# CS 340 README

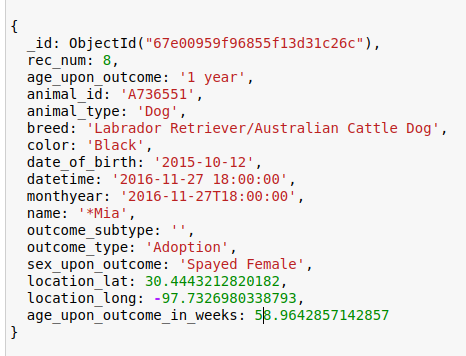
## About the Project/Project Title

CS 340 Grazioso Salvare Dog Database Project

<https://youtu.be/Hj-VlHxX-QQ>

## Motivation

The project aims to efficiently select shelter based dogs that are suitable to work in different rescue missions and its environments. The project looks up a database that contains animals and its particular traits. Here is an example of one dog returned from the database collection:



It shows a Labrador Retriever/Australian Cattle dog as well as its age and name. Knowing the breed of the animal and age are important factors to consider when searching for an effective rescue animal. In order to accomplish the mission, several objectives must be met. The MongoDB database would need to interact with the client. To do this, there is a python driver as well as CRUD functionality to create, read, update, and delete the database collection. In addition to that, there will be a client-side interface so that the client can use a no code approach. This increases client productivity by using intuitive search and filtering instead of doing a manual search for the best animal for the job.

## Getting Started

As an open source project. It will be available in the open source platform GitHub. MongoDB was chosen due to its high availability, developer productivity, and low complexity. It is also well suited to scale as business grows. Lastly, Python3 is integrated with MondoDB via a driver, and this allows the project to interact with the database via user interaction in the web or else where. The python module is called python\_crud\_final and the main library should import the AnimalShelter class.

Overall, python3 allows for advanced data analytics and integration with frameworks such as Dash, the front-end part, dashboard, of the client server. Additionally, python simplifies integrating with external projects by importing the project into the script. MongoDB allows for highly scalable applications and reduced time managing the schema compared to RDBMS.

To get started:

|  |
| --- |
| git clone [repository URL] cd project mongoimport --username="${MONGO\_USER}" --password="${MONGO\_PASS}" \ --port=${MONGO\_PORT} --host=${MONGO\_HOST} --db aac -collection animals \  --authenticationDatabase admin --drop ./aac\_shelter\_outcomes.csv -type csv\ --headerline ./aac\_shelter\_outcomes.csv -v jupyter notebook |

jupyter notebook provides a convenient method to view and test the CRUD python module, which interacts with the MongoDB database. The only other option would be to interact directly with MongoDB via the terminal. Jupyter notebook is going to be what allows clients to visualize the data. Every component will get integrated into one window via jupyter notebook.

Additionally, we are logging into the database as aacuser, who has read/write privileges on the AAC database only.

|  |
| --- |
| db.runCommand({connectionStatus:1}) {  authInfo: {  authenticatedUsers: [ { user: 'aacuser', db: 'admin' } ],  authenticatedUserRoles: [ { role: 'readWrite', db: 'AAC' } ]  },  ok: 1 } |

## Installation

MongoDB Community Edition

Linux OS: Ubuntu

Jupyter Notebook

Python3

## Usage

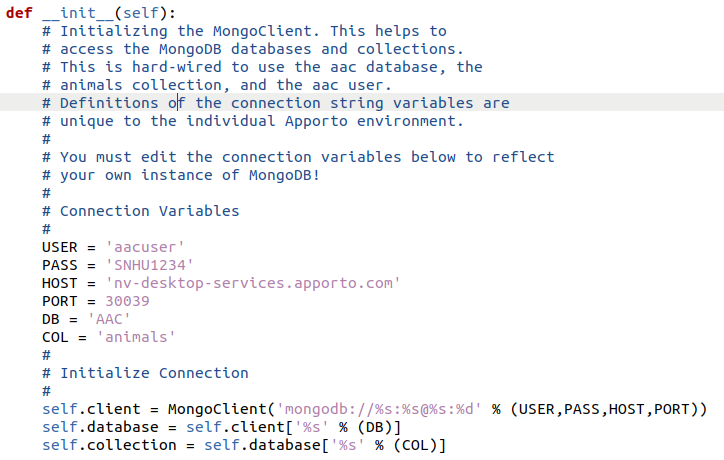
The python module implements all functionalities of CRUD: create, read, update, and delete. Additionally, update and delete functionality provides the ability to operate on one or more document. Besides CRUD, the database and module authenticates directly to user aacuser and connects to the AAC database and animals collection.

### Code Example

The module contains 2 libraries pymongo and bson.objectid in order to make a connection to MongoDB.



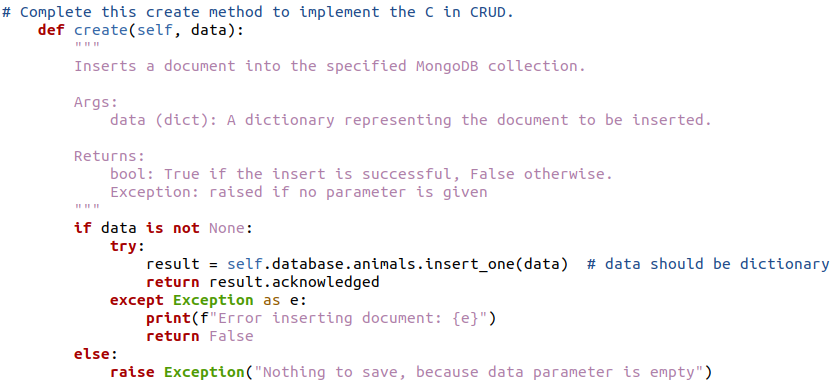
Next there is an AnimalShelter class that is used by our main process. The \_\_init\_\_ method is the first to run.



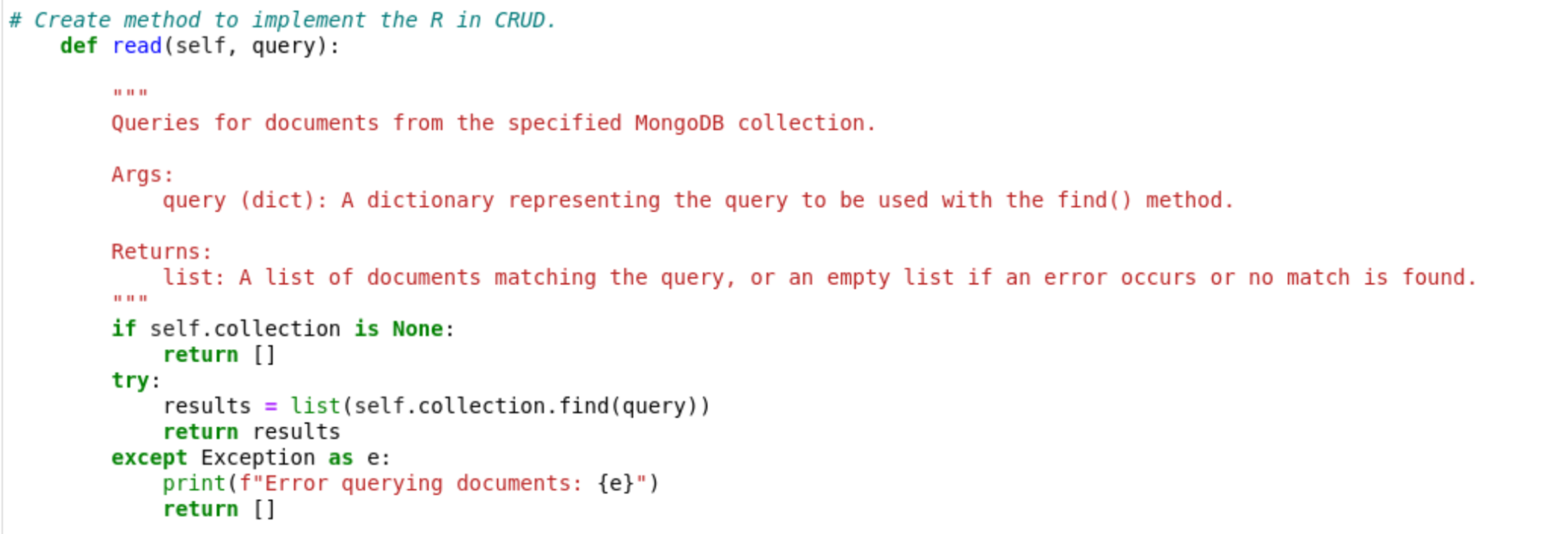
As shown, the init contains hard coded values that authenticates to the database as aacuser, AAC database, and animal collection. MongoClient, imported from pymongo, is used as the driver that connects to MongoDB. Additionally, 3 properties are set: client, database, and collection. collection is used to make the CRUD operations.

With that done, the module contains the 4 CRUD implementations.

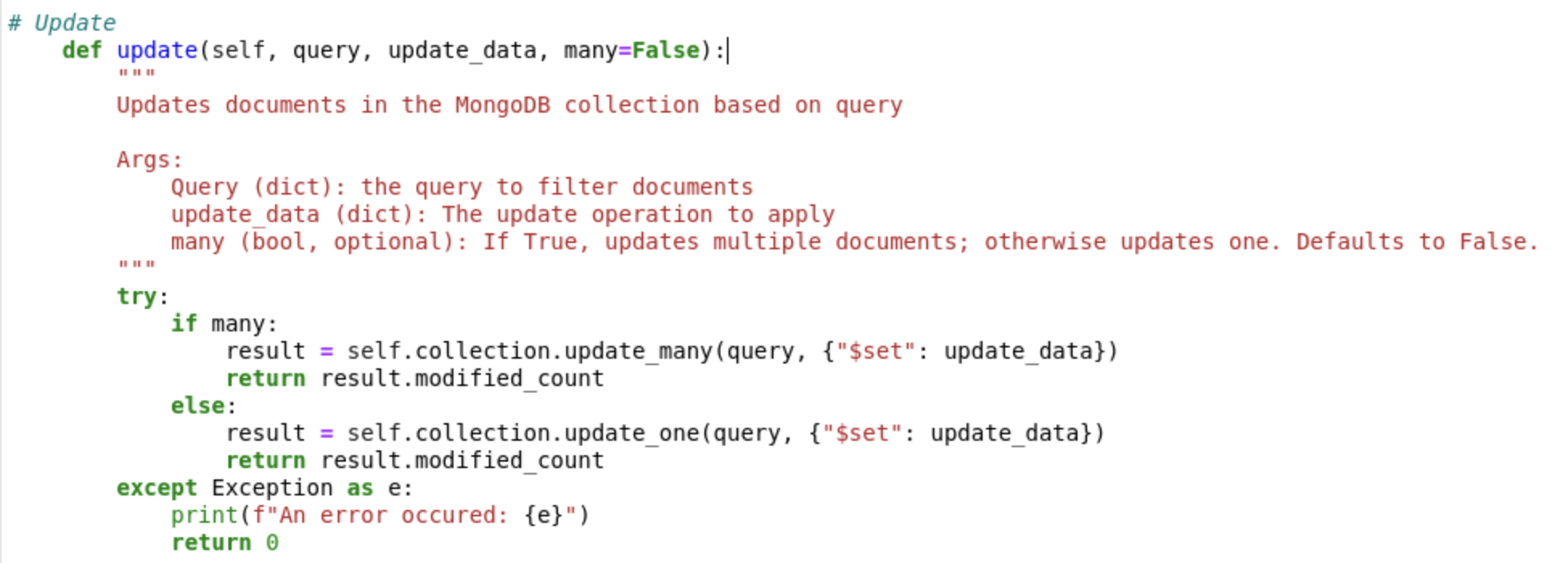
The first is create. It takes a dictionary argument and inserts one document with the insert\_one API. The return value is True or False, depending on successful execution.



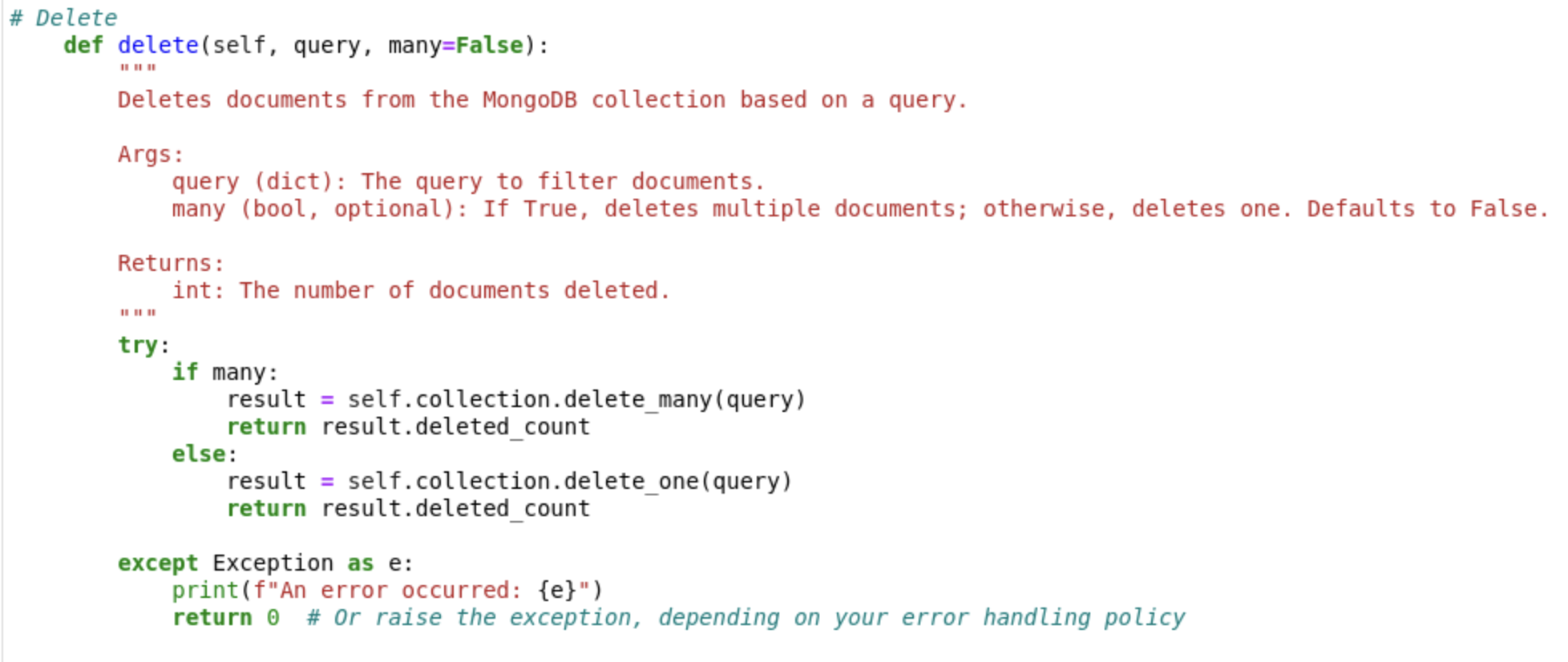
Next is read. It takes a dictionary argument that is looked up in the database. The read method returns a list of documents that matched the argument. If no matches are made, the read method returns an empty list.



The update method takes 3 arguments. The query arguments takes a dictionary to filter the collection. update\_data argument is a dictionary that gets upserted into the filtered documents. The last argument many has a default value of False. A false value updates one document, whereas a True value updates all of the filtered collection. The return value is the number of updated documents.



The delete method takes a dictionary and bool argument. The method uses either delete\_many or delete\_one API depending on whether the boolean value is True or False. True results in more than 1 being processed. The return value is the number of documents deleted.

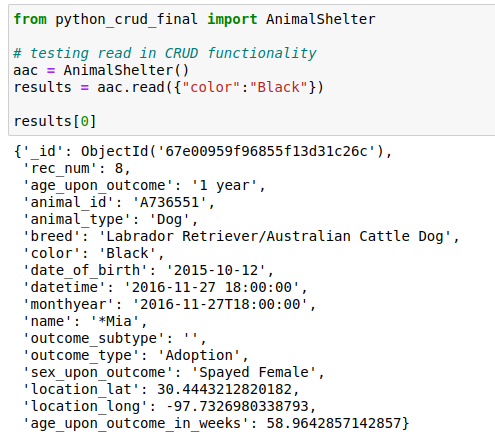


### Tests

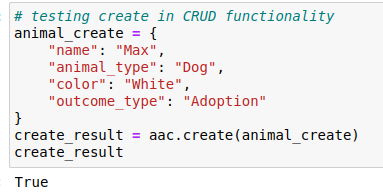
Proof of concepts have been tested on the 4 CRUD functionalities of the python module. The module is imported into the main python process; the ipynb file

### Screenshots

The read method is tested. The python module is imported and instantiated with aac. aac is used to call the read method. The return value is a list and the first value is printed by explicitly calling the first element of the list.



In the create proof of concept. A dictionary is created and is the argument for the create method. The create method returns a boolean value to show whether the operation was processed or not.



Update method is able to update one or more values. As you can see below, a document is created into the database to test the update method. In cell 2, the document is updated from name:Max12345 to name:Jane12345. The result is a value of 1, the number of affected documents. Furthermore, it shows that Jane12345 is deleted and not Max12345. This is because Max12345 no longer exists in the database when the delete method is executed.



The update\_many functionality is implemented when the update method uses True argument at the second position argument. To test this functionality, a names\_to\_update array of two created names is used to filter the collection. filter variable uses the array of names using the $in operator. The first argument to update is the filter. The second is what should be updated in the returned list of documents. In this case, an additional property and value is added: flag:True. The third argument specifies that we are working with more than 1 value. In other words, the update\_many API.



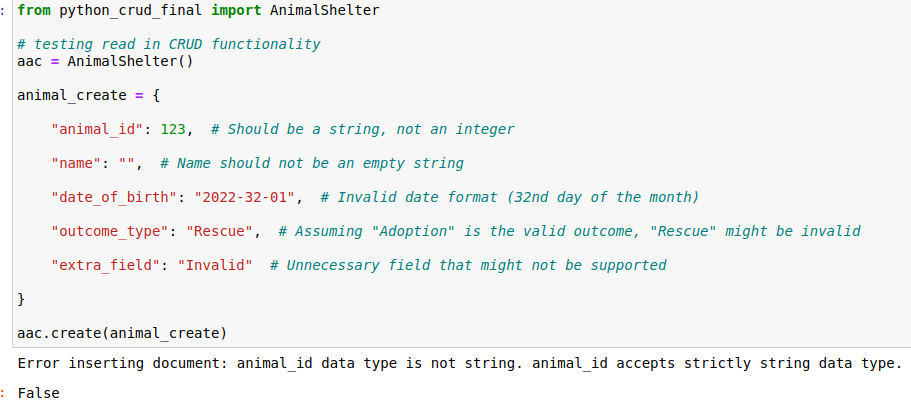
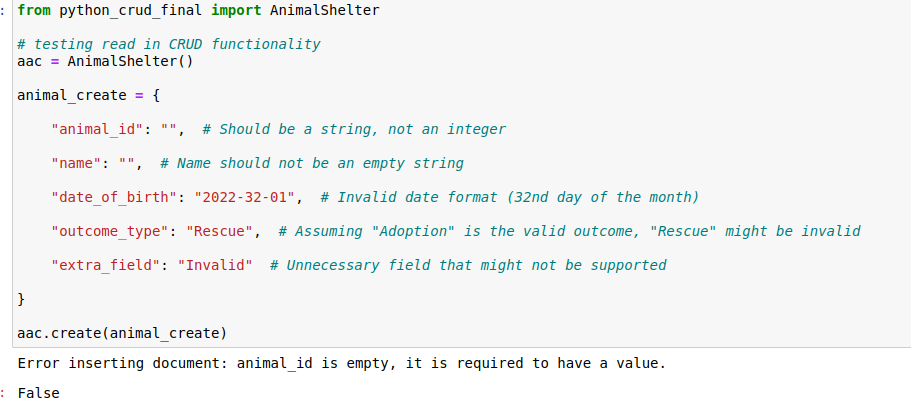
In the delete, the first functionality to test was deleting one value from the collection. The create method is called, creating a document with name of Max12345. Next the document is queried and displayed. Delete method is called with the default argument of False. It returns a value of 1, showing the 1 document was deleted.



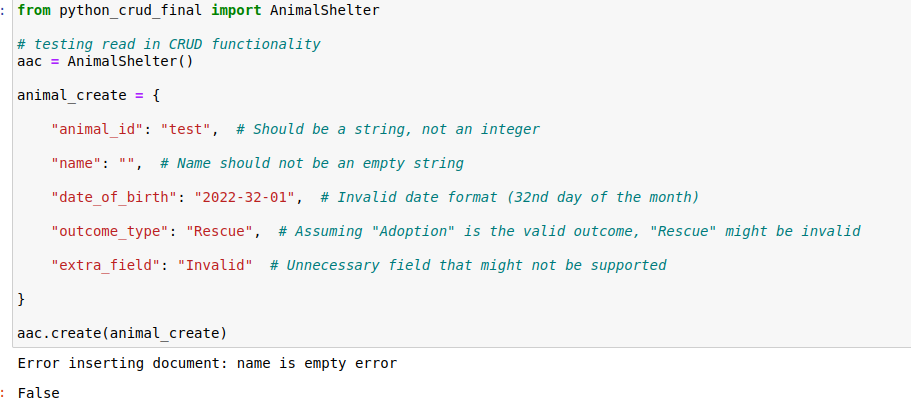
To delete many, mongoDB requires more than 1 value to query in order to delete. This is done through the logical query $or, which would take an array of documents. MongoDB will then filter through all documents that match each of the dictionary values in the filter array. The delete method returns the number of documents deleted.



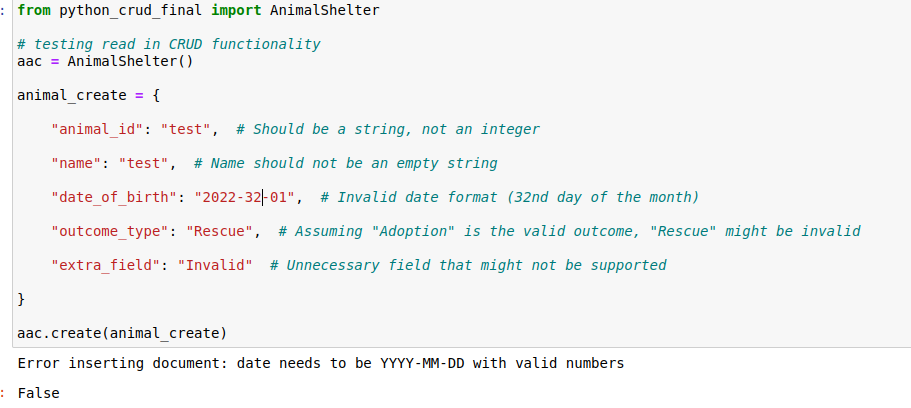
Additional tests were done to check for invalid inputs such as with the create method. For animal\_id. The next screenshots show two tests checking for values that are not strings and empty strings. If those kinds of inputs are entered, the module will return an error.

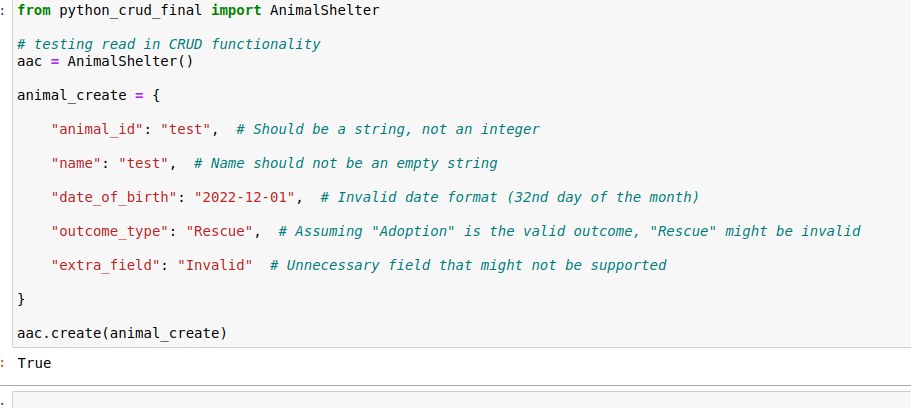


For name, test is done to check that name can’t be empty value.



Date of birth validation checks that proper numbers are only accepted





## Contact

Fellipe de Moraes