Master assignment BaseCamp

Period: 10-14 June 2024

Shipments: A step into the world of cargo!

Bron afbeelding: https://slicesofbluesky.com/port-charts/

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 Shipment theme. These shipments of all types of materials (containers, oil, sand, etc.) from one port to another over a sea route conducted by different types of vessels.

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.

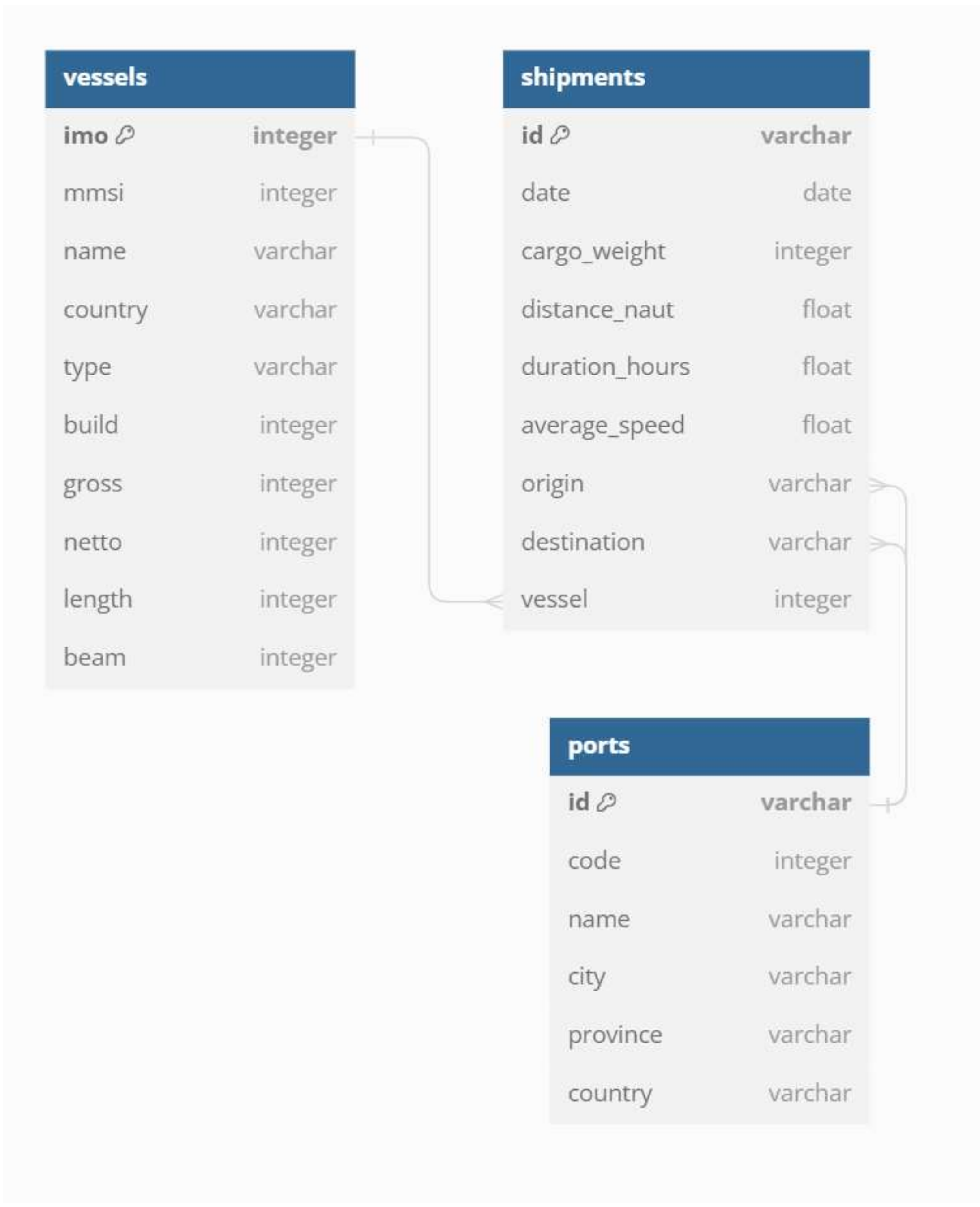
Topics that will be covered:

- 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 (shipments.db) and a JSON-file (shipments.json) with information about vessels, ports and shipments. 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



vessels	
imo	**integer**
mmsi	integer
name	varchar
country	varchar
type	varchar
build	integer
gross	integer
netto	integer
length	integer
beam	integer

shipments	
id	**varchar**
date	date
cargo_weight	integer
distance_naut	float
duration_hours	float
average_speed	float
origin	varchar
destination	varchar
vessel	integer

ports	
id	**varchar**
code	integer
name	varchar
city	varchar
province	varchar
country	varchar

The database consists of 3 tables

Table **vessels** has the following fields:
imo (integer), mmsi (integer), name (string), country (string), type (string),
build [year] (integer), gross (integer), netto (integer), length (integer), beam* (integer)*

Table **ports** has the following fields:
id (string), code (integer), name (string),
city (string), province (string), country (string),

Table **shipments** has the following fields:
id (string), date (date), cargo_weight (integer), distance_naut (float), duration_hours (float),
average_speed [in knots] (float),
origin (string), destination (string),
vessel (integer)

: in `shipments.json` you will find a* ***size* *like: 155 / 54, which is the length / beam, you need to convert this before inserting it in the database.*

Classes

You have to make three classes in three separate files (*port.py, vessel.py, shipment.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 **Port** has the following attributes:
id (str), code (int), name (str), city (str), province (str), country (str)

The class should also have at least these methods:
get_shipments() (returns a tuple of Shipments for this port (can be in destination or origin))

Port
- id : str - code: int - name : str - city : str - province: str - country: str
+ get_shipments() -> tuple(Shipments)

Class **Vessel** has the following attributes:
imo (int), mmsi (int), name (str), country (str), type (str), build (int), gross (int), netto (int), length (int), beam (int)

The class should also have at least these methods:

get_* shipments() (returns a tuple of Shipments for this vessel)*
get_fuel_consumption(distance: float) (returns a number based on the following calculation)

- *efficiency * (gross_weight / netto_weight) * distance*
- *value should be rounded up to maximum of 5 decimals*
- *efficiency can be calculated by ship type, use the following list with values:*

Aggregates Carrier = 0.4	*Landing Craft = 0.4*
Bulk Carrier = 0.35	*Nuclear Fuel Carrier = 0.35*
Bulk/Oil Carrier = 0.35	*Palletised Cargo Ship = 0.4*
Cement Carrier = 0.4	*Passenger/Container Ship = 0.3*
Container Ship = 0.3	*Ro-Ro Cargo Ship = 0.4*
Deck Cargo Ship = 0.4	*Self Discharging Bulk Carrier = 0.35*
General Cargo Ship = 0.4	*Vehicles Carrier = 0.35*
Heavy Load Carrier = 0.4	*Wood Chips Carrier = 0.4*

Vessel
- imo : int - mmsi: int - name : str - country: str - type : str - build : int - gross: int - netto: int - length: int - beam : int
+ get_fuel_consumption(distance: float) -> float + get_shipments() -> list(Shipment)

Class **Shipment** has the following attributes:
id (str), date (date), cargo_weight (int), distance_naut (float), duration_hours (float), average_speed (float), origin (str) [Port.id], destination (str) [Port.id], vessel (int) [Vessel.imo]

The class should also have at least these methods:
get_ports() (returns a dict of Ports as {"origin": Port(…), "destination": Port(…)})
get_vessel () (returns Vessel object)
*calculate_fuel_costs(price_per_liter: float, vessel: Vessel) (returns total price [rounded down to 3 decimals behind comma] of the shipment based on the duration in hours * fuel_consumption of the vessel * price per liter)*
convert_speed(to_format: string) (returns converted speed of this shipment in the provided format [rounded down to 6 decimals behind comma], options are: Knts, Mph, Kmph)
[#! should raise a ValueError if an unsupported format was given]
convert_distance(to_format: string) (returns converted distance of this shipment in the provided datetime format [rounded down to 6 decimals behind comma], options are: NM, M, KM, MI, YD)
[#! should raise a ValueError if an unsupported format was given]
convert_duration(to_format: string) (returns converted duration in the provided datetime format)
(example: %D:%H will return days:hours, options are: %D = days, %H = hours, %M = minutes)

Shipment
- id : str - date: date - cargo_weight: int - distance_naut: float - duration_hours: float - average_speed: float - origin: str - destination: str - vessel: int
+ get_ports() -> dict(str: Port, str: Port) + get_vessel() -> Vessel + calculate_fuel_costs(price_per_liter: float, vessel: Vessel) -> float + convert_speed(to_format: string) -> float + convert_distance(to_format: string) - > float + convert_duration(to_format: string) -> string

JSON example

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

```
[
  {
    "date": "01-01-2023",
    "tracking_number": "78067E7F-D833-4312-A805-C1355F51F065",
    "cargo_weight": 15649,
    "distance_naut": 5879.249,
    "duration_hours": 864.595,
    "average_speed": 6.8,
    "origin": {
      "id": "MYTPP",
      "code": 55750,
      "name": "Tanjung Pelepas",
      "alias": null,
      "city": "Tanjung Pelepas",
      "province": "Johor",
      "country": "Malaysia"
    },
    "destination": {
      "id": "TRGEM",
      "code": 48947,
      "name": "Gemlik",
      "alias": null,
      "city": "Gemlik",
      "province": "Bursa",
      "country": "Turkey"
    },
    "vessel": {
      "imo": 9913547,
      "mmsi": 477736400,
      "country": "Hong Kong",
      "name": "TIGER LONGKOU",
      "type": "Deck Cargo Ship",
      "build": 2022,
      "gross": 23040,
      "netto": 26200,
      "size": "192 / 37"
    }
  },
  ...
]
```

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 vessels are there? -> int
   # Return: integer of amount
2. What is the longest shipment distance? -> Shipment
   # Return: object of type Shipment
3. What is the longest and shortest vessel? -> tuple[Vessel, Vessel]
   # Return: tuple(longest, shortest), both objects of type Vessel
4. What is the widest and smallest vessel? -> tuple[Vessel, Vessel]
   # Return: tuple(widest, smallest), both objects of type Vessel
5. Which vessels have the most shipments? -> tuple[Vessel, ...]
   # Return: tuple of all vessels with the most Shipments (objects of type Vessel each)
6. Which ports have the most shipments? (order by id) -> tuple[Port, ...]
   # Return: tuple of all ports with the most Shipments (objects of type Port each)
7. Which ports (origin) had the first shipment? -> tuple[Port, …]
   # Return: tuple of all ports with the first Shipment (objects of type Port each)
8. Which ports (origin) had the first shipment of a specific vessel type? -> tuple[Port, …]
   # Return: tuple of all ports with the first Shipment by vessel of type X (objects of type Port each)
9. Which ports (origin) had the latest shipment? -> tuple[Port, …]
   # Return: tuple of all ports with the latest Shipment (objects of type Port each)
10. Which ports (origin) had the latest shipment of a specific vessel type? -> tuple[Port, …]
    # Return: tuple of all ports with the latest Shipment by vessel of type X (objects of type Port each)
11. Which vessels have docked port Z between period X and Y? -> tuple[Vessel, ...]
    # Return if `to_csv = False`: tuple of all vessels (objects of type Vessel each, no duplicates)
    # Return if `to_csv = True`: None, but generate a CSV file:
    -`Vessels docking Port Z between X and Y.csv`
    with fields: "imo, mmsi, name, country, type, build, gross, netto, length, beam"
12. Which ports are located in country X? ->tuple[Port, ...]
    # Return if `to_csv = False`: tuple of all ports (objects of type Port each, no duplicates)
    # Return if `to_csv = True`: None, but generate a CSV file:
    - `Ports in country X.csv`
    with fields: "id, code, name, city, province, country"
13. Which vessels are from country X? -> tuple[Vessel, ...]
    # Return if `to_csv = False`: tuple of all vessels (objects of type Vessel each, no duplicates)
    # Return if `to_csv = True`: None, but generate a CSV file:
    - `Vessels from country X.csv`
    with fields: "imo, mmsi, name, country, type, build, gross, netto, length, beam"

Testing

The following unit tests need to be written:

1. test_convert_duration():
   Test to check if duration is converted correctly based on the given arguments
2. test_convert_distance():
   Test to check if distance is converted correctly based on the given arguments
3. test_convert_speed():
   Test to check if speed is converted correctly based on the given arguments
4. test_get_fuel_consupmtion():
   Test to check if the fuel consumption is calculated correctly based on the distance
5. test_calculate_fuel_costs():
   Test to check if the fuel costs are calculated correctly based on the price per liter
6. test_get_ports ():
   Test to check if the returned ports are correct
7. test_get_shipments ():
   Test if the returned shipments contain the required shipment(s)

Technical requirements

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

Filenames to submit in CodeGrade:

- *port.py*
- *vessel.py*
- *shipment.py*
- *shipmentapp.py*
- *shipmentreporter.py*
- *test_shipmentapp.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 14-06-2024 23:59 to hand in their work. Hand-in after Friday 14-06-2024 23:59 is blocked and not possible.

Deliverables (summary)

The minimal deliverables you have to hand in are the predefined files: *port.py, vessel.py, shipment.py, shipmentapp.py, shipmentreporter.py, test_shipmentapp.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!