Master assignment BaseCamp

Period: 13-17 January 2025

Fishing: A step into the world of fishing!

Source Image: generated with MS Copilot

Table of Contents

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.

Using the techniques you have learned in Python, you will complete an assignment based on the theme of fishing. The assignment involves analyzing catches of various types of fish by contestants from around the world.

The application you develop will extract information from files, store it in other sources, generate reports, and perform calculations to answer the questions we present.

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 like the challenge. 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

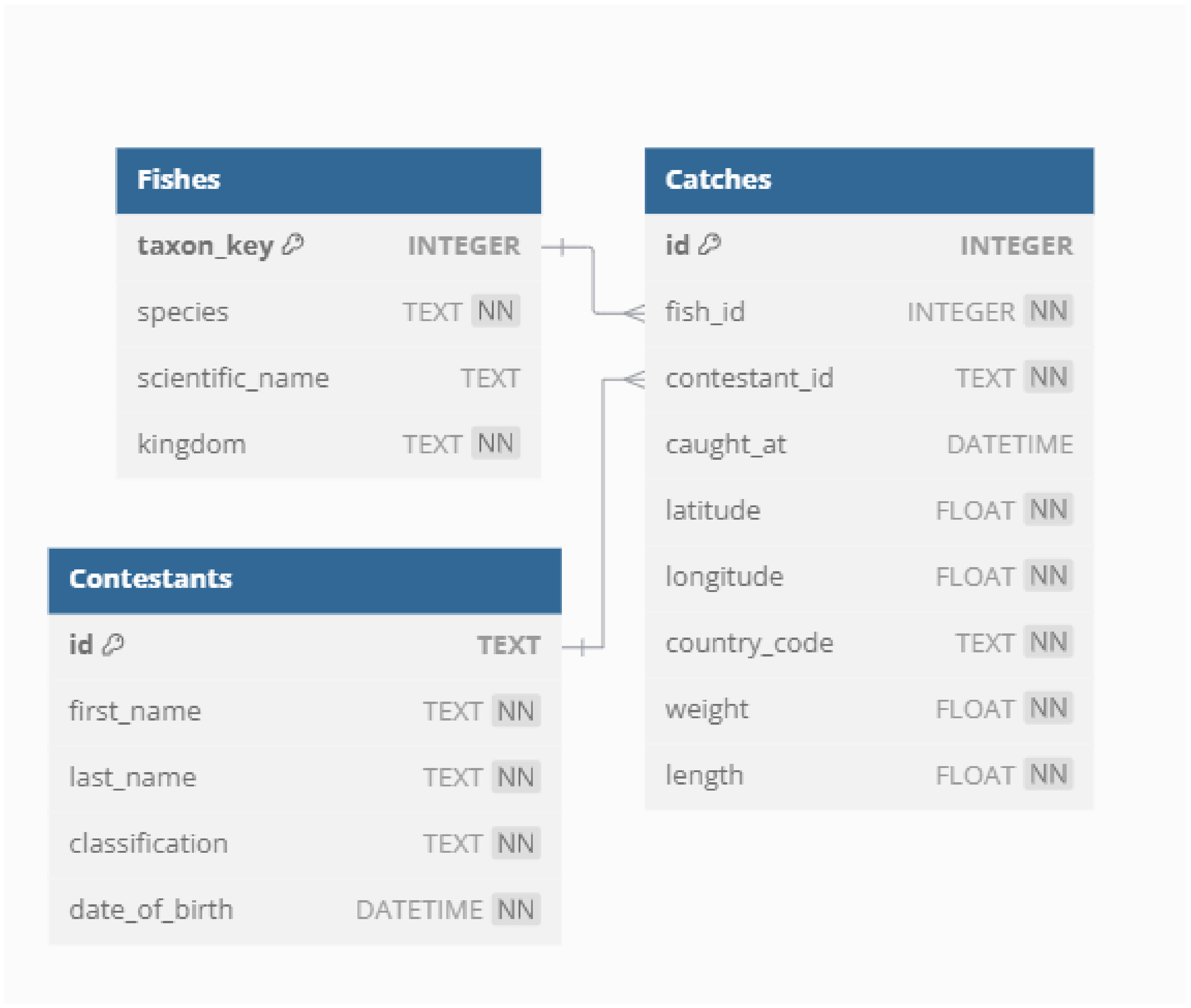
Database

We have already made an empty sqlite3 database for you (catches.db) and a JSON-file (catches.json) with information about Catches, Fishes and Contestants. With empty we mean that the tables are already created and only must be filled with the data from the JSON file.

The database should only be populated if it is empty, and this check should be performed when the application starts. When populating the database, make sure to use one transaction (one commit). Otherwise, your script will take too long to run on CodeGrade and you will not be able to see if the other tests pass.

Tip: Make a copy of the empty database before populating it. This way, if something goes wrong, you can easily restore the empty database and start fresh.

Schema database



The database consists of 3 tables

Table **Fishes** has the following fields:
taxon_key (integer), species (string), scientific_name (string), kingdom (string)

Table **Catches** has the following fields:
id (integer), fish_id (integer), contestant_id (string), caught_at (datetime), latitude (float), longitude (float), country_code (string), weight (float), length (float)

Weight is in grams, length is in centimeters.

Table **Contestants** has the following fields:
id (string), first_name (string), last_name (string), classification (string), date_of_birth (datetime)

Classes

You have to make three classes in three separate files (*catch.py*, *contestant.py*, *fish.py*):

- *Every class should have an* ***__init__()*** *method with the attribute fields as parameters, so they can be passed when initializing the class*
- *Every class should have a* ***__repr__()*** *method for repr/printing (@dataclass style). This method is already given (see template) and should not be changed.*

Class **Catch**:

- **Attributes:**
  id (int) fish (int), contestant (str), caught_at (datetime), latitude (float), longitude (float), country_code (str), weight (float), length (float)
  Weight is in grams, length is in centimeters.
-
- ***Mandatory Methods:***
  - ***get_contestant() -> Contestant***
    Returns the contestant who caught the fish in this catch.
  - ***get_fish() -> Fish***
    Returns the fish caught in this catch.
  - ***get_weight_in_local_units() -> str***
    Returns a string that displays the mass of the catch in the local unit of mass based on the country code. Use the table below to determine the appropriate unit and conversion formula for the country
    Use like: "{value_unit_a} {name_unit_a}, {value_unit_b} {name_unit_b}"
    example"1000.00 Gram, 1.00 Kilogram"

C Code	***Country***	***Common Units of Mass***	***Conversion Formula from Grams***
US	***United States***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
LR	***Liberia***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
MM	***Myanmar***	***Viss (v), Pound (lb)***	***Viss: Grams ÷ 1600*** ***Pounds: Grams ÷ 453.592***
CA	***Canada***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
GB	***United Kingdom***	***Stone (st), Pound (lb)***	***Stones: Grams ÷ 6350.29*** ***Pounds: Grams ÷ 453.592***

AU	***Australia***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
BS	***Bahamas***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
FJ	***Fiji***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
JM	***Jamaica***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
PG	***Papua New Guinea***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
TO	***Tonga***	***Pound (lb), Ounce (oz)***	***Pounds: Grams ÷ 453.592*** ***Ounces: Grams ÷ 28.3495***
IN	***India***	***Pound (lb), Kilogram (kg)***	***Pounds: Grams ÷ 453.592*** ***Kilograms: Grams ÷ 1000***
KH	***Cambodia***	***Pound (lb), Kilogram (kg)***	***Pounds: Grams ÷ 453.592*** ***Kilograms: Grams ÷ 1000***
TZ	***Tanzania***	***Pound (lb), Kilogram (kg)***	***Pounds: Grams ÷ 453.592*** ***Kilograms: Grams ÷ 1000***
PH	***Philippines***	***Pound (lb), Kilogram (kg)***	***Pounds: Grams ÷ 453.592*** ***Kilograms: Grams ÷ 1000***
LK	***Sri Lanka***	***Pound (lb), Kilogram (kg)***	***Pounds: Grams ÷ 453.592*** ***Kilograms: Grams ÷ 1000***

Notes on Units of Mass

- *If the country code is **not listed** in the table, default to **Grams** and **Kilograms**.*
- *Always use the provided conversion formulas for accuracy.*

- ***get_day_part() -> str***
  return what part of the day the catch was made. Use "Night", "Morning", "Afternoon" and "Evening" (0:00-5:59, 6:00-11:59, 12:00-17:59, 18:00-23:59)
- ***get_season() -> str***
  return what season the catch is done. Use "Spring", "Summer", "Autumn" and "Winter". You can use the meteorological seasons of the northern hemisphere in your computations.

- ***get_weight_category() -> str***
  This method returns the weight category of the fish. The categories are defined as:
  - *“**light**”: The fish weighs more than 2% less than the expected weight.*
  - *“**average**”: The fish’s weight is within ±2% of the expected weight.*
  - *“**heavy**”: The fish weighs more than 2% above the expected weight.*

Formula for Expected Weight To determine the **expected weight** of the fish, use the following formula:

$$weight = a * length^b$$

Length in cm

a = 0.0123

b = 3.1

Catch
- id: int - fish: int - contestant: str - caught_at: datetime - latitude: float - longitude: float - country_code: str - weight: float - length: float
+ get_contestant() -> Contestant + get_fish() -> Fish + get_weight_in_local_units() -> str + get_day_part() -> str + get_season() -> str + get_weight_category() -> str

Of course you can add any extra methods needed in your class, but the ones mentioned above are mandatory.

Class **Contestant**:

- **Attributes**:
  id (str), first_name (str), last_name (str), classification (str), date_of_birth (datetime)
- ***Mandatory Methods:***
  - ***get_age(at_date: date = date.today())***
    returns a integer of the contestant on the specified date. When no date is provided use the date of today
  - ***get_catches()***
    returns a tuple of all catches from the contestant

Contestant
- id: str - first_name: str - last_name: str - classification: str, - date_of_birth: datetime
+ get_age (self, at_date: date = date.today()) -> int + get_catches() -> tuple[Catch, …]

Of course you can add any extra methods needed in your class, but the one mentioned above is mandatory.

Class **Fish**:

- **Attributes**:
  taxon_key (str), species (str), scientific_name (str), kingdom (str)
- ***Mandatory Methods:***
  - ***get_catches()***
    returns a tuple of all catches that contains this kind of fish.

Fish
- taxon_key: int - species: str - scientific_name: str - kingdom: str
+ get_catches() -> tuple[Catch, …]

Of course you can add any extra methods needed in your class, but the one mentioned above is mandatory.

JSON example

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

```
[
    {
        "id": 1,
        "datetime": "2023-03-04 23:59:18",
        "coordinate": "-6.416667, 20.783333",
        "weight": 0.3,
        "length": 2.74,
        "fish": {
            "taxon_key": "5202660",
            "species": "Clarias dumerilii",
            "kingdom": "Animalia",
            "scientific_name": "Clarias dumerilii Steindachner, 1866",
            "occurrence_status": "PRESENT",
            "family": "Clariidae"
        },
        "country_code": "CD",
        "locality": "Tshikapa",
        "candidate": {
            "id": "71237630-0755-4857-b394-35285bfb3619",
            "first_name": "Stef",
            "last_name": "de Werd",
            "classification": "Pro",
            "date_of_birth": "1974-07-19",
            "length": 1.89
        }
    },
    { ...}
    ...
]
```

The coordinates in the JSON are structured as “lat, long”. Weight is in grams, length of catch in is in cm, length of candidate is in m.

Functionality

A template file is provided with a Reporter class with the following questions. In case of a collection return your answer sorted by the primary identifier.
The template will also have the methods and potential example output specified.

Questions asked in the class Reporter:

1. How many different fishes are there in the database? -> int
   # Return: integer of amount
2. What is the catch with the highest weight (assume only one catch is the longest)? -> Catch
   # Return: object of type Catch
3. What is the longest and shortest catch? -> tuple[Catch, Catch]
   # Return: tuple(longest, shortest), both objects of type Catch
4. What is the heaviest and lightest Catch? -> tuple[Catch, Catch]
   # Return: tuple(heaviest, lightest), both objects of type Catch
5. Which contestants have the most catches? -> tuple[Contestant, ...]
   # Return: tuple of all contestants with the most catches (objects of type Contestant)
6. Which fishes have the most catches? (order by id) -> tuple[Fish, ...]
   # Return: tuple of all fishes with the most catches (objects of type Fish)
7. Which contestant had the first catch? -> tuple[Contestant, …]
   # Return: tuple of all contestant with the first catch (objects of type Contestant)
8. Which contestant had the first catch of a specific fish type? -> tuple[Contestant, …]
   # Return: tuple of all contestant with the first catch by fish of type X (objects of type Contestant)
9. Which contestant had the last catch? -> tuple[Contestant, …]
   # Return: tuple of all contestant with the last catch (objects of type Contestant)
10. Which contestant had the last catch of a specific fish type? -> tuple[Contestant, …]
    # Return: tuple of all contestant with the last catch by fish of type X (objects of type Contestant)
11. Which contestant has fished between period X and Y? -> tuple[Contestant, ...]
    The function determines which contestants have fished between a given period of X an Y.
    The behavior of the function depends on the value of the parameter to_csv:
    If to_csv = False:
    The function returns a tuple containing all matching contestants. Each element in the tuple is a unique object of type Contestant.
    If to_csv = True:
    The function does not return anything (None). Instead, it generates a CSV file named:
    Contestant fishing between X and Y.csv
    The CSV file contains the following fields for each contestant:
    - "id"
    - "first_name"
    - "last_name"
    - "date_of_birth"
    - *"classification"*

12. *W*hich fishes are fished in country X? ->tuple[Fish, ...]
    This function determines which fishes are caught in Country X
    The behavior of the function depends on the value of the parameter to_csv:
    If to_csv = False:
    The function returns a tuple containing all matching Fishes. Each element in the tuple is a unique object of type Fish*.*
    If to_csv = True:
    The function does not return anything (None). Instead, it generates a CSV file named:
    Fishes in country X.csv
    The CSV file contains the following fields for each fish:
    - *"taxon_key"*
    - *"species"*
    - *"kingdom"*
    - *"scientific_name"*
13. Which contestants fished in country X? -> tuple[Contestant, ...]
    This function determines which contestants have been fishing in country X
    If to_csv = False:
    The function returns a tuple containing all matching Contestants. Each element in the tuple is a unique object of type Contestant*.*
    If to_csv = True:
    The function does not return anything (None). Instead, it generates a CSV file named:
    Contestants fished in country X.csv
    The CSV file contains the following fields for each contestant:
    - *"id"*
    - *"first_name"*
    - *"last_name"*
    - *"date_of_birth"*
    - *"classification"*

Testing

The test cases on CodeGrade use a different dataset than the JSON file that is provided to you. Thus, when you write unit tests, make sure not to use any data you find in the JSON file, as those will fail when you upload your code to CodeGrade.

Make sure the following methods are covered with tests:

In **test_catch.py** test the methods from **catch.py**

1. get_weight_in_local_units():
   Test the conversion to multiple different local mass units
2. get_day_part():
   Test to check if the right day parts will be returned
3. get_season():
   Test to check if the right seasons will be returned
4. get_weight_category():
   Test to check if the fish is in the right category

In **test_contestant.py** test the methods of **contestant.py**

1. get_age():
   Test if the age of the contestant is right calculated

Technical requirements

- Code standard PEP8
- Imports of useful libraries is allowed
- Python version 3.11 or higher
- Unit tests with Pytest

Filenames to submit in CodeGrade:

- *catch.py*
- *contestant.py*
- *fish.py*
- *fishingapp.py*
- *fishingreporter.py*
- *test_catch.py*
- *test_contestant.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 17-01-2025 23:59 to hand in their work. Hand-in after Friday 17-01-2024 23:59 is blocked and not possible.

Deliverables (summary)

The minimal deliverables you are required to hand in are the predefined files: *catch.py, contestant.py, fish.py, fishingapp.py, fishingreporter.py, test_catch.py, test_contestant.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!