White Star Line Project

Intention of the software

I intended to create a model that was efficient, in terms of the time the code took to run, as well as the number of lines of code and readability of the code. The code should at a minimum, pull in and read two files and display the data on a GUI. The code should also be able to pull out specific values from the files and display them into an output.

Issues during development

Within most of the code I only ran into minor errors that were easy to overcome. For example, not indenting code properly, or missing out ‘:’ at the end of a line of code. These, although they stopped the code from running, were easy to notice and quick to fix after running the code and seeing an error message. There were however, two larger issues regarding the ice height that took more attention to overcome.

Firstly, I struggled to return all the values for ‘height’ where there was ice. My original method was as follows:

```
# Create a label for total volume and set a starting point of 0

total_volume = 0

# Loop through all the locations where ice is above sea level

# For the coordinates in the list "above_sea" created:

for coordinates in above_sea:

    # Get the height for the coordinate in metres by dividing by 10

    _height = lidar_data[coordinates[0]][coordinates[1]] / 10

    # Work out the volume of the 1m x 1m area
    _volume = 1 * 1 * _height

    # Add volume to the total volume

    total_volume = total_volume + _volume

print("height", _height)
```

However, printing height to the output returned the value of 15.1m to the output. I realised this was surprisingly small compared to what I was expecting. To overcome this issue, I instead defined a function that returned all the values of height for each coordinate where I found there to be ice. As each of these heights corresponded to a 1m*2 area of ice, I was able to calculate the total mass by multiplying the sum of the heights by 1m and by 1m again – in other words, the sum of the heights cubed.

The second issue I came into was getting the model to return the values for 'height' in a format that could be summed together. Originally I appended the values to a list '.append[y]'. When looping through the coordinates to find the height, I was only able to get a list of lists to print to the output. This meant that when I tried to sum the total heights, I would get an error message. I overcame this by appending the values to the label 'berg_height' as an object, using parentheses '.append(y)' instead of squared brackets. This allowed me to access each value, rather than a list of lists that could not be summed.

Thought process going into the software design

I split the task into manageable sections and decided on possible code that could be used at each stage. This worked as follows:

1 Import relevant modules needed for the code to run
   - Import function

2 Create all labels needed for use further down the code.

- Attach values or set as empty lists for values to be inserted into.

3 Read in the lidar and radar files

- With open as f:
- Referred to previous assignment to find this code

4 Create and display the graph (GUI)

- Using matplotlib imported earlier in code

5 Find the coordinates where there is ice

- For loops

6 Return the corresponding values for height at each point

- Define a function that returns values
- For In statement

7 Calculate total mass of the iceberg

- Using mathematical formulas and the information provided in the instructions.

8 Print whether the iceberg can me moved in time to the output

- If Else statement.

Software development process followed

To begin with I printed out every step to the output. For example, I printed out the full list of coordinates where there was ice and the full list of values for height. I did this to check that my code was working as expected. To develop the code and make it more efficient, I turned off the print function for these parts of the code.

I also found it useful to use plain text within each ‘print’ function call. For example, using labels such as “volume” and “total mass” before printing a value. This enables not only me, but also any user of the model to see which values in the output relate to what label. This overall improved my codes readability.

I also had multiple .py files in use throughout the process. I did this so I could test out different code to see how it would work, before adding it to my main model file. This ensured my code didn’t get messy, and that only useful lines of code were included in the final model.

How the code is tested:

The code is timed for efficiency. The code has commented out sections that can be turned on to display coordinates and values to the output to show that the code is working correctly. The code checks the length of lists in order to see whether the correct number of values have been returned for height.

How I could work on this code to improve efficiency and make it more advanced:

Utilise more functions to shrink code to a fewer number of lines. Import data for multiple icebergs and determine which ones could be moved and which couldn't - functions could also be utilised in order to make this process more efficient and shrink the number of lines of code needed. The code could also create a more advanced GUI, for example, with a title.