A4-Doc-Faycal-Kilali

Made by Faycal Kilali for Programming Assignment 4, November 27, 2022 for Comp-1631.

Description / Utilization instructions

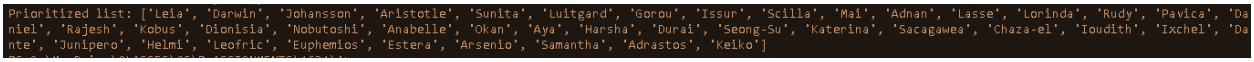
For Python: open the file in a python IDE, then run the code. Alternatively, invoke python onto the .py file and watch it work! You can invoke it in windows (if your pathing is correct for your Python installation environment) by **‘python A4-Kilali-Faycal.py’** w**i**thout the ‘’ marks. For the additional tests, you’ll need to edit the scripts as appropriate to the test suite provided.

Test Suite:
Intended output in general:

NOTE: I did change the folks.py provided a bit to make it easier to test! DO NOT COMPARE IT WITH TEH ORIGINAL AS MY RESULTS MAY BE DIFFERENT.

Test 0:

*This test was chosen in order to demonstrate an overall properly sorted dictionary of **many** people based on the rulesets we've defined.*

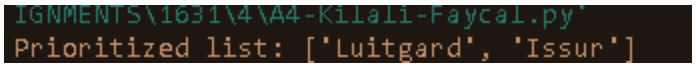


Test 1:

This test was chosen in order to demonstrate an easy to see example of survivability being the same (hence the pregnancy rule is checked for in this particular case, and makes a difference)

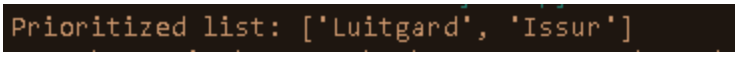
Test 2:

This test was chosen in order to demonstrate an easy to see example of survivability differing, with pregnancy the same (in here, pregnancy should not make a difference)



Test 3:

This test was chosen in order to demonstrate an easy to see example of survivability differing, with pregnancy NOT the same (in here, pregnancy should not make a difference)



Test 4:

This test was chosen in order to demonstrate a easy to see example of a working insertion of a new person with new attributes.

```
e' 'c:\Users\fayca\.vscode\extensions\ms-python.python-2022.18.2\pythonFiles\lib\python\debug
IGNMENTS\1631\4\A4-Kilali-Faycal.py'
Original prioritized list: ['Leia', '  Aristotle'],
New prioritized list: ['Leia', 'Carly', '  Aristotle']
```

Test 5:

This test was chosen in order to demonstrate a test for the large dictionary, where we insert a new person. The expected output is the person would be inserted into the correct position based on the rules we've defined.

```
Original prioritized list: ['Johansson', 'Darwin', 'Aristotle', 'Leia', 'Sunita', 'Luitgard', 'Gorou', 'Issur', 'Scilla', 'Mai', 'Adnan', 'Lasse', 'Lorinda', 'Rudy', 'Pav
ica', 'Daniel', 'Rajesh', 'Kobus', 'Dionisia', 'Nobutoshi', 'Anabelle', 'Okan', 'Aya', 'Harsha', 'Durai', 'Seong-Su', 'Katerina', 'Sacagawea', 'Chaza-el', 'Ioudith', 'Ixc
hel', 'Dante', 'Junipero', 'Helmi', 'Leofric', 'Euphemios', 'Estera', 'Arsenio', 'Samantha', 'Adrastos', 'Keiko'],
New prioritized list: ['Johansson', 'Darwin', 'Aristotle', 'Leia', 'Sunita', 'Luitgard', 'Gorou', 'Issur', 'Scilla', 'Mai', 'Adnan', 'Lasse', 'Lorinda', 'Rudy', 'Pavica',
 'Daniel', 'Rajesh', 'Kobus', 'Dionisia', 'Nobutoshi', 'Anabelle', 'Okan', 'Aya', 'Harsha', 'Durai', 'Seong-Su', 'Katerina', 'Sacagawea', 'Chaza-el', 'Ioudith', 'Ixchel',
 'Dante', 'Carly', 'Junipero', 'Helmi', 'Leofric', 'Euphemios', 'Estera', 'Arsenio', 'Samantha', 'Adrastos', 'Keiko']
```

Test 6:

This test was chosen in order to demonstrate the case where two people are completely identical and how they are prioritized (in here, the random number generator of the algorithm kicks in, to ensure fairness, so the result will be 50% chance of one or the other person each time the list is sorted.

NOTE: this'll have two different results based on chance as per our algorith, both are demonstrated in two separate pictures.

```
Prioritized list: ['Luitgard', 'Issur']
```

```
Prioritized list: ['Issur', 'Luitgard']
```

Self reflection (ideas on improving the project or one’s approach):

- If I had more time, I’d implement more rules but finals is fast approaching. a criminal record rule, with the caveat that some crimes should hold no weigh (especially if the incarnation is false, which happens quite often in the states.).
- In particular, for criminality, if someone is a serial killer it’d be far easier to make the rule set for criminal record rule as the governmental justice system isn’t always just, which can be seen quite clearly in the states with their false incarnation rates (I don’t have access to that, though).
- My rules could be more specific and just if I had more specific information, as demonstrated in the previous bulletpoint.

The following rules would’ve been implemented if I had more time: **Age**: If the age difference is significant enough (In here, we set a concrete value of an age difference of 30), then the younger person will be the one to save, as they are more likely to have a far longer life to live, and hence are more likely to contribute to humanity technologically or socially.
Occupation: Only two occupations hold weight in the list of occupations provided in the folks.py file, those being Doctor and Teacher, with Doctor holding more weight than Teacher for deciding who gets saved.
The merit behind why those two hold weight is as follows.
Doctors can save many lives, and or advance medicine. Whilst teachers can teach many people, certainly serving a role overall in contributing to humanity.
The reason why a Doctor holds more weight than a Teacher is due to the shortage of medical staff, especially given epidemics, and the overall suffering that results as a consequence thereof, that is, they are incredibly vital for the health and wellbeing of society. Although Teachers are also vital, with the wealth of knowledge available currently one can self-teach, and or learn from textbooks and so on to the point where its arguably less of a vital role than a Doctor.
Criminality record: If a person is a convicted criminal (that is, they hold a criminal record), then given that all the rules above are tied, the person with no criminal record will have preference (this is a deterministic decision as statistically those who had been convicted in the states are very likely to recommit a crime, therefore less likely to contribute to humanity’s technology or society, and extending this ruleset would require

a serious amount of determining just what the crime is, which will branch to a whole project in its own)

Comments:

Very interesting programming assignment, only issue I'm seeing with it is that you refer to people as "patients" in the PA4.pdf sometimes which seems to likely be a typo. Additionally, I think people should be allowed to demonstrate what morals they want to install rather than being told they can make bad rules (which seems to tell me that they'd get bad grades if you interpret them as "bad rules") where bad rules is a topic of ethical discussion. Its best to let them be entirely free with the rules they choose, within a reasonable manner, rather than limit their creativity to the subjectivity of some of the ethical dilemmas present in those contexts.

Additionally, this is a rather lengthy PA, especially around finals.

Plan (and justification for the ruleset)

In order to maximize the amount of people we save in the context of the scenario provided, we save the person(s) based on the following descending order of importance (priority).
Note in the rules below, we consider them in descending order priority, that is in such a way that we'd only consider pregnancy if our rules for Survivability were determined to be insufficient to make a decision for the two potential victims priority (that is, a tie in the rule), and so on.
Overall, this approach attempts to be as unbiased as possible, and weighs the potential of the individual contributing to humanity or society overall after the survivability rule when other rules are not satisfied (you can see this under Occupation and Criminality Record *(In US, the national recidivism rate is 67.5 percent, so its less likely they'll contribute)* ruleset)

Survivability: The person with the highest survivability is saved first in all cases as this maximizes the amount of people we save overall.
Pregnancy: it may potentially count as two people, so we can argue that saving a

pregnant person is also saving their baby in a way, hence increasing the amount of people we save.

If all the above rulesets are tied, then one of the two potential victims will be randomly chosen by a random number generator in order to make it as fair as possible.

All other attributes will hold no weight on whether the person will receive preference or not for being saved, that is, they hold no weight and will hence not be considered.

Algorithm:

(This algorithm uses indentation to better convey the order of execution of the process).

Define our dictionary of people.

Retrieve the attributes of individuals as a list (list_of_attributes)

Take the survivability value of each key:pair and insert it into a new list labelled list_of_survivability_og.
We create a shallow copy of the survivability_og list.

for each element in our list of survivability_og,

> we find the highest value, we then use that to find its respective index in survivability_og, we insert that into a new list that we call the survivabiltiy index, then we “substitute” in “False” for the position of that element in survivability_og (so we don’t loop through the same max element multiple times)

We reorder the list of attributes by using our sorted list of indices based on the sorting of survivability that we’ve performed as a new list. We then create a list of tuples of two values (pregnancy, survivability) in the same ordered manner. (We do this by swapping positions in the indices of the list of names based on the ordered survivability indices we found, for each element in our list of names, likewise we do the same for the attributes to keep it ordered.)
We check if the survivability is the same between any two people in the list, if it is, then we check if one of them is pregnant, if they are they get preference, if it is a tie (both not pregnant and or both pregnant) then we invoke a random number generator to dictate who gets preference with both having an equal chance of being picked, whoever gets picked gets preference.

In order to insert a new person to the list, we process the unordered people list to acquire a prioritized people’s list (prior to the person being added to it, for comparison purposes and later insertion), then we insert the person to the unsorted people’s list (with the new person’s attribute, at the end of the list). Afterwards, we process the unsorted people’s list (with the new person and his attributes added to it). Then, we find the index of the person in that processed list, and we insert it into the prioritized people’s list that we initially found at that index position.