Assignment 4

Klement/cgx702

Question 1

Question 1.1

He choose to represent the Colour data type as a list of countries. This means that all the countries with the same Colour are in a list.

Question 1.2

I would instead make the Colour data type a tuple, containing a string and a list. The string would represent the colours, like green or red. And the list would represent all the countries that have that colour.
This would mean that the new extendColouring would have the signature:
`val extendColouring:  NeighbourRelation -> Country -> tuple list -> tuple list`

Question 1.3

I would extend the class with a function to check, that the neighbor Relation list, has the same amount of unique countries, as the final colouring List.
The idea is to convert both lists into a, with each unique country list and check that both of the list are the same length.
I would also add a function that checks that none of the countries in colouring are neighbours.
We simply loop through the colouring list and the neighbour relation list and check if they are neighbours.

Question 1.4

I would test with a few different inputs, using input Partitioning meaning we split these into a few different valid inputs, with no overlap:

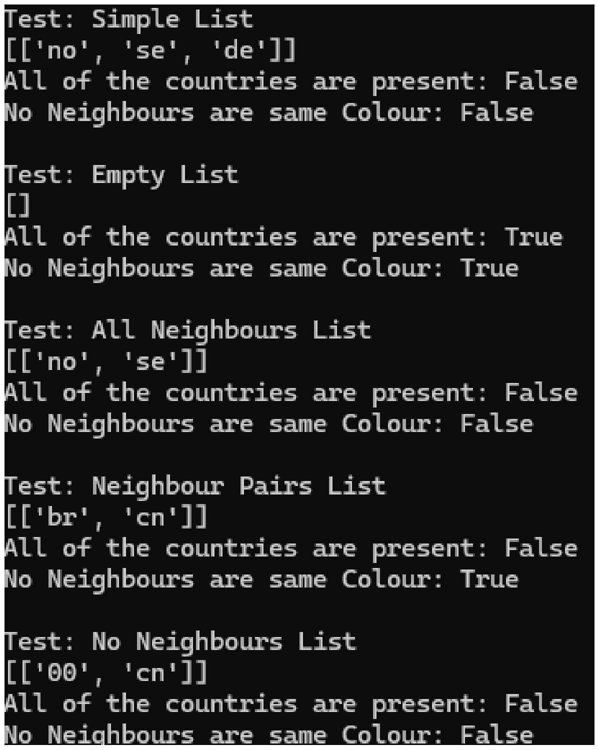
- A normal input List, like the simply we are supplied, testing that the function works as expected.
- A empty input List.
- A input List where they are all neighbours.
- A Input List where the countries are only neighbours with one other country
- A input List where none are neighbours.

Question 1.5

Running these tests i encountered three different problems:

- Not every country is given a colour, example 'da' in simpleTest and in the all neighbours test.
- Sometimes neighbours have the same colour, example 'no' and 'se' in simpleTest.
- Countries, which only appeared in the second slot of the a neighbour relation where not given a node and therefore not given a colour, found with pair `('00','br'` and in the neighbourPairTest, with only half of the neighbourRelation countries getting colours.

These where the initial output of the tests.



```
Test: Simple List
[['no', 'se', 'de']]
All of the countries are present: False
No Neighbours are same Colour: False

Test: Empty List
[]
All of the countries are present: True
No Neighbours are same Colour: True

Test: All Neighbours List
[['no', 'se']]
All of the countries are present: False
No Neighbours are same Colour: False

Test: Neighbour Pairs List
[['br', 'cn']]
All of the countries are present: False
No Neighbours are same Colour: True

Test: No Neighbours List
[['00', 'cn']]
All of the countries are present: False
No Neighbours are same Colour: False
```

To fix these issues i made changes the places in the colouring file. The nodes list is supposed to contain every country that is the neighbour relation input. Before it was only the first country of each tuple that was added to the nodes list. This means that some countries would be forgotten if they weren't in the first slot of one of the tuples. In the initialization of the object I added a loop which adds every country once to the nodes list.
Next is the issues with neighbour having the same colour this is because in the areNieghbours function it used to only check if the tuple `(c1,c2)` is present in the nr instead of also checking the `(c2,c1)`. This means the order in which the countries stood in the neighbour relation list changed the outcome of the colouring.
Next was the problem with not every country being represented this was a simple fix. In the first line of the extendColouring function i changed from return and list with an empty list if the colouring list is empty to returning a list with a list of the country instead.
I made one additional change in the code, this issue i didn't spot because of a test, but instead i spotted this because the program kept crashing after i fixed the other issues. In line 29 in the extendColouring function, i changed the recursive call from extendColouring(country, colouring[:1]) to extendColouring(country,colouring[1:]). This means we call the extendColouring function with the tail of the colouring List instead of the head.

Question 2

Question 2.1

To design the functions to answer the question we first need to handle the flight data list. Here i use two function. The generateFlightsList which takes a file path and outputs a list of dictionaries.
These dictionaries have use the first line of the flight data as keys, these are all the different data associated with each flight, this makes looking up the specific data for each flight very easy.
The generateFlightsList starts by calling a helper function called `read_to_list`, which takes as input the file path of the csv file and outputs a list, where each element of the list is a line in the csv file. This code is taken directly from a lesson.

The generateFlightsList function then takes each element in the list outputted from the read to list function and strips where there is a `','`. This means we now have a list with a list. We take the first element of that list and set that to be our keys. Now we go through every line in the after the first one and make a dictionary with a keys corresponding to the data. This means we have a list where every line in the csv file, except the first one, is a dictionary with all of the data keyed to the same data.
To answer the first question of the amount of flights landing at copenhagen airport each month, i defined the function findFlightsLandingAtCopenhagen. This takes a flightList as input and outputs a dictionary, where the keys correspond to each month and one is the total amount of flights landing at copenhagen all year.
The function start by looping through every flight in the flightList and checking if the `'name_ades'` is copenhagen. We then add one to the total, we then split the date and check if second element of the list, which is the month in the format of this file, is already a key in our dictionary and if it is add one to the counter of that month and if it isn't the simply create the key with a value of 1.
This gives us a total of 29292 flights that landed at Copenhagen airport in 2022 and my computer found the result in 0.073 seconds.
To Answer the question of which cities flights departed to Ibiza i made the function findFlightsToIbiza. it takes flightList as an input and outputs a list with every place where flights departed from. Here we also loop through every flight in flightList we check if the flight name ades is Ibiza and then save the depature name if it is not already in the list.
This function gives us a total of 33 different airports where flights to ibiza departed from and my computer found the result in 0.135 seconds.
To answer the question of how many flights flew in spring 2022 between March 20th and june 21st. I defined the function findFlightsInSpring, that takes the flightList as input and outputs a integer which is the amount of flights in spring.
Here we loop to every flight in the flightList we split the date into a list seperated by the - and converts the month and day to integers, since this data is only for 2022, i choose not to care about year. We then check if the month is equal to 4 or 5. Or if the month is equal to 3 and the day is after 20. Or if the month is equal to 6 and the day is before the 21st. If it is we then add 1 to the total amount of flights.
This gives a total of 144303 flights in spring and my computer found the result in 0.133 seconds.

Question 2.2

To test these functions i used the assert function in python. Here we check for a lot of different input using input partitioning to see if we get the right amount of flights. We always test a list where none of the flights fits any of the conditions, which should give the same result as running with an empty List.
Then for Copenhagen I test with a list with two flights in two different months, to see if they are keyed to the right month and also that the total value also increments correctly.
Then for Ibiza I test with a two unique depature and duplicate departure places. This is to see that every new departure place is added and none are added twice.
Then for flights in Spring I test with a List with a flight in the middle of spring and also test the edges to see that it runs correctly for the weirder dates, meaning march 20th and june 21st.

Question 3

Question 3.1

Question: What are data types good for? Answer: Data types are good so that the computer know what operation it can perform on a certain piece of data. There is a big difference between `'707'`, a string, and `707`, an integer and which operations the computer can perform on the data.

Question 3.2

Question: What is object-oriented programming good for? Answer: Using object-oriented programming groups data and functions that are related. This means that repetitive tasks can be simplified and any additional processes that we might need to add can be expanded onto the original class.

Question 3.3

Question: What is a limitation of lists as a data type?
Answer: Finding specific data is slow in larger lists, since we can't check specific data unless we have the index of it. Meaning we have to check one by one.