Classes and Databases: `Person`

ISTA 350 Hw3, Due 2/27/2020 at 11:59 pm

Introduction. This homework is intended to review and extend your knowledge of classes in Python and to practice storing a program's state in a database between runs. You will be writing the first class in a program that encapsulates the concept behind Facebook. We may finish the program later in the semester. Don't turn in code that hangs the test program – this will result in a 0.

Instructions. Create a module named `hw3.py`. Below is the spec for four methods and eight functions. Implement them and upload your module to the Hw3 D2L Assignments folder.

Testing. Download `hw3_test.py` and auxiliary testing files and put them in the same folder as your `hw3.py` module. Run it from the command line to see your current correctness score. Each of the methods and functions is worth 8.33% of your correctness score. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, `ISTA 350 Hw3`, and a brief summary of the module. Each method/function must contain a docstring. Each docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code (not necessary on this assignment).

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

Resources.
https://docs.python.org/3/tutorial/
https://docs.python.org/release/2.5.2/lib/string-methods.html
https://docs.python.org/3/tutorial/classes.html
https://docs.python.org/3/library/sqlite3.html
http://www.sqlite.org/index.html
Practice in the SQLite3 shell: https://www.sqlite.org/cli.html

Nice pages on creating decorators (examples are in Python 2). However, you will see the * operator (not multiplication), which we haven't learned. It is variously called star-args, the splat operator, and the unpack operator. It packs/unpacks sequence objects. Warning: these pages are likely to look like mumbo-jumbo at this point.
http://www.artima.com/weblogs/viewpost.jsp?thread=240808
http://www.artima.com/weblogs/viewpost.jsp?thread=240845

`class Person:`

`init`: Each `Person` object has four instance variables, called `first`, `last`, `bday`, and `email`, and `init` has four corresponding string parameters, each of which has the empty string as a default argument. The first parameter is the `Person`'s first name, the second the last name, the third his/her birthday, and the last the `Person`'s e-mail. If any of the parameters are the empty string, get a value from the user using one of the following prompts:

```
Enter person's first name:
Enter person's last name:
Enter person's birthday:
Enter person's e-mail:
```

Each colon is followed by a space.

`repr`: This method returns a string in the following format: `'Rich Thompson: 5/21, rm@g'`.

`read_person`: This is a class method that reads the data necessary from a text file to create and return a `Person` instance. It takes one argument, a file object (not a filename). It reads a line from the file. If the line is empty, return `False`. Otherwise, use the contents of this and the next three lines as the first name, last name, birthday, and e-mail of a new `Person`. Remember to use the `classmethod` decorator.

This method allows one to traverse a file of `Person` objects, or perhaps `Person` objects with some other data interspersed, and read and create one `Person` object from it at a time.

`write_person`: This instance method takes one argument, a file object. It writes the instance variables, one per line, to the file in this order: first, last, birthday, email.

--------- Function Specifications ----------

`open_persons_db`: This function returns a connection to a database. Read the `os.path` documentation to learn how to determine if a file exists. Determine whether or not `persons.db` exists and store this information in a variable. Connect to `persons.db`. Set its `row_factory` to `sqlite3.Row`. If it's a new database, create `friends` and `colleagues` tables with column names `first`, `last`, `bday`, and `email`. These are all `TEXT` fields. `email` is the primary key. Return the database.

`add_person`: This function has four parameters. The first is a `Person` database as described above. The second is a `Person` object. The third is a Boolean with a default argument of `True` that tells the function if the `Person` is a friend. The last is a Boolean with a default argument of `False` that tells the function if the `Person` is a colleague. Name the last two parameters `friend` and `colleague`, respectively (this is necessary for testing purposes).

If both Booleans are `False`, print the following message and return `False`: `'Warning: <email address> not added – must be friend or colleague'`. By default, the `print` function prints to `stdout`. Print your warning to `stderr`. To do this, import `sys`, then pass `sys.stderr` to the `print` function using the keyword `file`. (In Powershell, there is no visible

difference between printing to `stdout` and `stderr`. In many other programs, text printed to `stderr` will print to the console in red.)

Otherwise, insert the `Person` into the appropriate tables and return `True`. Don't forget to commit your changes to the database.

`delete_person`: This function takes a `Person` database and a `Person`. Delete the `Person` from all tables.

`to_Person_list`: This function takes a `cursor` object as its sole argument and returns a list of `Person` objects constructed from the data in the rows iterated over by the `cursor`.

The following functions should use `to_Person_list`:

`get_friends`: This function takes a `Person` database as its sole argument and returns a list of `Person` objects representing all of the friends in the database.

`get_colleagues`: This function takes a `Person` database as its sole argument and returns a list of `Person` objects representing all of the colleagues in the database.

`get_all`: This function takes a `Person` database as its sole argument and returns a list of `Person` objects representing all of the friends and colleagues in the database without duplicates.

`get_and`: This function takes a `Person` database as its sole argument and returns a list of `Person` objects representing all of the people who are both friends and colleagues in the database.