Assignment 2

Key Information

Overview

Objective

- Work collaboratively with your team (nominally, the teams have size 2).
- Develop a navigation system with a text-based interface (see tasks in next pages).
- Practice the basic programming concepts from weeks 1-9.
- Download the following scaffold to populate with your solutions:

A2_scaffold.zip

Please read the individual docstrings for further details.

Submission Requirements

- You will work on each task as different functions in your Ed workspace.
- One group member must make the final submission on Moodle for the whole group.
- Your final submission must be one zip file with the necessary and sufficient files, as shown below.

```
A2_<group_name>.zip
└─> city.py
└─> csv_parsing.py
└─> country.py
└─> itinerary.py
└─> map_plotting.py
└─> onboard_navigation.py
└─> path_finding.py
└─> vehicles.py
└─> requirements.txt
└─> (any other file you may have created for Task 8)
```

Make sure that all functions are in the file they need to be in, with appropriate function names and arguments.

It is possible to complete the assignment without importing more modules than the ones which are either imported or recommended. However, you may import other modules. Make sure they are an improvement to the program, though! Furthermore, please note that Ed has a restricted set of modules, which you cannot add to. Finally, be aware that your TAs may ask you about what these modules do and how you use them during the interview.

To ensure your code passes test cases, you will be required to include a requirements.txt file in your final submission. In order to generate requirements.txt, open a terminal in your project directory and execute the below command:

```
pip freeze > requirements.txt
```

A requirements.txt file is a plain text document that lists the names and version numbers of Python packages required for a specific project or application. This file allows developers and users to recreate the exact package environment needed for the code to run correctly, ensuring consistent behaviour across different systems. By including a requirements.txt file in a project, it simplifies the process of installing dependencies, making it easier to share, collaborate, and deploy.

Documentation

In this assignment, some documentation is provided as part of the scaffold code. However, any function, method, class or file you add must have docstrings.

In-line comments

You are expected to use in-line comments throughout your program to make sure it is clear to the reader. The amount of comments necessary depends on how clear the code is by itself. Therefore, a good habit is to write clear code, so that the amount of in-line comments required is reduced. This will also reduce the number of bugs created.

Comment your code as you write it rather than after reaching a final solution, as it's a lot easier to remember why certain decisions were made and the thought process behind them when the code is fresh in your mind.

Ensure your variable names are meaningful and concise so that your code is understandable at first read. When a reader is going through your program, it should read like a story that is self-explanatory, using in-line comments to explain the logic behind blocks of code and any additional information that is relevant to the functionality of the code.

Avoid repeating yourself by commenting on lines of code that are fairly self-explanatory, and instead reserve your comments to explain higher level logic, such as explaining the overall purpose behind a small block of code.

Recommended Approach

Working as a team:

- You are expected to work with all members of your team (typically this is a single other teammate).
- You are expected to attend your applied classes to meet with your team members and the TA who supervises your team.
- During the applied class, your TA will check on your progress and on how well the team is working together. This is the opportunity to raise issues. But don't throw your teammate under the bus. There is always friction in a team. Work together to smooth it out.
- You will need to meet regularly with your team and collaborate on the assignment.

Setting up your workspace:

- At the beginning of the assignment, one member needs to:
    - Create an Ed workspace for your team and share it with the other members,
    - Upload the code scaffold into workspace and share workspace with the team.
- You are expected to work on the program together using the shared workspace in Ed.

Note that if all of you are already familiar with tools like git, you may use them instead. But we won't provide technical support.

Testing your code:

- The assignment is broken up into individual coding tasks.
- You can code and test using the Ed lessons.
- These lessons are **not** to submit your work, only to test your code against some basic automated checks.

Scaffold code:

- All tasks but Task 8 starts with scaffold code.
- Make sure you do not modify the variable, function, method and class names; these will be needed to pass automated tests. Your TA will rely on these existing to understand your program and mark efficiently.

Academic Integrity

- Your code will be checked against all other groups with a similarity checker.
- Anything you are able to google can be easily found by the teaching team and compared against your work.
- In this assessment, you must not use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task. This includes ChatGPT.

How can I avoid academic integrity issues?

- Never copy code from anywhere. If you learn something useful online, rewrite it from scratch. It's also the best way to make sure you have understood it.
- If a fellow student asks you for a solution to a question, try to help them build their solution. Do **not** give them yours.
- Giving your solution is just as much of an Academic Integrity breach as receiving it.
- If you feel like you physically cannot submit the assignment on time, please submit an extension request, and seek help.
- For extensions, see Moodle AU or Moodle MA.
- For help, see Moodle AU or Moodle MA.

How will I be assessed?

Your assignment mark will come from:

1. your *team submission,*
2. your *contribution factor.*

There will be an interview scheduled with a TA (Teaching Assistant) after the submission deadline, which will help determine criteria 1 and 2.

1. Team submission (common)

This mark is the same for all team members and comes from a review of your submission by a TA, following a rubric.

Note that automated tests do not contribute to the mark. They are here to help you check the correctness of your programs. But a program that passes all tests can still be incorrect. Furthermore, test cases do not evaluate the readability of your code, which our TAs will assess.

2. Contribution factor (individual)

(This is the revised plan following the changes to the A1 interview marking scheme)

The contribution factor is individual, and based on your interview only.

After your individual interview with your tutor, you will be assigned one of five colours:

- **Black (0.1)**
  - If you do not attend the interview. Note that you can apply for special consideration (see how to apply for a short extension on Moodle).
- **Red (0.15)**
  - Insufficient evidence
- **Orange (0.5)**
  - Describe (without coding) how to change the program you have submitted to meet different requirements, for one task (chosen by the student).
  - The task chosen by the student must be among Task 5-8, unless they have not been attempted in the submission.
- **Blue (0.8)**
  - Using the "orange" question the student answered, implement changes within the program you have submitted to meet different requirements (on the same task and question).
- **Green (1.0)**

    - Describe (without coding) how to change the program you have submitted to meet different requirements, for one task (chosen by the TA).

Templates of questions include:

- Modify `xyz()` such that instead of type `A`, it takes as argument type `B`. (Plus details.)
- Create a method `xyz()` in class `A` that takes `foo` as argument, computes something related to `self` and `foo`, and returns `bar`.
- Starting from a copy of your implementation of class `A`, create class `B` that does something instead of another thing.

What happens during the individual interview?

- The individual part of the interview lasts 20 minutes.
- The interview is open book. In particular, you may use a search engine or a coding website (e.g. stackoverflow, geeksforgeeks). You may not use ChatGPT or other Generative AI tools.
- You can refer to any part of the program you have submitted (not just the task you are working on).
- You may ask the TA to explain the question or part thereof.

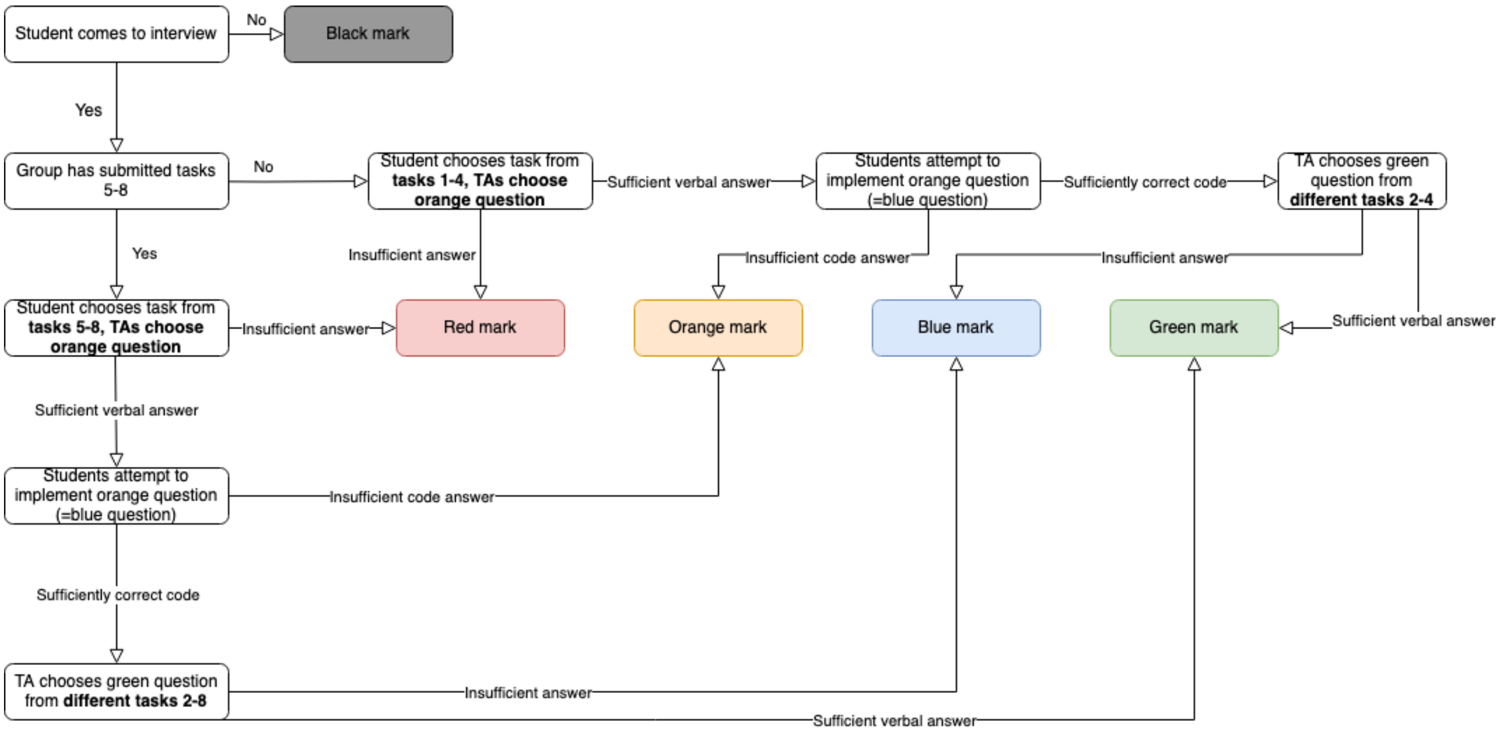
This is the procedure that your TA will follow during the individual part of the interview:

1. Your TA will make sure that the interview is being recorded.
2. Both TA and student turn on their camera. If this is not possible, the interview is rescheduled, in person if need be.
3. Your TA will check your student ID via camera.
4. Share the full screen (not the application) via zoom.
5. Pick a task to get quizzed on from among Tasks 5-8. If you have not attempted any of these questions, then and only then you can pick a different task.
6. Your TA will give you a **first question** to answer on task you chose.
    1. You will first be asked to give a **verbal** answer (spoken or written, no coding) based on the program you have submitted. You may write notes to provide an answer.
    2. If you succeed, you will be asked to **implement** these changes in a text editor (e.g. ed workspace).
7. If you do not answer the question within the interview time, then the interview is over. You will not be asked a second question.
8. If there is time left, your TA will give you a **second question** on a different task chosen by them.
    1. You will only be asked to give a verbal answer (no coding) based on the program you have submitted.
    2. You may write notes to help with your answer.
    3. You may provide the answer by directly modifying the program instead of describing the

modifications.

9. The interview ends when you run out of time, or if your TA tells you that you have answered the second question satisfactorily.

You are not allowed to communicate to other students the questions that were asked to you. This would be a breach of academic integrity.


Student comes to interview
No
Black mark
Yes
Group has submitted tasks 5-8
No
Student chooses task from **tasks 1-4, TAs choose orange question**
Sufficient verbal answer
Students attempt to implement orange question (=blue question)
Sufficiently correct code
TA chooses green question from **different tasks 2-4**
Yes
Insufficient answer
Insufficient code answer
Insufficient answer
Student chooses task from **tasks 5-8, TAs choose orange question**
Insufficient answer
Red mark
Orange mark
Blue mark
Green mark
Sufficient verbal answer
Sufficient verbal answer
Students attempt to implement orange question (=blue question)
Insufficient code answer
Sufficiently correct code
TA chooses green question from **different tasks 2-8**
Insufficient answer
Sufficient verbal answer

Effective team size and team size factor

Some teams may not have exactly 2 members. Furthermore, some teammates might be inactive. So the *effective size* of your team might not be 2. Your TA will be the one making this determination. If the effective team size is:

- 2 or 3, then the team size factor is 100%.
- 1, then the team size factor is 120%.

Your mark will be multiplied by this factor. Inactive students won't receive this factor.

If for some reason you end up in a team of 1, your TA will try to merge teams in order to make teams of size 2.

Note that, if you decide to work alone, even though you have an active teammate in your team, you will not receive the contribution factor.

Late penalties

Up to 7 days

- 10% lost per calendar day (or part thereof).

More than 7 days

- Zero (0) marks with no feedback given.

See on Moodle AU or Moodle MA for how to apply for special consideration.

Overall mark (individual)

Your overall mark for the assignment will be computed according to the program below.

```
max_mark = 100

team_submission_mark = float(input("Team submission mark [0, {}] = ".format(max_mark)))
contribution_factor = float(input("Contribution factor {0.1, 0.15, 0.5, 0.8, 1}= "))
effective_team_size = int(input("Effective team size {0, 1, 2, 3} = "))
late_days = int(input("Days late {0, 1, ...} = "))

#note that we are doing no data validation

possible_team_size_factors = [1.00, 1.20, 1.00, 1.00]
if contribution_factor >= 0.8:
    team_size_factor = possible_team_size_factors[effective_team_size]
else:
    #Inactive students (not green or blue) do not get
    #the effective team size bonus
    team_size_factor = 1

mark = min(max_mark, \
min(max_mark, team_submission_mark * team_size_factor) * contribution_factor)

if late_days <= 7:
    late_penalty = 10 * late_days
else:
    late_penalty = max_mark

final_mark = max(0, mark - late_penalty)

print("The final mark is", final_mark)
```

Project description

The client

Pierre's side business, *Papa Pierre's Pâtisseries*, delivers French yumminess to almost a thousand major locations around the world thanks to its technologically-advanced fleet of vehicles:

- The `CrappyCrepeCar` is a flying car that can travel between any two cities in the world, but moves pretty slowly.

- The `DiplomacyDonutDinghy` is a small boat which is licensed to travel on diplomatic hyperlanes. So it moves extra fast between capital cities. It can also travel between any two cities in the same country. However, it can only move from one country to another via their capitals.

- The `TeleportingTarteTrolley` is a trolley bus that can teleport between any two cities if they are close enough, regardless of countries. Because teleportation technology is still in its infancy, it takes time to program and execute a blink between two cities.

All images have been generated by DALL·E. Clearly it's not yet very good at spelling.

The program

Your job is to create a program that will be used in these vehicles to help them travel between cities, by finding a shortest path between cities and creating a map of the trip.

As an example of what this could look like, we plotted below the trip Paris -> Kuala Lumpur -> Melbourne.

Task 1 - City

Task

- Create a class `City` by defining all the methods that currently contain a `#TODO`.
- Also complete functions `get_city_by_id` and `get_cities_by_name`.
- Use the documentation and type hints of each method/function to determine how to implement them.
- You must use the module `geopy` to approximate the distance between two instances of `City` using the *great circle* method. It is your responsibility to research this module.

Ensure that your implementation of `City` uses the attributes `self.name`, `self.coordinates`, `self.city_type`, `self.population` and `self.city_id`, as currently created in the scaffold.

Class `City` has two class variables, `id_to_cities` and `name_to_cities`, that contain all instances of that class created so far. You need to ensure that these class variables are gradually populated in `__init__`. We give an example here.

Input/output example

Main code

```
if __name__ == "__main__":
    create_example_countries_and_cities()
    test_example_countries_and_cities()
```

Output

```
Melbourne (1036533631)
Melbourne's name is Melbourne
Melbourne's population is 4529500
The distance between Melbourne and Sydney is 714 km
```

We provide the function `create_example_cities()` and a `main` to help you with your own tests. You can modify them, but be aware that `create_example_cities()` is used in the `main` of other tasks.

Task 2 - Country

Task

- Create a class `Country` by defining all the methods that currently contain a `#TODO`.
- Also complete functions `add_city_to_country` and `find_country_of_city`.
- Use the documentation and type hints of each method/function to determine how to implement them.
- You must use the module `tabulate` to output the table of cities in `print_cities`. It is your responsibility to research this module.

Ensure that your implementation of `Country` uses the attributes `self.name`, `self.iso3`, as currently created in the scaffold. Also create an attribute `self.cities` of type `list`, and ensure that it is populated with the cities added to the country.

Class `Country` has one class variable, `name_to_countries`, that contains all instances of that class created so far. You need to ensure that this class variable is gradually populated in `__init__`. We give an example here.

In `print_cities`, you may:
- Use python's built-in `sort`. Research it if you do not know how to use it.
- Print the header of the table with `print` by itself, and the table itself using `tabulate`.

Input/output example

Main code

```
if __name__ == "__main__":
    create_example_countries()
    test_example_countries()
```

Output

```
Cities of Australia
-----  ---------  ------------------------  ---------  ----------  ----------
Order  Name       Coordinates               City type  Population  City ID
0      Sydney     (-33.865, 151.2094)    admin      4840600     1036074917
1      Melbourne  (-37.8136, 144.9631)   admin      4529500     1036533631
2      Canberra   (-35.2931, 149.1269)   primary    381488      1036142029
-----  ---------  ------------------------  ---------  ----------  ----------
```

We provide two functions `create_example_countries()` and `test_example_countries()`, as well as a `main`, to help you with your own tests. You can modify them, but be aware that `create_example_countries()` is used in the `main` of other tasks.

Task 3 - Itinerary

Task

- Create a class `Itinerary` by defining all the methods that currently contain a `#TODO`.
- Use the documentation and type hints of each method/function to determine how to implement them.

⚠ Ensure that your implementation of `Itinerary` uses the attribute `self.cities`, as currently created in the scaffold.

Input/output example

Main code

```
if __name__ == "__main__":
    create_example_cities()
    test_itin = Itinerary([get_cities_by_name("Melbourne")[0],
                           get_cities_by_name("Kuala Lumpur")[0]])
    print(test_itin)

    #we try adding a city
    test_itin.append_city(get_cities_by_name("Baoding")[0])
    print(test_itin)

    #we try inserting a city
    test_itin.min_distance_insert_city(get_cities_by_name("Sydney")[0])
    print(test_itin)

    #we try inserting another city
    test_itin.min_distance_insert_city(get_cities_by_name("Canberra")[0])
    print(test_itin)
```

Output

```
Melbourne -> Kuala Lumpur (6368 km)
Melbourne -> Kuala Lumpur -> Baoding (10579 km)
Sydney -> Melbourne -> Kuala Lumpur -> Baoding (11293 km)
Sydney -> Canberra -> Melbourne -> Kuala Lumpur -> Baoding (11294 km)
```

End of Week 7 Checkpoint

We recommend you complete at least Tasks 1, 2 and 3 by the end of week 7.

Your TA will check the progress of your team during your applied class.

Task 4: Map Plotting

Task

- Write a function `plot_itinerary` that plots an `Itinerary` on a map and outputs it as a file.
- You must use the module `basemap` from `matplotlib`. It is your responsibility to research this module.
- You may customise the look of the map (e.g. different projections, colours, etc), as long as the trip is clearly visible.

There are no automated tests for this task.

If you plot the map of the entire globe, you can ignore the constraints on size and padding given in the docstring of `plot_itinerary`.

If you want to become more familiar with map projections, we recommend this Wikipedia article and this video:

Input/output example

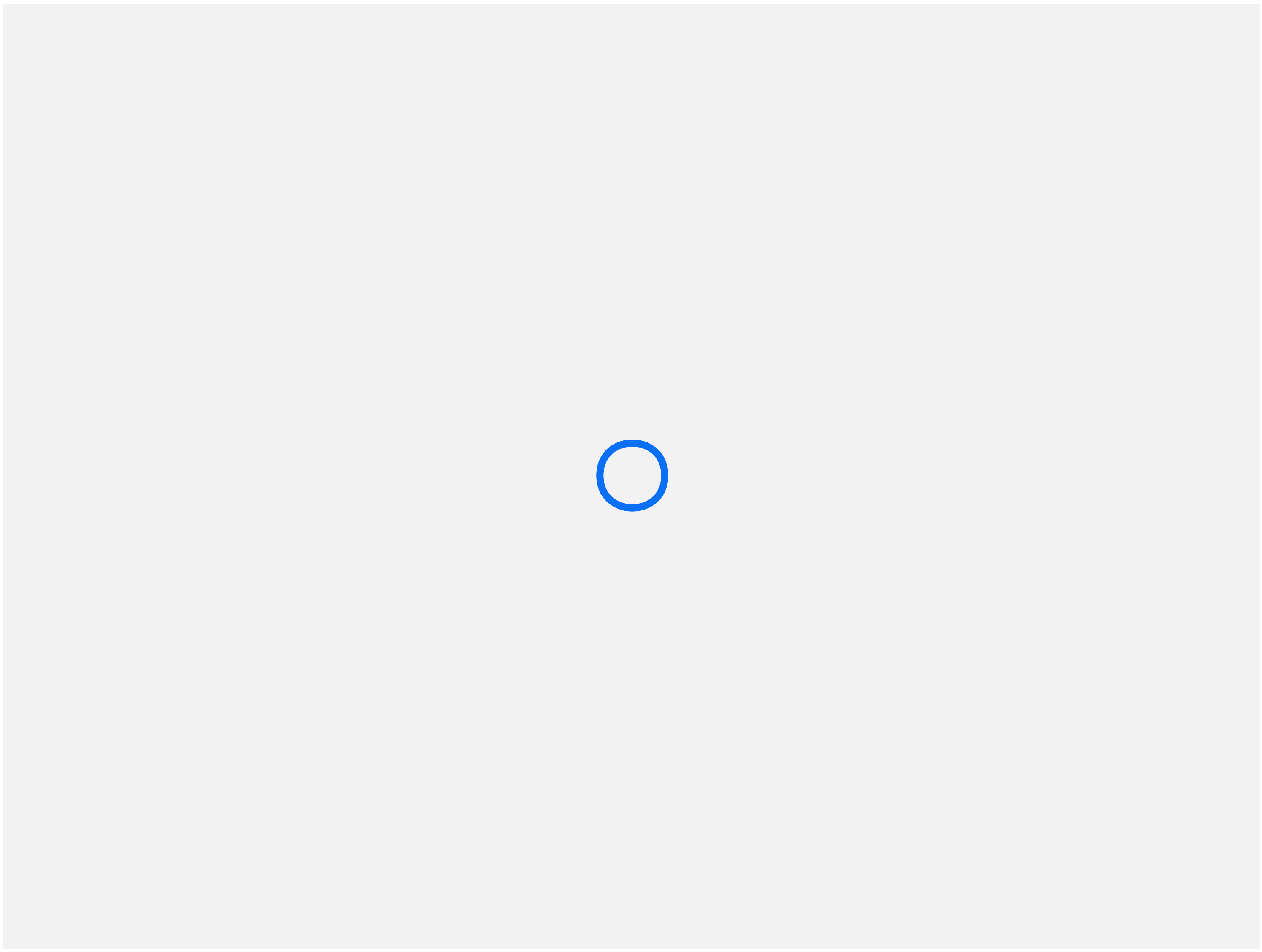
Main code

```
if __name__ == "__main__":
    # create some cities
    city_list = list()

    city_list.append(City("Melbourne", (-37.8136, 144.9631), "primary", 4529500, 1036533631))
    city_list.append(City("Sydney", (-33.8688, 151.2093), "primary", 4840600, 1036074917))
    city_list.append(City("Brisbane", (-27.4698, 153.0251), "primary", 2314000, 1036192929))
    city_list.append(City("Perth", (-31.9505, 115.8605), "1992000", 2039200, 1036178956))

    # plot itinerary
    plot_itinerary(Itinerary(city_list))
```

Output



Note that ed workspaces and lessons may need to be refreshed (F5) to show updated pictures when your code generates them.

Task 5: Vehicle

Task

- Create a class `Vehicle` with an abstract method `compute_travel_time` and a (non-abstract) method `compute_itinerary_time`, which you need to implement in the class `Vehicle`.
- Create three child classes: `CrappyCrepeCar`, `DiplomacyDonutDinghy`, `TeleportingTarteTrolley`, each of which moves between cities in a unique way (see documentation).

⚠ You may add imports.

Input/output example

Main code

```
if __name__ == "__main__":
    #we create some example cities
    create_example_countries()

    from_cities = set()
    for city_id in [1036533631, 1036142029, 1458988644]:
        from_cities.add(get_city_by_id(city_id))

    #we create some vehicles
    vehicles = create_example_vehicles()

    to_cities = set(from_cities)
    for from_city in from_cities:
        to_cities -= {from_city}
        for to_city in to_cities:
            print(f"{from_city} to {to_city}:")
            for vehicle in vehicles:
                print(f"\t{vehicle.compute_travel_time(from_city, to_city)} hours with {vehicle}.")
```

Output

```
Canberra (1036142029) to Kuala Lumpur (1458988644):
        33 hours with CrappyCrepeCar (200 km/h).
        14 hours with DiplomacyDonutDinghy (100 km/h | 500 km/h).
        inf hours with TeleportingTarteTrolley (3 h | 2000 km).
Canberra (1036142029) to Melbourne (1036533631):
        3 hours with CrappyCrepeCar (200 km/h).
        5 hours with DiplomacyDonutDinghy (100 km/h | 500 km/h).
        3 hours with TeleportingTarteTrolley (3 h | 2000 km).
```

```
Kuala Lumpur (1458988644) to Melbourne (1036533631):
        32 hours with CrappyCrepeCar (200 km/h).
        inf hours with DiplomacyDonutDinghy (100 km/h | 500 km/h).
        inf hours with TeleportingTarteTrolley (3 h | 2000 km).
```

End of Week 8 Checkpoint

We recommend you complete at least Tasks 4 and 5 by the end of week 8.

Your TA will check the progress of your team during your applied class.

Task 6: CSV Parsing

Task

- Write a function `create_cities_countries_from_csv` that parses a csv file and creates instances of the class `City` and `Country`.
- You must use the module `csv` to read the data. It is your responsibility to research this module.
- Use the data file provided (worldcities_truncated.csv). It is a subset of the data available at https://simplemaps.com/data/world-cities.

⚠ You may add imports.

CSV format

- Use the columns `"city_ascii", "lat", "lng", "country", "iso3", "capital", "population", "id"` to read the data necessary for the creation of `City` and `Country` instances. You can assume these columns will always exist. (Note that the column `"capital"` corresponds to City's `city_type`).
- You can assume that the first row will always contain the column names.
- You **cannot** assume that the number of rows and their order will always be the same. The two CSV files we provide have different columns. Your program should work for both (and more).

⚠ If the population field is not an integer (e.g. an empty string ''), treat the population as 0.

Input/output example 1

```
if __name__ == "__main__":
    create_cities_countries_from_csv("worldcities_truncated_aus.csv")
    for country in Country.name_to_countries.values():
        country.print_cities()
```

Input

The main code uses `worldcities_truncated_aus.csv`.

Output

```
Cities of Australia
-----  ---------  --------------------  ---------  ----------  ----------
Order  Name       Coordinates           City type  Population  City ID
0      Sydney     (-33.865, 151.2094)   admin      4840600     1036074917
```

```
1      Melbourne  (-37.8136, 144.9631)  admin      4529500     1036533631
2      Brisbane   (-27.4678, 153.0281)  admin      2360241     1036192929
3      Perth      (-31.9522, 115.8589)  admin      2039200     1036178956
4      Adelaide   (-34.9275, 138.6)     admin      1295714     1036538171
5      Canberra   (-35.2931, 149.1269)  primary    381488      1036142029
-----  ---------  --------------------  ---------  ----------  ----------
```

The output is also provided in the file below.

csv_parsing_aus_output.txt

Input/output example 2

Main code

```
if __name__ == "__main__":
    create_cities_countries_from_csv("worldcities_truncated.csv")
    for country in Country.name_to_countries.values():
        country.print_cities()
```

Input

The main code uses `worldcities_truncated.csv`.

Output

The countries are listed in order of appearance in the CSV file.

```
Cities of Japan
-----  ----------  -------------------  ---------  ----------  ----------
Order  Name        Coordinates          City type  Population  City ID
0      Tokyo       (35.6839, 139.7744)  primary    39105000    1392685764
1      Osaka       (34.752, 135.4582)   admin      15490000    1392419823
2      Nagoya      (35.1167, 136.9333)  admin      9522000     1392407472
3      Yokohama    (35.4333, 139.6333)  admin      3757630     1392118339
4      Fukuoka     (33.6, 130.4167)     admin      2280000     1392576294
5      Sapporo     (43.0621, 141.3544)  admin      1961690     1392000195
6      Kawasaki    (35.5167, 139.7)                1539522     1392003356
7      Kobe        (34.6913, 135.183)   admin      1513193     1392978082
8      Kyoto       (35.0117, 135.7683)  admin      1464890     1392622735
9      Saitama     (35.8617, 139.6453)  admin      1320571     1392017719
10     Hiroshima   (34.4, 132.45)       admin      1198021     1392373695
11     Sendai      (38.2683, 140.8694)  admin      1058070     1392457903
12     Chiba       (35.6073, 140.1064)  admin      981738      1392107144
13     Setagaya    (35.6464, 139.6533)             940509      1392792380
14     Kitakyushu  (33.8833, 130.8833)             935084      1392003140
15     Sakai       (34.5667, 135.4833)             823523      1392003291
16     Niigata     (37.9161, 139.0364)  admin      790646      1392913753
17     Hamamatsu   (34.7167, 137.7333)             788211      1392174500
-----  ----------  -------------------  ---------  ----------  ----------
Cities of Indonesia
-----  --------------  -------------------  ---------  ----------  ----------
```

```
Order   Name              Coordinates           City type   Population   City ID
0       Jakarta           (-6.2146, 106.8451)   primary     35362000     1360771077
1       Surabaya          (-7.2458, 112.7378)   admin       2885555      1360484663
2       Bandung           (-6.95, 107.5667)     admin       2875673      1360313023
(OUTPUT TRUNCATED FOR SPACE REASON. SEE FILE BELOW)
```

The complete output is provided in the file below.

csv_parsing_output.txt

▶ Expand

Task 7: Path Finding

Task

- Write a function `find_shortest_path` that returns a shortest path as an `Itinerary` from one `City` to another, for a given `Vehicle`.
- Here, a shortest path is one that has the **smallest travel time** for the given `Vehicle`. This is not about distance. Note that for some vehicles and some cities, there may not be a path at all.
- You must use the module `networkx` to compute shortest paths. It is your responsibility to research this module.

If you are not familiar with graphs or the shortest path problem, we have provided some background information.

Please note that the automated tests for this task will take quite a long time to execute. The timeout is set to 5 minutes, however if your tests are still timing out please review your solution.

Input/output example

Main code

```
if __name__ == "__main__":
    create_cities_countries_from_csv("worldcities_truncated.csv")
    vehicles = create_example_vehicles()

    from_cities = set()
    for city_id in [1036533631, 1036142029, 1458988644]:
        from_cities.add(get_city_by_id(city_id))

    #we create some vehicles
    vehicles = create_example_vehicles()

    to_cities = set(from_cities)
    for from_city in from_cities:
        to_cities -= {from_city}
        for to_city in to_cities:
            print(f"{from_city} to {to_city}:")
            for test_vehicle in vehicles:
                shortest_path = find_shortest_path(test_vehicle, from_city, to_city)
                print(f"\t{test_vehicle.compute_itinerary_time(shortest_path)} hours with {test_veh
                      f" with path {shortest_path}.")
```

Output

On your end you may get these output lines in a different order. That's fine.

```
Melbourne (1036533631) to Canberra (1036142029):
        3 hours with CrappyCrepeCar (200 km/h) with path Melbourne -> Canberra (466 km).
        5 hours with DiplomacyDonutDinghy (100 km/h | 500 km/h) with path Melbourne -> Canberra (46
        3 hours with TeleportingTarteTrolley (3 h | 2000 km) with path Melbourne -> Canberra (466 k
Melbourne (1036533631) to Kuala Lumpur (1458988644):
        32 hours with CrappyCrepeCar (200 km/h) with path Melbourne -> Kuala Lumpur (6368 km).
        19 hours with DiplomacyDonutDinghy (100 km/h | 500 km/h) with path Melbourne -> Canberra ->
        27 hours with TeleportingTarteTrolley (3 h | 2000 km) with path Melbourne -> Sydney -> Noum
Canberra (1036142029) to Kuala Lumpur (1458988644):
        33 hours with CrappyCrepeCar (200 km/h) with path Canberra -> Kuala Lumpur (6527 km).
        14 hours with DiplomacyDonutDinghy (100 km/h | 500 km/h) with path Canberra -> Kuala Lumpur
        27 hours with TeleportingTarteTrolley (3 h | 2000 km) with path Canberra -> Sydney -> Noume
```

It will take some time and memory to find a shortest path. If your approach is inefficient in terms of run time or memory usage, ed might kill your program before it terminates.

Task 8: Onboard Navigation

Task

Design a program that allows a user to:

- Select a Vehicle from a list of existing vehicles (at least the three examples given in `create_example_vehicles`).
- Select an origin and destination city from among all cities in `worldcities_truncated.csv` (any city must be selectable, including cities that have homonyms).
- Find a shortest path between the two given cities, if there exists one.
- If there is a path, create a map of the path, and exit. Otherwise, just exit.

The program needs to validate user input, and recover from invalid inputs. For example, if the user is asked to choose from Vehicle 1 to 3, and the user inputs `0`, `123.45`, `kek`, etc, then the program needs to ask the user for an input again. This holds for all user inputs. In order to do so, you **must** use Exceptions.

We do not provide a scaffold or tests for this task. You're completely in charge of the design of the program! You may create additional files. Just don't forget to include them in your submission!

End of Week 9 Checkpoint

Ideally, you will have completed Tasks 1-7 (and even perhaps Task 8) by the end of Week 9.

Your TA will check the progress of your team during your applied class.

Example of class with a register of its instances.

Here's an example that should help you understand how to populate `Country.countries` and `City.cities` in Task 1.

```
class MyClass():

    #a class variable that will list all instances
    instances = []

    def __init__(self, name: str):
        #initialise this instance normally
        self.name = name

        #add this instance to the class instances
        MyClass.instances.append(self)

#we create some instances
MyClass("Never")
MyClass("Gonna")
MyClass("Give")
MyClass("You")
MyClass("Up")
#note how we have no local variable to access these instances

#we can access these instances via the class
for instance in MyClass.instances:
    print(instance.name)
```

Background information on graphs and shortest paths

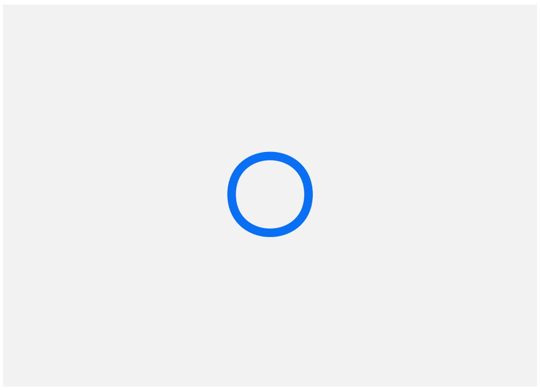
We ask you to compute a shortest path between two cities. The typical way to solve this problems consists in using a *graph*.

Graph

A graph is an abstract representation made of *nodes* and *edges.*

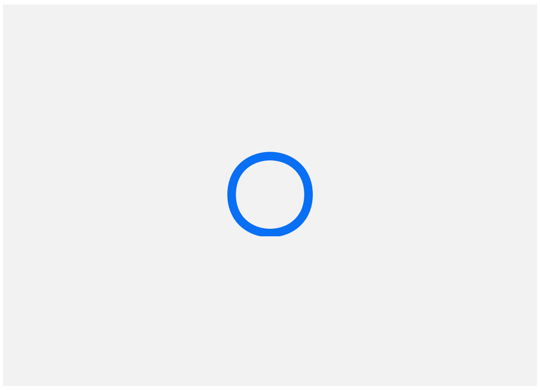
For us, a node would represent a city (for example, Melbourne):

An edge is a link between two nodes (for example, between the Melbourne and Kuala Lumpur nodes):



For us, a link represents the fact that one can travel between M and KL.

In order to represent the fact that there is a travel duration, we add *weights* on edges:



Here, this would mean that it takes 5 hours to travel from M to KL.

Shortest path example

In the example above, the shortest path between M and KL would take 5 hours. Let's see a more complex example.

In this example, we have added the cities Beijing, Paris, and Tokyo, as well as travel times.

Note that not all possible edges exist in this graph. For example, one cannot travel directly from Melbourne to Paris.

Can you find a shortest path from Melbourne to Paris in this graph?

▶ Expand

In this assignment, we do not ask you to write an algorithm to find a shortest path in a graph. Instead, we recommend you use algorithms provided in existing modules.

Marking Rubric

1 Automatic Zoom

Changelog (11 changes)

09/05/2023 - change 1

Added flowchart to "How will I be assessed" slide

17/04/2023 - change 1

Added

to Task 7.

17/04/2023 - change 2

Added tests for task 6.

17/04/2023 - change 3

Added tests for task 7.

Amendment to task description:

18/04/2023 - change 4

Added instructions for generating a requirements.txt file in your final submission - Overview slide.

20/04/2023 - change 5

In Task 2, changed

to

20/04/2023 - change 6

Fixed the automated tests of Task 7 for the `TeleportingTarteTrolley`. See this post.

26/04/2023 - change 7

Updated the marking scheme of the interview.

02/05/2023 - change 8 (minor)

Updated task descriptions:

- Task 2 example output erroneously included quotation marks for coordinates in Country.print_cities() - removed.
- Task 5 example output was from a previous version of the main method, updated.

02/05/2023 - change 9

Added the following info to Task 4 (Map Plotting):

> If you plot the map of the entire globe, you can ignore the constraints on size and padding given in the docstring of `plot_itinerary`.

02/05/2023 - change 10

Added section *What happens during the individual interview?* in the slide How will I be assessed?

02/05/2023 - change 11

Added the following info to Task 6 (CSV Parsing):

⚠ You may add imports.

(see this post).