Master assignment BaseCamp

Period: 8-12 January 2024

Ice skating: A speedy event for success!

Bron afbeelding: https://www.reuters.com/lifestyle/sports/speed-skating-dutch-target-first-beijing-winter-games-medal-womens-3000m-2022-02-04/

Introduction

In recent weeks you have been preparing for a success in Basecamp. In the master assignment, we're going to take a small event 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 Speed Ice Skating theme. These are events to well-known ice tracks, where skaters have tried to win the speed skating event.

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 (iceskatingapp.db) and a JSON-file (events.json) with information about skaters and their events. 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 **skaters** has the following fields:
id (integer), first_name (string), last_name (string), nationality (string), gender (string), date_of_birth (date)

Table **tracks** has the following fields:
id (integer), name (string), city (string), country (string), outdoor (bool), altitude (integer)

Table **events** has the following fields:
*id (integer), name (string), track_id (integer) <- id of row in table tracks, date (date), duration (float) *, laps (integer), winner (string), category (string)*

Table **event_skaters** is a table to connect skaters and events based on their **id:**
skater_id (integer) <- id of row in table skaters, event_id (integer) <- id of row in table events

**: in `events.json` you will find a time like: 29:39.192 (MM:SS.MS) (max 4 decimals accuracy on MS)*
For ease, convert this time to total seconds (float) for the duration (with milliseconds)

Classes

You have to make three classes in three separate files (*skater.py, track.py, event.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 **Skater** has the following attributes:
id (int), first_name (str), last_name (str), nationality (str), generder (str), date_of_birth (date)

The class should also have at least these methods:
get_age(date: date = now()) *(returns the age in years from a specific date/or today)*
get_events() (returns a list of Event's)

Skater
- id : int - first_name : str - last_name : str - nationality : str - gender: str - date_of_birth: date
+ get_age(date: date = now()) -> int + get_events() -> list(Event)

Class **Track** has the following attributes:
id (int), name (str), city (str), country (str), outdoor (bool), altitude (int)

The class should also have at least these methods:
get_events() (returns a list of Event's for this track)

Track
- id : int - name : str - city: str - country : str - outdoor : bool - atltitude : int
+ get_events() -> list(Event)

Class **Event** has the following attributes:
id (int), name (str), track_id (int), date (date), duration (float), laps (int),
winner (str), team (str)

The class should also have at least these methods:
add_skater(skater: Skater) (adds skater to event table event_skaters via the id of the passed skater object and the id of this event)
get_skaters() (returns a list of Skaters
search in table event_skaters for all skater_id's on this event, search all skaters with those id's)
get_track () (returns Track object)
convert_date(to_format: string) (returns converted date of this event in the provided datetime format)
convert_duration(to_format: string) (returns converted duration in the provided datetime format)
(example: %M:%S will return minutes:seconds, options are: %M = minutes, %S = seconds)

Event
- id : int - name : str - track_id: int - date : date - distance: int - duration : float (seconds.milliseconds) - laps : int - winner: str - category: str
+ add_skater(skater: Skater) + get_skaters() -> list(Skater) + get_track() -> Track + convert_date(to_format: string) - > string + convert_duration(to_format: string) -> string

JSON example

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


```
[
  {
    "id": 1,
    "title": "Essent ISU World Cup - 1500m Men Division A",
    "season": 2004,
    "start": "2003-11-08T01:00:00Z",
    "distance": {
      "name": "1,500 Meter",
      "distance": 1500,
      "lapCount": 4
    },
    "category": "M",
    "track": {
      "id": 29,
      "name": "Hamar Olympic Hall",
      "city": "Hamar",
      "country": "NOR",
      "isOutdoor": false,
      "altitude": 123,
      "latitude": "60.793162",
      "longitude": "11.101411"
    },
    "results": [
      {
        "startNumber": 26,
        "startLane": "O",
        "armband": "red",
        "rank": 1,
        "time": "1:47.370",
        "timeBehind": "0.000",
        "skater": {
          "id": 537,
          "firstName": "Erben",
          "lastName": "Wennemars",
          "country": "NED",
          "gender": "M",
          "dateOfBirth": "1975-11-01"
        }
      },
      ...
    ]
  },
  ...
]
```

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 skaters are there? -> int
   # Return: integer of amount
2. What is the highest located track? -> Track
   # Return: object of type Track
3. What is the longest and shortest event? -> tuple[Event, Event]
   # Return: tuple(longest, shortest), both objects of type Event
4. Which event has the most laps for the given track_id -> tuple[Event, ...]
   # Return: tuple of all events with most laps for that track (objects of type Event each)
5. Which skaters have made the most events -> tuple[Skater, ...]
   # Return: tuple of all skaters with the most Events (objects of type Skater each)
6. Which skaters have made the most successful events -> tuple[Skater, ...]
   # Return: tuple of all skaters won won the most Events (objects of type Skater each)
7. Which tracks has the most Events -> tuple[Track, …]
   # Return: tuple of all tracks with the most Events (objects of type Track each)
8. Which Track had the first event? -> Track
   # Return: object of type Track
9. Which Track held the first outdoor event? -> Track
   # Return: object of type Track
10. Which Track was had the last event? -> Track
    # Return: object of type Track
11. Which Track held the last outdoor event? -> Track
    # Return: object of type Track
12. Which skaters have raced track Z between period X and Y? -> tuple[Skater, ...]
    # Return if `to_csv = False`: tuple of all skaters (objects of type Skater each, no duplicates)
    # Return if `to_csv = True`: None, but generate a CSV file:
    -`Skaters on Track Z between X and Y.csv`
    with fields: "id, first_name, last_name, nationality, gender, date_of_birth (format: %Y-%M-%D)""
13. Which tracks are located in country X? ->tuple[Track, ...]
    # Return if `to_csv = False`: tuple of all tracks (objects of type Track each, no duplicates)
    # Return if `to_csv = True`: None, but generate a CSV file:
    - `Tracks in country X.csv`
    with fields: "id, name, city, country, outdoor, altitude"
14. Which skaters have nationality X? -> tuple[Skater, ...]
    # Return if `to_csv = False`: tuple of all skaters (objects of type Skater each, no duplicates)
    # Return if `to_csv = True`: None, but generate a CSV file:
    - `Skaters with nationality X.csv`
    with fields: "id, first_name, last_name, nationality, gender, date_of_birth (format: %Y-%M-%D)"

Testing

The following unit tests need to be written:

1. test_amount_of_events_of_skater():
   Test to check if the amount of Events for a specific Skater is returned correctly
2. test_age_of_skater():
   Test to check the correct age in years is returned of a specific skater
3. test_amount_of_events_of_track():
   Test to check if the amount of Events for a specific track is returned correctly
4. test_event_date_conversion():
   Test to check if the returned date matches the specified format for that Event date
5. test_event_duration_conversion():
   Test to check if the duration is converted to the specified format
6. test_amount_of_skaters_on_event():
   Test to check the amount of skaters on a specified Event
7. test_track_on_event():
   Test to validate if the given track of a specified Event is correct

Technical requirements

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

Filenames to submit in CodeGrade:

- *skater.py*
- *track.py*
- *event.py*
- *iceskatingapp.py*
- *iceskatingreport.py*
- *test_iceskatingapp.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 5 minutes,
so test your code locally first before you upload it to CodeGrade!

Deadlines

Start at Monday morning and have until Friday 12-01-2024 23:59 to hand in their work. Hand-in after Friday 12-01-2024 23:59 is blocked and not possible.

Deliverables (summary)

The minimal deliverables you have to hand in are the predefined files: *skater.py, track.py, event.py, iceskatingapp.py, iceskatingreporter.py, test_iceskatingapp.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!