursday 12-01-2023 23:59 to hand in their work. Hand-in after Thursday 12-01-2023 23:59 is blocked and not possible.

Di-Do groups will start at Tuesday morning and have until Friday 13-01-2023 23:59 to hand in their work. Hand-in after Friday 13-01-2023 23:59 is blocked and not possible.

Within CodeGrade there are 2 assignments (1 for the Ma-Wo group & 1 for the Di-Do group) make sure you **hand in your work in the correct assignment!** Also make sure to hand-in some work prior to the deadline, in case handing in doesn't work at the last moment!

Deliverables (summary)

The minimal deliverables you have to hand in are the predefined files: *climbers.py, mountain.py, expedition.py, climbersapp.py, climbersreporter.py, test_climbersapp.py.* These will be tested against predefined tests in CodeGrade. As a student you are free to hand in more work if you extended this assignment to show your knowledge. These extras can also be handed in via CodeGrade.

Good luck!