# Animal CRUD: A Friendly and Helpful Guide

## About the Project/Project Title

The AnimalCRUD.py python module is intended to be a bridge between a main python script and the Austin Animal Center Mongo database. This python module implements for CRUD (create, read, update, delete) functionality for the Pymongo module which interfaces with a MongoDB server. With this module, you can **C**reate documents, **R**ead documents, **U**pdate documents, and **D**elete documents within the animals collection of the database.

## Motivation

This module was developed in order to support a larger data analysis project. Having it as an independent module enables reusability for any other projects. By leveraging the pymongo module, It establishes a set of commands which will allow a developer who is unfamiliar with MongoDB to work with the database without needing to be familiar with the specific mechanics of MongoDB. Python is not only beginner friendly but also offers a robust suite of data analysis tools. MongoDB is a newer style of database which has some advantages over traditional SQL such as being much easier to scale, in case the database system needs to expand at some point.

## Getting Started

**How server connections are handled:**

When connecting to a MongoDB server, you will need some information in order to achieve a proper connection.

* The USERNAME and PASSWORD values will be your username and password. If you set up your own MongoDB server, you should already know your username. If you did not set it up yourself, you may need to reach out to your organization in order to obtain your sign-in credentials
* The HOST value is very similar to the ubiquitous website URL and should already be familiar to a non-technical audience.
* Web traffic also requires a PORT. When using a browser like Microsoft Edge, when visiting sites like facebook the ports are usually assumed to be one of a few different standard web browsing ports. Here, we must cannot assume the port. Finding which port you need is beyond the scope of this readme; please reach out to your server administrator.
* The DB variable is the database which you will be using. We previously named our database “AAC”, but yours may differ. Please run show databases in the mongo shell in order to determine which databases you can use.
* The COL or COLLECTION variable is the collection which you will be using. Here, we are using the collection “animals”, but yours may differ. Please run show collections in the mongo shell in order to determine which collection you can use.

**Importing from CSV to MongoDB:**

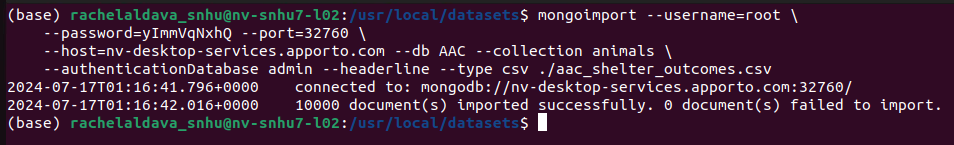
Currently, AnimalCRUD does not have the functionality to import CSV documents into a server, although this is a planned feature. In order to import from a CSV file, you will need to enter terminal and type in the following command:

mongoimport --username=root \

--password=yImmVqNxhQ --port=32760 \

--host=nv-desktop-services.apporto.com --db AAC --collection animals \

--authenticationDatabase admin --headerline --type csv ./aac\_shelter\_outcomes.csv



Please reference the above section in order to determine which values to change. Additionally, you may want to focus on where your CSV file is located. The “./” portion of “./acc\_shelter\_outcomes.csv” is a linux specific notation meaning something along the lines of “in the directory in which the terminal is running”. For convenience, you may wish to use the full path. In Windows, this might be something like "C:\Users\rache\Documents\ acc\_shelter\_outcomes.csv ".

**Creating a new user:**

If you have not already done so, you may wish to add a non-administrator user. Doing so can enhance the security of your project. You can enter the mongo shell by typing “mongosh”, typing “use admin”, then typing a command similar to:

db.createUser(

{

user: "aacuser",

pwd: "aacpassword",

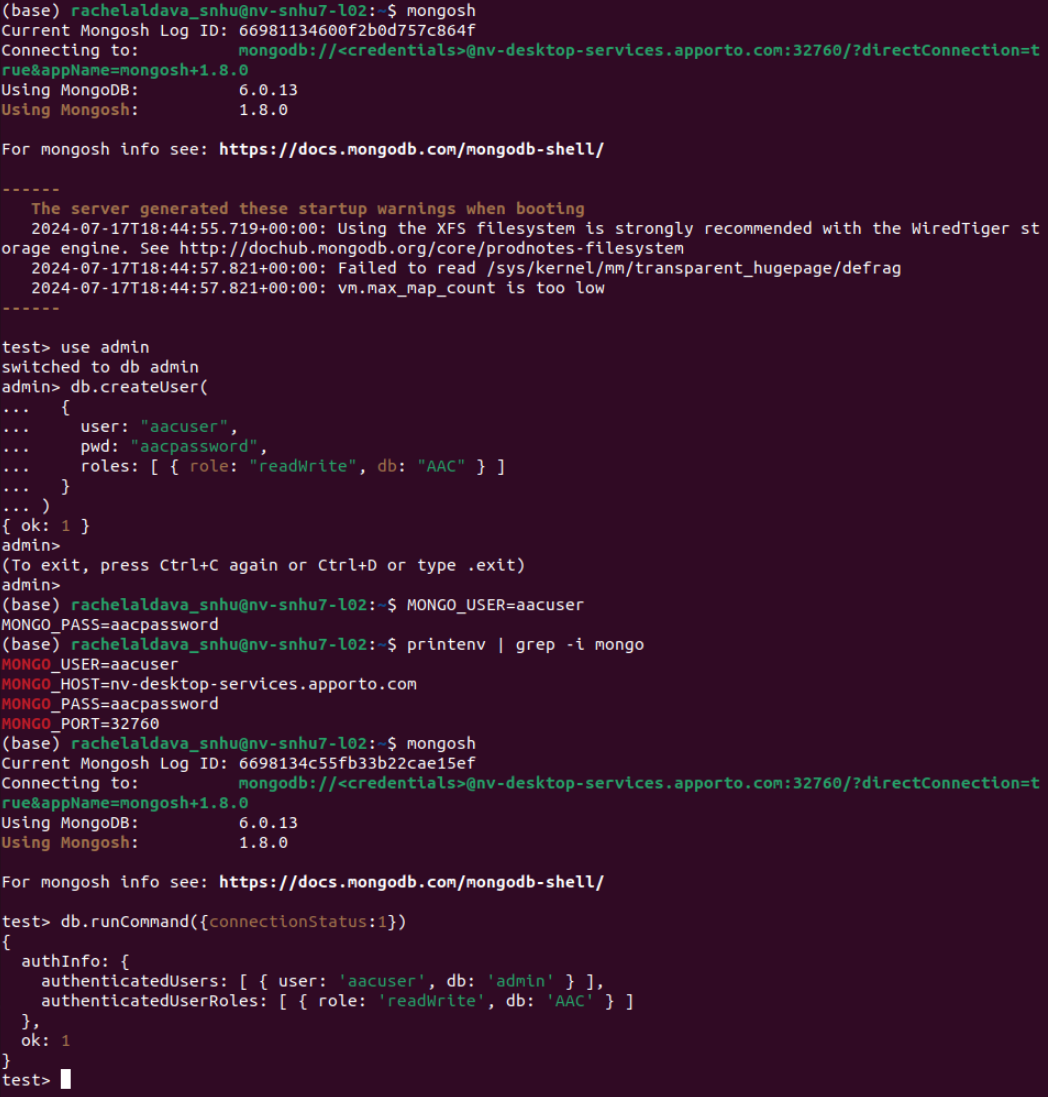
roles: [ { role: "readWrite", db: "AAC" } ]

}

)

Where “user” is the name of the user you wish to create and “pwd” is the password.

Below is an example of these commands being executed:



Please make a note of the four lines with **RED** letters, starting with **MONGO**\_USER. This will be the sign-on information you will want to use in the next section.

**Getting started using AnimalCRUD:**

1. ensure you have the required files installed (please reference the next section: ”installation”)
2. Ensure that the server information is correct. You will need to open AnimalCRUD.py and change the HOST, PORT, DB, or COL values to the ones which you have been assigned (Please reference the “how server connections are handled” and the “creating a new user” for more information about achieving a proper connection)

def \_\_init\_\_(self, USER, PASS):  
 # Hard-coded database details - Edit these values where appropriate  
 # Host and port number are needed to connect with anything on the internet. For common types of traffic, such as loading a web page, the port numbers are standardized. For our application, you will need to verify the port your host is running on.  
 HOST = 'nv-desktop-services.apporto.com'  
 PORT = 32760  
 DB = 'AAC' # The name of the database you are connecting to  
 COL = 'animals' # The name of the collection you will be working with  
 try:  
 # Initialize Connection  
 self.client = MongoClient('mongodb://%s:%s@%s:%d' % (USER, PASS, HOST, PORT)) # connect to the host  
 self.database = self.client['%s' % (DB)] # Select the database  
 self.collection = self.database['%s' % (COL)] # Select the collection.  
  
 # Check connection  
 self.database.animals.find\_one()  
 except Exception:  
 print("Invalid server or login credentials provided.")

1. In the python file you are developing, you will need to import the file:

import AnimalCRUD

1. Then, whenever you are ready to start working with the database, you need to instantiate the object:

animals = AnimalCRUD.AnimalShelter()

The connection is tested in the process of executing AnimalCRUD.Animalshelter() If you encounter no errors, then you have successfully connected and queried the database.

Currently, there are four commands to interact with the database:

* 1. Create

# This method implements the C in CRUD: CREATE  
# data\_kv\_json should be dictionary  
# Returns a bool: True if the data was successfuly inserted  
def create(self, data\_kv\_json):  
 if data\_kv\_json is not None: # If the data exists  
 try:  
 return self.database.animals.insert\_one(  
 data\_kv\_json).acknowledged # insert the data and return a confirmation  
 except Exception: # This can happen when there are two duplicate documents  
 print("Attempted to create duplicate document. Skipping...")  
 return False  
 else: # else there's a problem'  
 raise Exception("CREATE ERROR: Nothing to create, because data\_kv\_json parameter is empty")

This is a function which inserts a new document into the database. This function’s parameter is a dictionary of key:value pairs which should be formatted like a JSON document. When adding documents, it may be good practice to ensure that your dictionary keys and values align with the existing documents in the database. Because I added .acknowledged at the end, the returned value will be a Boolean value; true if the insertion was successful, otherwise false.

* 1. Read:

# This method implements the R in CRUD: READ  
# search\_key\_value should be dictionary  
# Returns a python list of python dictionaries  
# Each dictionary in the list is a document  
def read(self, search\_key\_value):  
 if search\_key\_value is not None: # if there was a search parameter provided  
 # To return a cursor instead of a list:  
 # return self.database.animals.find(search\_key\_value)  
  
 # To return a list instead of a cursor:  
 return [doc for doc in self.database.animals.find(search\_key\_value)]  
 else: # else there was a problem  
 raise Exception("READ ERROR: Nothing to read, because search\_key\_value parameter is empty")

This is a function which searches through the database and returns a list of documents which match the search criteria. This function’s parameter is the search criteria, which is a dictionary of key:value pairs which should be a subset of values within a document in the database. {‘name’:’Rover’} would return a list of animals who are named Rover. Uncommenting the other return line would cause the returned datatype to change from a python list to a pymongo specific datatype called a cursor, which has some interesting features which are not used by this project.

* 1. Update:

# This method implements the U in CRUD: UPDATE  
# search\_key\_value should be dictionary  
# data\_kv\_json should be dictionary  
# The function is additive; if the dictionary contains fields which exist in the document, it updates;  
# else it inserts. It does not remove.  
# returns the number of documents which were modified (update\_one should only return 0 or 1)  
def update\_one(self, search\_key\_value, data\_kv\_json):  
 # if there is both a search value AND a data provided  
 if (search\_key\_value is not None) and (data\_kv\_json is not None):  
 # update the document and return the number of documents updated  
 return self.database.animals.update\_one(search\_key\_value, {"$set": data\_kv\_json}).modified\_count  
 else: # else there was a provlem  
 if (search\_key\_value is None) and (data\_kv\_json is not None):  
 raise Exception("UPDATE ERROR: Nothing to update, because search\_key\_value parameter is empty")  
 elif (search\_key\_value is not None) and (data\_kv\_json is None):  
 raise Exception("UPDATE ERROR: Nothing to update, because data\_kv\_json parameter is empty")  
 else:  
 raise Exception(  
 "UPDATE ERROR: Nothing to update, because search\_key\_value and data\_kv\_json parameter are empty")  
  
# This method implements the U in CRUD: UPDATE  
# search\_key\_value should be dictionary  
# data\_kv\_json should be dictionary  
# The function is additive; if the dictionary contains fields which exist in the document, it updates;  
# else it inserts. It does not remove.  
# returns the number of documents which were modified  
def update\_many(self, search\_key\_value, data\_kv\_json):  
 # if there is both a search parameter AND data provided  
 if (search\_key\_value is not None) and (data\_kv\_json is not None):   
 # Modify the documents and return the number of documents which were modified  
 return self.database.animals.update\_many(search\_key\_value, {"$set": data\_kv\_json}).modified\_count  
 else: # Else there was a problem  
 if (search\_key\_value is None) and (data\_kv\_json is not None):  
 raise Exception("UPDATE ERROR: Nothing to update, because search\_key\_value parameter is empty")  
 elif (search\_key\_value is not None) and (data\_kv\_json is None):  
 raise Exception("UPDATE ERROR: Nothing to update, because data\_kv\_json parameter is empty")  
 else:  
 raise Exception(  
 "UPDATE ERROR: Nothing to update, because search\_key\_value and data\_kv\_json parameter are empty")

There are two functions which will update documents. They are essentially the same, except that update\_one() updates the first document it finds which matches the search criteria whereas update\_many() updates all matching documents. Like read(), these functions will take a search parameter. The second parameter is the value you wish to update. Currently, updating is strictly additive; there is no way to remove fields without removing the entire document. If a field already exists, it will be rewritten with the data. Otherwise, the field will be added to the document. The returned value will be the number of documents updated.

* 1. Delete:

# This method implements the D in CRUD: DELETE  
# search\_key\_value should be dictionary  
# returns the number of documents which were deleted (delete\_one should only return 0 or 1)  
def delete\_one(self, search\_key\_value):  
 if search\_key\_value is not None: # if a search parameter was provided  
 # Delete the document, if there, and return the number of documents deleted  
 return self.database.animals.delete\_one(search\_key\_value).deleted\_count  
 else: # Else there was a problem  
 raise Exception("DELETE ERROR: Nothing to delete, because search\_key\_value parameter is empty")  
  
# This method implements the D in CRUD: DELETE  
# search\_key\_value should be dictionary  
# returns the number of documents which were deleted  
def delete\_many(self, search\_key\_value):  
 if search\_key\_value is not None: # if there was a search value provided  
 # Delete the files and return the number of files deleted  
 return self.database.animals.delete\_many(search\_key\_value).deleted\_count   
 else: # Else there was a problem  
 raise Exception("DELETE ERROR: Nothing to delete, because search\_key\_value parameter is empty")

These two functions will delete documents. Like read(), it they take a search parameter. Using delete\_one() will cause the first located document which matches the search parameter will be deleted. Similarly, delete\_many() will delete all matching documents. The returned value will be the number of documents which were deleted.

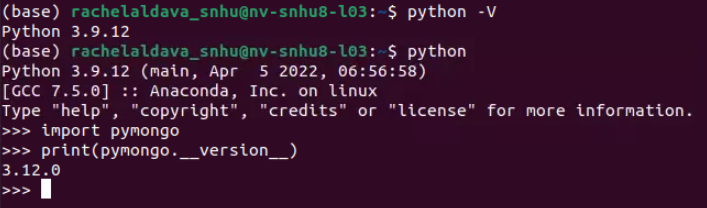
Reproduction and challenges:

If one were to reproduce or write a similar code, one should consult the pymongo documentation: <https://pymongo.readthedocs.io/en/stable/> , which contains a comprehensive breakdown of the functionality of pymongo as well as offering an excellent tutorial section.

## Installation

1. Download AnimalCRUD.py and place it in the working directory of your project. If you will be using an IDE, such as PyCharm or Jupyter, you should ensure it is installed properly as well. If you are not using an IDE, you should ensure that you have an up-to-date version of Python installed. This module may not be compatible with older versions of python 3 and WILL NOT be compatible with any python 2 installation. This project was mostly coded in a text-editor and executed within Jupyter.
2. this module has two dependencies: pymongo and bson.objectid. You must install these prior to using this module. If you are using an IDE, please reference the module installation documentation which is specific to your IDE. If you are running python directly, you may run “pip install pymongo” and “pip install bson.objectid”. If you are running PyCharm, then you would go to settings -> Python interpreter –> (hit the “+” symbol above the list of packages) -> type in “pymongo” -> click “pymongo” -> click install. Repeat these steps for bson.objectid.

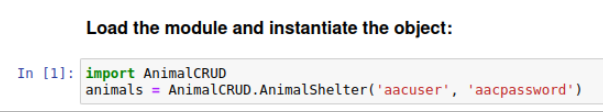
You can check your current python version by typing “python -V” into your console. To check pymongo version, you can enter the python shell and type “import pymongo”, then “print(pymongo.\_\_version\_\_)”.

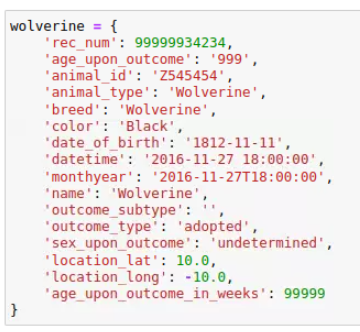
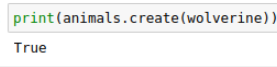


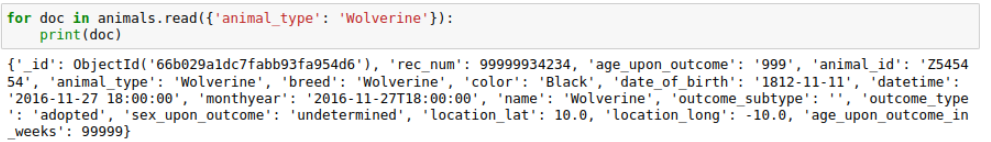
1. You will also need to have access to a server hosting the Austin Animal Care Mongo database. Installing and setting up this database is beyond the scope of this project.

## Usage

### Code Example

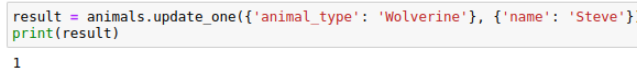
* Open up your preferred python development environment (here we will use Jupyter)
* You will need to import the module and instantiate the object (remember to place AnimalCRUD.py in the project folder):  
  here, we are telling the script to load the AnimalCRUD module from the working directory (For more information, see the installation section)  
    
  We are then instantiating an object named “animal” which we will use to call our CRUD functions. For parameters, we are using the username “aacuser” and “aacpassword”. The constructor will result in an error if the connection is not successful; for troubleshooting, please reference the “installation” and “getting started” sections.
* To create a document, we need some data. A database document will have the same structure as a Pythonic dictionary, which has a series of key:value pairs. For more information, you can reference <https://www.w3schools.com/python/python_dictionaries.asp> . Here is the dictionary we will be working with:

**  
Then, inserting the document into the database is as simple as:  
Where we are using our CRUD object called “animals” to call the create() function, which we are passing the document we named “wolverine”. Finally, we print() the results. The “True” result indicates that we successfully inserted the document. Note: You do not need to include the print() portion in order to successfully insert a document.

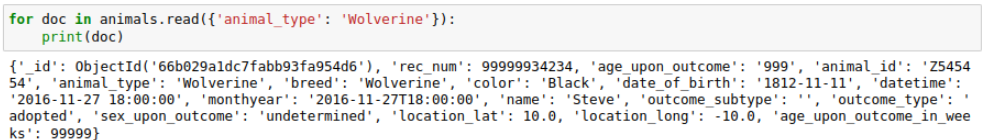
* To read a document from the database, you need to formulate some search parameters. The read() function will accept a dictionary as input and attempt to find any document with that key-value pair. A blank dictionary {} would result in all documents in the database being read. In a previous section we created a document for a wolverine. We can read it like so: 

Note that what read() returns is a python list which contains dictionaries. You can read about python lists here: <https://www.w3schools.com/python/python_lists.asp>. In the above code we are iterating through each document within the list, temporarily naming the current document as “doc”, then calling the print() function with the dictionary doc as the parameter. You could manipulate “doc” any way you would a dictionary, however your changes would not be saved unless you were to use an update function (which we will discuss next)

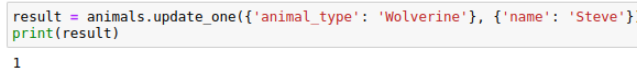
* There are two update functions: update\_one() and update\_many(); they will update one document or multiple documents, respectively. To modify a document, you will need a dictionary to serve as a search parameter, just like how we use the read() function. This time, however, we will also need whatever data we will be updating with. This will also be a python dictionary. The update methods will use the search parameter to find the documents, then will add the data dictionary to the existing document. If the document already has the same “key” value, then the “value” for that key will be updated. For example, we can search for an animal which is a wolverine and update the name to “Steve”:



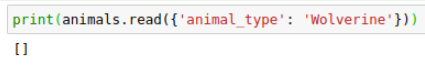
The “1” indicates that one document was updated. The update\_one() function found the first wolverine it could find, updated it, then stopped. If update\_many() were called, using the same values, it would search the entire database and rename any wolverine it could find. We can call the read method again to verify the change:



* There are two delete functions: delete\_one() and delete\_many(); they will delete one document or multiple documents, respectively. To delete a document, you will need a dictionary to serve as a search parameter, just like how we use the read() function. The update methods will use the search parameter to find the documents and delete them. For example, we can search for an animal which is a wolverine and delete it.



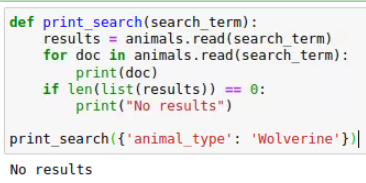
The “1” indicates that one document was deleted. The delete\_one() function found the first wolverine it could find, deleted it, then stopped. If delete\_many() were called, using the same values, it would search the entire database and delete any wolverine it could find. We can call the read method again to verify the change:



As we can see, it returns an empty list, which means where are no more wolverines. Note that the print statement was slightly changed in order to show that the list is empty.

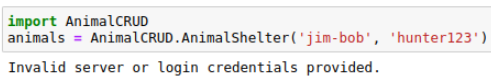
### Tests

Before we dig in to trouble-shooting, we can take some time to speed up the testing process by defining a print method which can improve clarity:

  
Here, we are defining a new function called “print\_search” which will print out the output of a read() command, except when the list is empty, in which case it will print No results”. As we can see, there are currently no wolverines in the database.

* **Testing instantiation and server connection:**

If the instructions are followed properly, and the hard-coded connection info such as port number, database, and collection are configured properly, a user could run into an error with sign-on details. In the event of such an error, the following message should appear:



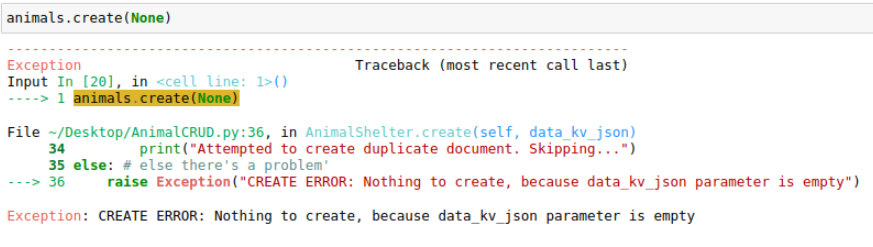
Let’s take a look at the AnimalCRUD constructor again:

def \_\_init\_\_(self, USER, PASS):  
 # Hard-coded database details - Edit these values where appropriate  
 # Host and port number are needed to connect with anything on the internet. For common types of traffic, such as loading a web page, the port numbers are standardized. For our application, you will need to verify the port your host is running on.  
 HOST = 'nv-desktop-services.apporto.com'  
 PORT = 32760  
 DB = 'AAC' # The name of the database you are connecting to  
 COL = 'animals' # The name of the collection you will be working with  
 try:  
 # Initialize Connection  
 self.client = MongoClient('mongodb://%s:%s@%s:%d' % (USER, PASS, HOST, PORT)) # connect to the host  
 self.database = self.client['%s' % (DB)] # Select the database  
 self.collection = self.database['%s' % (COL)] # Select the collection.  
  
 # Check connection  
 self.database.animals.find\_one()  
 except Exception:  
 print("Invalid server or login credentials provided.")

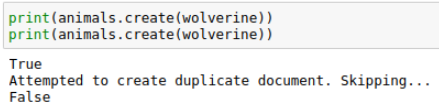
This error could be caused by poor information provided, such as username, password, port number, or host address. It could also be caused by the server being offline.

If this error message does not appear, it means that you have successfully connected and read one document using the find\_one() command through pymongo.

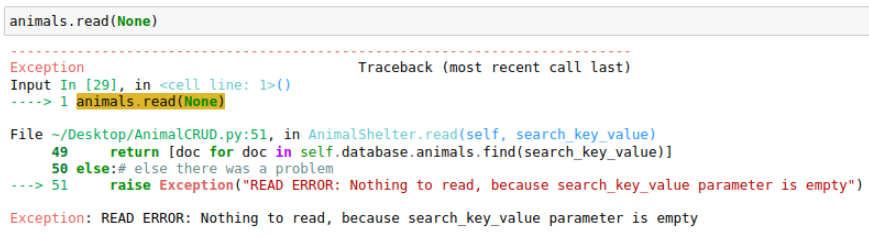
* **Testing create():**
  + 1. When we call create with a parameter value of None, it should throw an exception:



* + 1. When trying to insert two identical documents, it would normally cause a collision. AnimalCRUD should prevent this by detecting the error and skipping the second insertion:

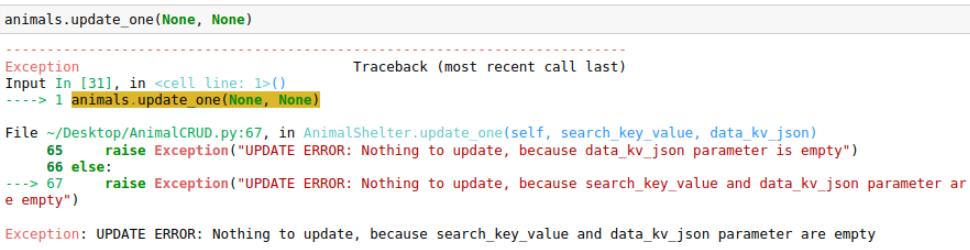
  
The first line “True” indicates that the first insertion was successful, the third line “False” indicates that the second insertion was not successful.

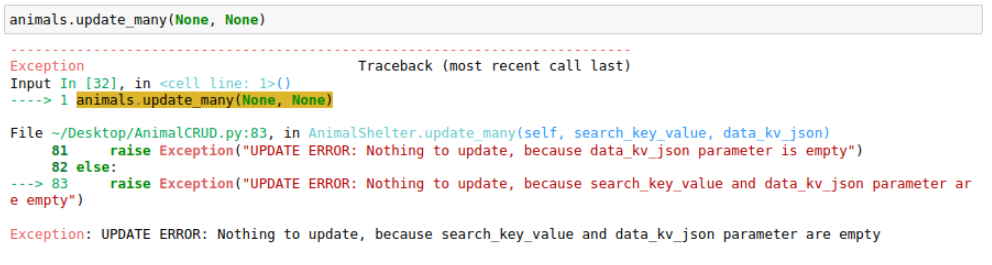
* **Testing read():**

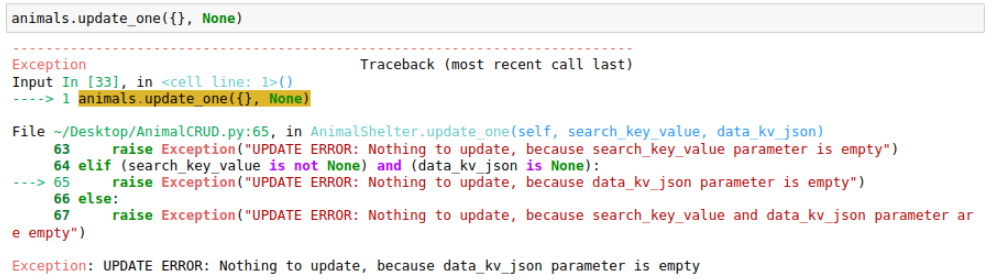
1. When we call read with a parameter value of None, it should throw an exception: 
2. When a searching for something which does not exist in the database, it should return an empty list:  
    

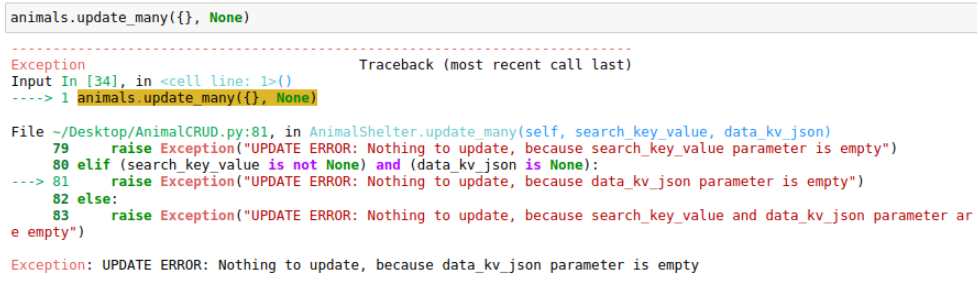
* Testing update\_one and update\_many():

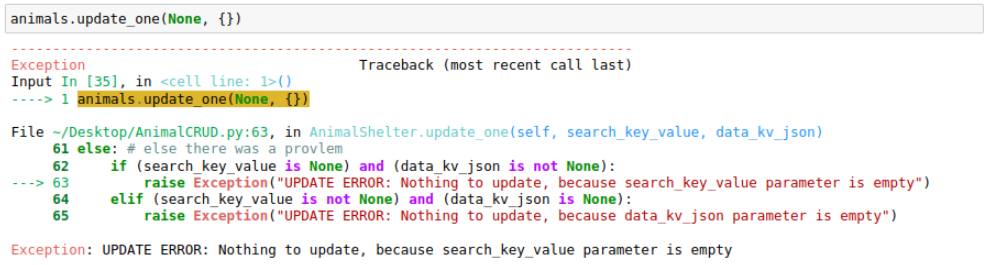
1. When we call an update function with any parameter value of None, it should throw an exception:

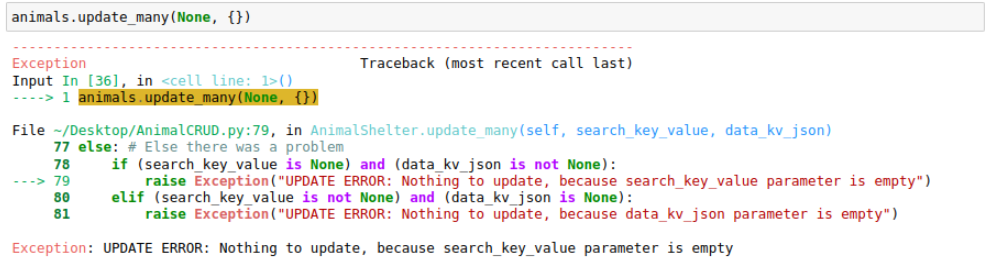
**Case: update\_one(None, None):**

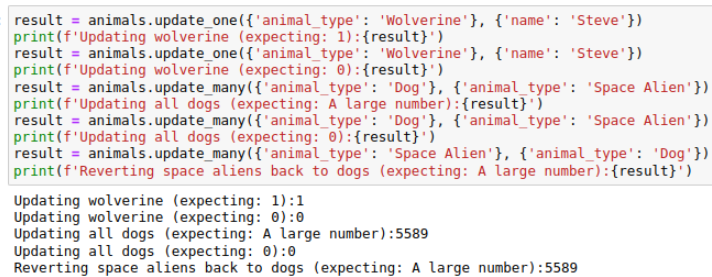
**Case: update\_many(None, None):  
**

**Case: update\_one({some data}, None):  
**

**Case: update\_many({some data}, None):  
**

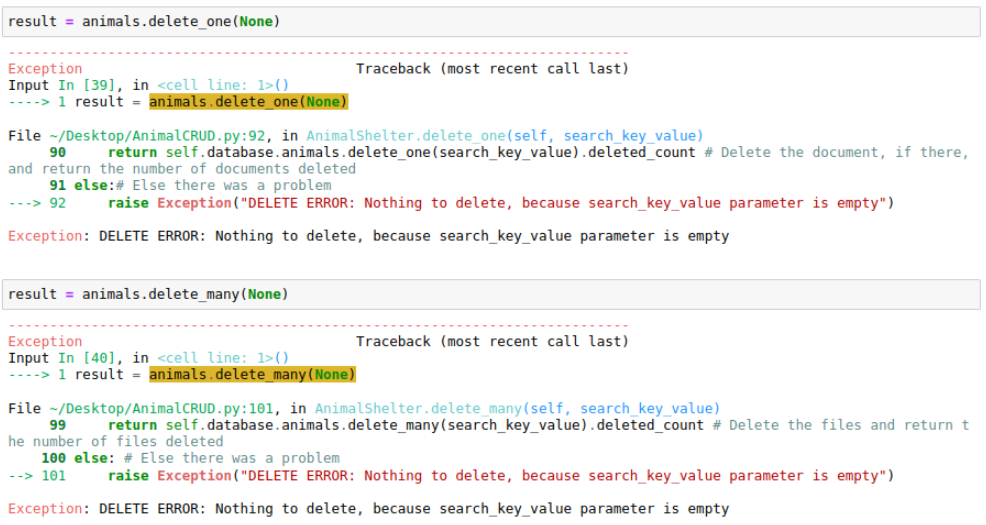
**Case: update\_one(None, {some data}):  
**

**Case: update\_many(None, {some data}):**

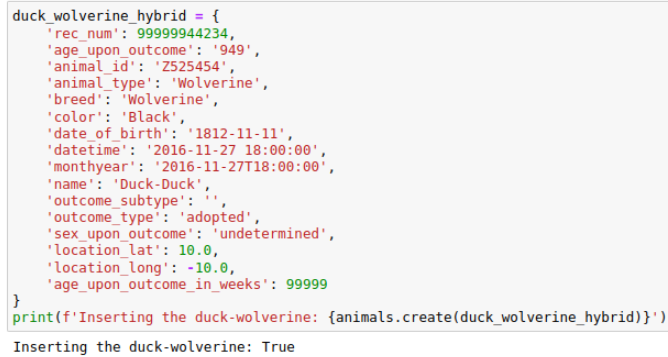
1. We should expect that running identical update statements twice will result in 0 updates for the second run:  
   

So first, we named the first wolverine we found “Steve” using update\_one(), which as successful. Next, we tried to name the same wolverine “Steve” using update\_one(), but no changes were made because its name was already “Steve”. Then, to test update\_many(), we changed all dogs to space aliens, which turned out to be 5589 dogs. We tried again to change all dogs to space aliens, but we found no dogs, so we updated 0. Finally, we changed all the space aliens back to dogs because we don’t want to mess up our database.

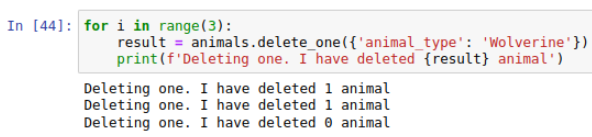
* Testing delete\_one and delete\_many():

1. When we call a delete function with a parameter value of None, it should throw an exception:  
   
2. When we call a delete function, we expect to eventually run out of documents to delete:

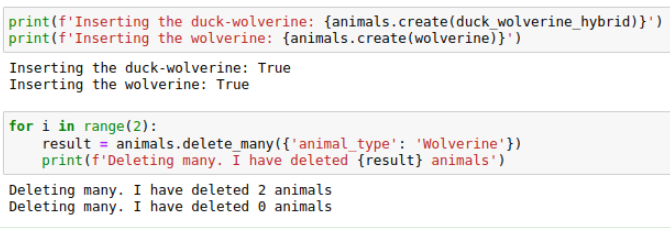
We already have one wolverine in the database right now. If you don’t, you can just insert it again using the previous instructions. Let’s create another document the same way we did previously:



At this point we have two wolverines. Let’s test how many delete\_one() commands it takes to run out of wolverines:



Looping through the delete command three times shows us that we deleted successfully on the first two times, but failed to delete on the second time, as expected.  
  
Now, let’s reset the wolverines and test delete\_many():



As expected, we were able to delete all the wolverines on the first try.

### Screenshots

Screenshots, and non-image code samples, have been featured throughout this document.

## Roadmap/Features (Optional)

There are a few features which could be added in order to improve the user experience:

* Bulk creation: The ability to import a large number of documents at once rather than individually. This could also take the form of reading from a csv file.
* Single read: Currently the read function returns a list of results rather than the first result. There may be situations where finding the first result is preferable.
* More robust initial connection exception handling. Right now, the code simply checks if it can successfully run a read\_one() command. If it can’t, it assumes that something is wrong with the connection. It would be useful to end-users to know whether the problem is with the connection settings, the authentication settings, or some other issue which prevents a simple read\_one() command.
* Server dropped connection error handling: Due to the database and workstation being externally hosted, we have not had the chance to develop solutions to errors involving server communication.
* Error handling involving huge create/read/update/delete jobs
* Index creation/handling: in order to improve performance, it may be possible to create and manage indexes from within the AnimalCRUD.py module. However, due to how they affect the design decisions for the database, any feature would need to be heavily tested in order to ensure proper functionality and security.

## Contact

Rachel Aldava

https://github.com/RachelAldava

rachel@aldava.net