Assignment 5 Python - Art Database

Due Feb 18, 2016 by 12pm **Points** 90 **Submitting** an external tool
Available until Feb 20, 2016 at 12pm

This assignment was locked Feb 20, 2016 at 12pm.

As an art historian, you are writing code to allow the world's museums and art collectors to each maintain their own database of their collections. Anyone could then learn and use this simple, standard interface to search any of the databases. This would make it easy to find any art piece of interest, thus encouraging museum loans, collector sales, academic studies, and such.

From a programming perspective, this assignment emphasizes nested data structures, especially dictionaries, and mutation.

- No code is provided.
- **Test** (http://codeskulptor.appspot.com/owltest/?urlTests=comp130.assignment_art_database_tests.py&urlPylintConfig=comp130.pylint_config.py)
- See submission button at the bottom of the page.

For your databases, you decide to only have a few pieces of information about each artwork — artist name (string), artwork name (string), year created (integer), description (string), and owner (string). Based upon client research, you believe the most common usage will be for people to search for one or more artworks by the same artist using the artist name and artworks' names. You decide to optimize your data organization for this common usage by using two levels of dictionaries, keyed by the artist name and artwork name, respectively. Thus, your data structure is a dictionary that maps each artist name to a dictionary that maps each artwork name to a tuple of year, description, and owner. (Note: You may use a default dictionary for either level, if you prefer.) You will probably want to make one or more example databases to test your own code.

Your goal is to write the following functions to create and modify any database with the structure described above.

- **(3 points)** `empty_db()` — Returns an empty database, i.e., an empty dictionary or default dictionary.

  Such a function is called a constructor, since it constructs a database.

  The purpose of this function might be a bit mysterious. The idea is that you can use either a dictionary or a default dictionary for the database. Moreover, if we hadn't placed that restriction on you, you could have represented the database with a list or tuple. Whatever your representation is, it should be somewhat hidden from the user of your code. By calling this function, someone can create an empty

database without knowing how you represent the database. In fact, within your own code, any time you need an empty database, you should also be calling this function.

- **(10 points)** `add_item(database, artist, artwork, year, description, owner)` — Mutates the database to add one artwork. The database need not contain anything by that artist previously. However, if this is a duplicate artwork name for a particular artist, it instead does not mutate the database. It does not print any error message in this case. It returns a Boolean `True` or `False` indicating whether the artwork was added to the database.

- **(10 points)** `change_owner(database, artist, artwork, new_owner)` — Mutates the database to change the owner of the indicated artwork by the indicated artist. If no such artwork is in the database, there is no mutation. It returns a Boolean indicating whether the artwork ownership was updated.

  Note that the owner data is within a tuple, so you cannot just mutate that data within the tuple. Instead, you need to create a similar tuple, and mutate the database to contain this new tuple.

When getting ("selecting") information from the database, the following operations each return a data structure of the same form of the overall database. This allows the result to be the input of another select operation to further narrow down which data you are interested in. Write the following functions.

- **(10 points)** `select_artist(database, artist)` — Returns a new database with all of the information for the given artist in the given database.

- **(10 points)** `select_artwork(database, artwork)` — Returns a new database with all of the information for any artwork with the given name in the given database.

- **(10 points)** `select_year(database, year)` — Returns a new database with all of the information for any artwork from the given year in the given database.

- **(10 points)** `select_description(database, keyword)` — Returns a new database with all of the information for any artwork having a description containing the given keyword in the given database.

- **(10 points)** `select_owner(database, owner)` — Returns a new database with all of the information for any artwork owned by the given owner in the given database.

After selecting the desired results, printing out the resulting data as a dictionary wouldn't be very user-friendly. So, write the following function.

- **(10 points — not automatically tested)** `format_results(database)` — Returns a string with one line per artwork. For each artwork, it lists the artist, artwork, year, description, and owner, each separated by a comma and a space.

  This is not automatically tested because we have not specified a particular order for the database items to appear. Sorting the data would be relatively difficult at this point in the course.

Of course, you should your code should have good style and documentation.

- **(10 points — not automatically tested)** **Code style**

- **(9 points — 1 point per function — not automatically tested)** Documentation strings are accurate and precise.

An Aside

Later in the course, we will introduce the idea of object-oriented programming. Part of the idea is a generalization of this example. All of these functions have to do with databases. Furthermore, all of them need to know the representation of a database. So, if we package a database and all of these functions into an "object", everything that deals with the database representation is contained within that object. Any code outside the object won't be able to see or change that representation without going through the object and its functions.