CS 340 Project One
GCZ79
11/28/2025

About the Project
This project focuses on developing the CRUD functionalities (Create, Read, Update, and Delete) of a Python module for interacting with the Austin Animal Center (AAC) MongoDB database.
This module enables users to connect to a MongoDB database, insert new records, query existing records using key-value pairs, update existing records, and delete records.
The project is divided into two files:

- CRUD_Python_Module.py = CRUD operations module
- ProjectOneTestScript.ipynb= Testing script

Motivation
Grazioso Salvare, an innovative international rescue-animal training company, requires a software application that enables them to interact with the existing databases of five animal shelters in Austin, Texas, to identify dogs suitable for search-and-rescue training.
Global Rain, the software engineering company that employs me, has contracted for a full-stack development application, and the present document has been produced to accompany the CRUD Python module.

Getting Started
This module provides a simple interface to perform CRUD (Create, Read, Update, and Delete) operations on the Austin Animal Center (AAC) MongoDB database. Users can import the module into a Python script or Jupyter Notebook and instantiate the AnimalShelter class with the provided credentials to interact with the database.

- Database name: aac
- Collection name: animals
- User account: aacuser
- Password: NoSQLNoParty

The user aacuser has read and write permissions to the aac.animals collection.

The connection URI follows this structure: mongodb://aacuser:NoSQLNoParty@localhost:27017

This module uses pymongo, the official Python driver for MongoDB.

pymongo was chosen because it is officially supported and well-documented, and it provides a Pythonic API for MongoDB operations that supports authentication, query operators, and connection pooling, which are necessary for connecting to the AAC database securely and efficiently.

The CRUD module is implemented in a Python class named AnimalShelter, and its methods are:

Create: create()

- Validates that the input is a dictionary
- Inserts a document using insert_one()
- Returns True if successful, otherwise False
- Handles errors using try/except blocks

Read: read()

- Uses MongoDB's find() method to query the database
- Accepts a filter dictionary such as {"species": "cat"}
- Returns a list of the resulting documents
- Returns an empty list if there is no match and handles errors

Update: update()

- Validates that both query and new_values are dictionaries
- Uses MongoDB's update_many() to modify all documents matching the query
- Requires update operators such as {"$set": {...}}
- Returns the number of documents successfully modified
- Handles errors with try/except and returns 0 on failure

Delete: delete()

- Validates that the query parameter is a dictionary
- Uses MongoDB's delete_many() to remove documents matching the filter
- Returns the number of documents successfully deleted
- Handles errors with try/except and returns 0 on failure

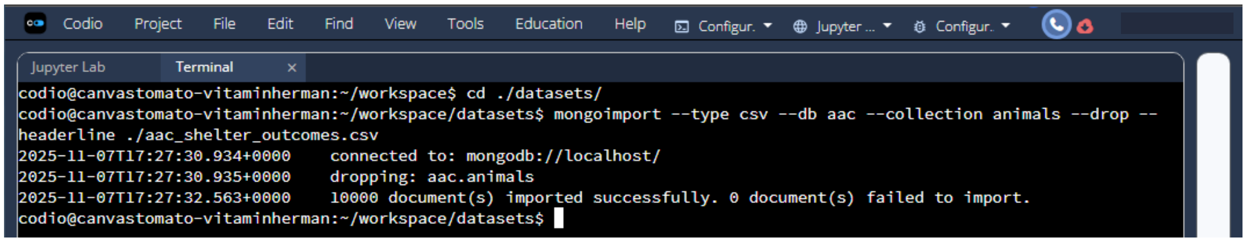
Installation

This project uses the following tools and libraries:

- MongoDB
  The project has been created in the cloud-based platform Codio, but it can also run locally by connecting to the cloud-hosted MongoDB Atlas database.
- Python 3.x
  The primary language for developing the CRUD module, it can be downloaded from here.
- Pymongo Library
  Used to connect Python to MongoDB, this library provides the MongoClient class and database/collection methods such as insert_one() and find().

MongoDB Import execution

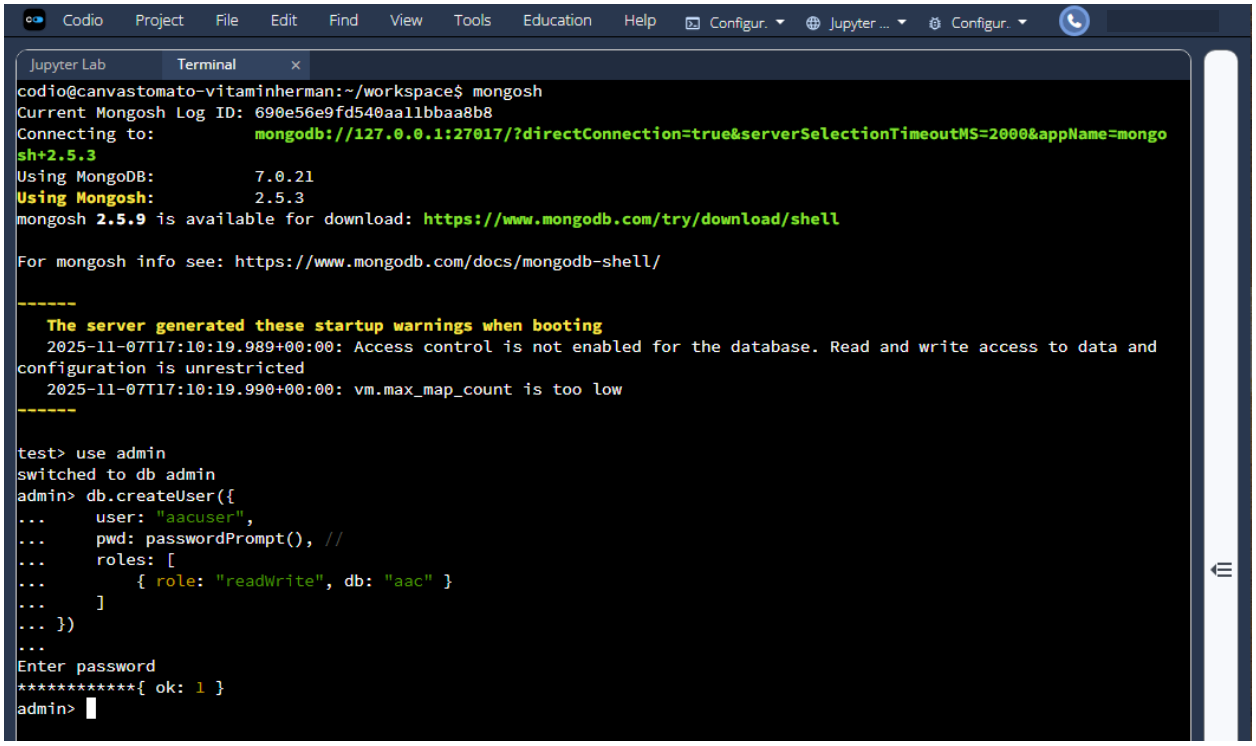
- cd ./datasets # move into folder with datasets
- mongoimport --type=csv --headerline --db aac --collection animals --drop ./aac_shelter_outcomes.csv
  # import CSV into MongoDB (specify type, use first line of CSV as a key for the documents, name of DB
  # and collection, drop existing animals collection before importing new data, path to CSV file)



```
codio@canvastomato-vitaminherman:~/workspace$ cd ./datasets/
codio@canvastomato-vitaminherman:~/workspace/datasets$ mongoimport --type csv --db aac --collection animals --drop --
headerline ./aac_shelter_outcomes.csv
2025-11-07T17:27:30.934+0000    connected to: mongodb://localhost/
2025-11-07T17:27:30.935+0000    dropping: aac.animals
2025-11-07T17:27:32.563+0000    10000 document(s) imported successfully. 0 document(s) failed to import.
codio@canvastomato-vitaminherman:~/workspace/datasets$
```

Creation of "aacuser" with read and write privileges to "aac"

- mongosh # start MongoDB shell
- use admin # create the "aacuser" under "admin" database (for centralized user management)
- db.createUser({
    user: "aacuser",
    pwd: passwordPrompt(), # Password: NoSQLNoParty
    roles: [
      { role: "readWrite", db: "aac" }
    ]
  })

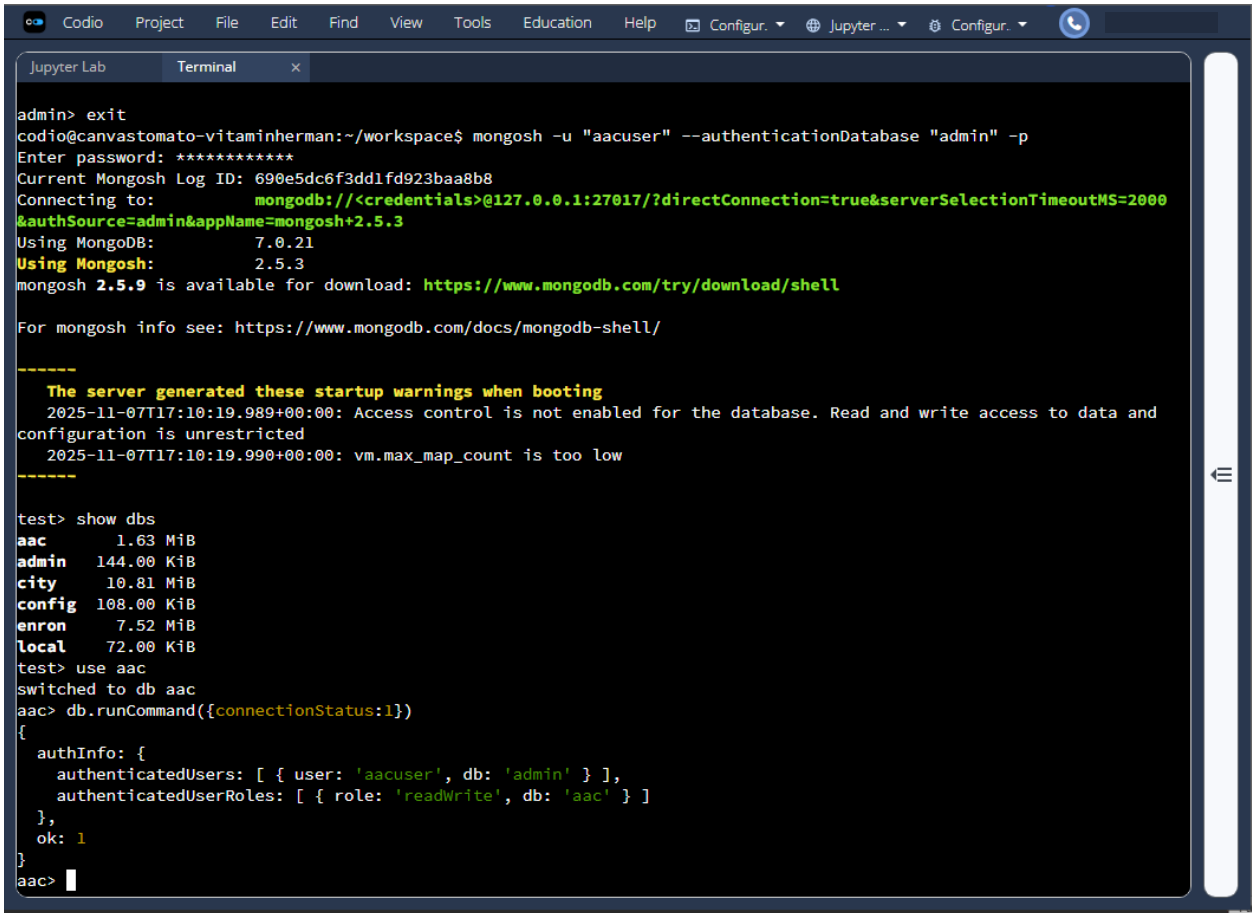


```
codio@canvastomato-vitaminherman:~/workspace$ mongosh
Current Mongosh Log ID: 690e56e9fd540aa11bbaa8b8
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongo
sh+2.5.3
Using MongoDB:          7.0.21
Using Mongosh:          2.5.3
mongosh 2.5.9 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-11-07T17:10:19.989+00:00: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
   2025-11-07T17:10:19.990+00:00: vm.max_map_count is too low
------

test> use admin
switched to db admin
admin> db.createUser({
...     user: "aacuser",
...     pwd: passwordPrompt(), //
...     roles: [
...         { role: "readWrite", db: "aac" }
...     ]
... })
...
Enter password
************{ ok: 1 }
admin>
```

Note: MongoDB does not support switching authenticated users within the same shell session. After creating a new user, you must exit the current session and reconnect using the new user's credentials.

User "aacuser" authentication to "aac" database

- mongosh -u "aacuser" --authenticationDatabase "admin" –p # login as "aacuser" (pw: NoSQLNoParty)
  show dbs # optional: check available databases
- use aac # switch to "aac" database
  db.runCommand({connectionStatus:1}) # optional: verify connected user and their roles



```
admin> exit
codio@canvastomato-vitaminherman:~/workspace$ mongosh -u "aacuser" --authenticationDatabase "admin" -p
Enter password: ************
Current Mongosh Log ID: 690e5dc6f3dd1fd923baa8b8
Connecting to:          mongodb://<credentials>@127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
&authSource=admin&appName=mongosh+2.5.3
Using MongoDB:          7.0.21
Using Mongosh:          2.5.3
mongosh 2.5.9 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-11-07T17:10:19.989+00:00: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
   2025-11-07T17:10:19.990+00:00: vm.max_map_count is too low
------

test> show dbs
aac       1.63 MiB
admin   144.00 KiB
city     10.81 MiB
config  108.00 KiB
enron     7.52 MiB
local    72.00 KiB
test> use aac
switched to db aac
aac> db.runCommand({connectionStatus:1})
{
  authInfo: {
    authenticatedUsers: [ { user: 'aacuser', db: 'admin' } ],
    authenticatedUserRoles: [ { role: 'readWrite', db: 'aac' } ]
  },
  ok: 1
}
aac> 
```

Southern New Hampshire University

Usage

1. Import the module into a separate Python script or Jupyter Notebook

   ```
   from CRUD_Python_Module import AnimalShelter
   ```

2. Instantiate the class with the required AAC username and password

   ```
   shelter = AnimalShelter('aacuser', 'NoSQLNoParty')
   ```

3. Profit! Now you have access to the four CRUD operations that can be used like this:

```
# CREATE

doc = {"animal_id": "TEST123", "name": "Mr Whiskers"}

shelter.create(doc)


# READ

results = shelter.read({"name": "Mr Whiskers"})

print(results)


# UPDATE

shelter.update({"name": "Mr Whiskers"}, {"$set": {"name": "Mr Whiskers II The Great"}})


# DELETE

shelter.delete({"animal_id": "TEST123"})
```

Additional demonstrations of the module's functionality are provided in the test script ProjectOneTestScript.ipynb included with this project.
This script shows complete usage patterns for all four CRUD operations and can be used as a reference when interacting with the AnimalShelter class.
Several screenshots on the following pages further illustrate these operations and their outputs.

Southern New Hampshire University

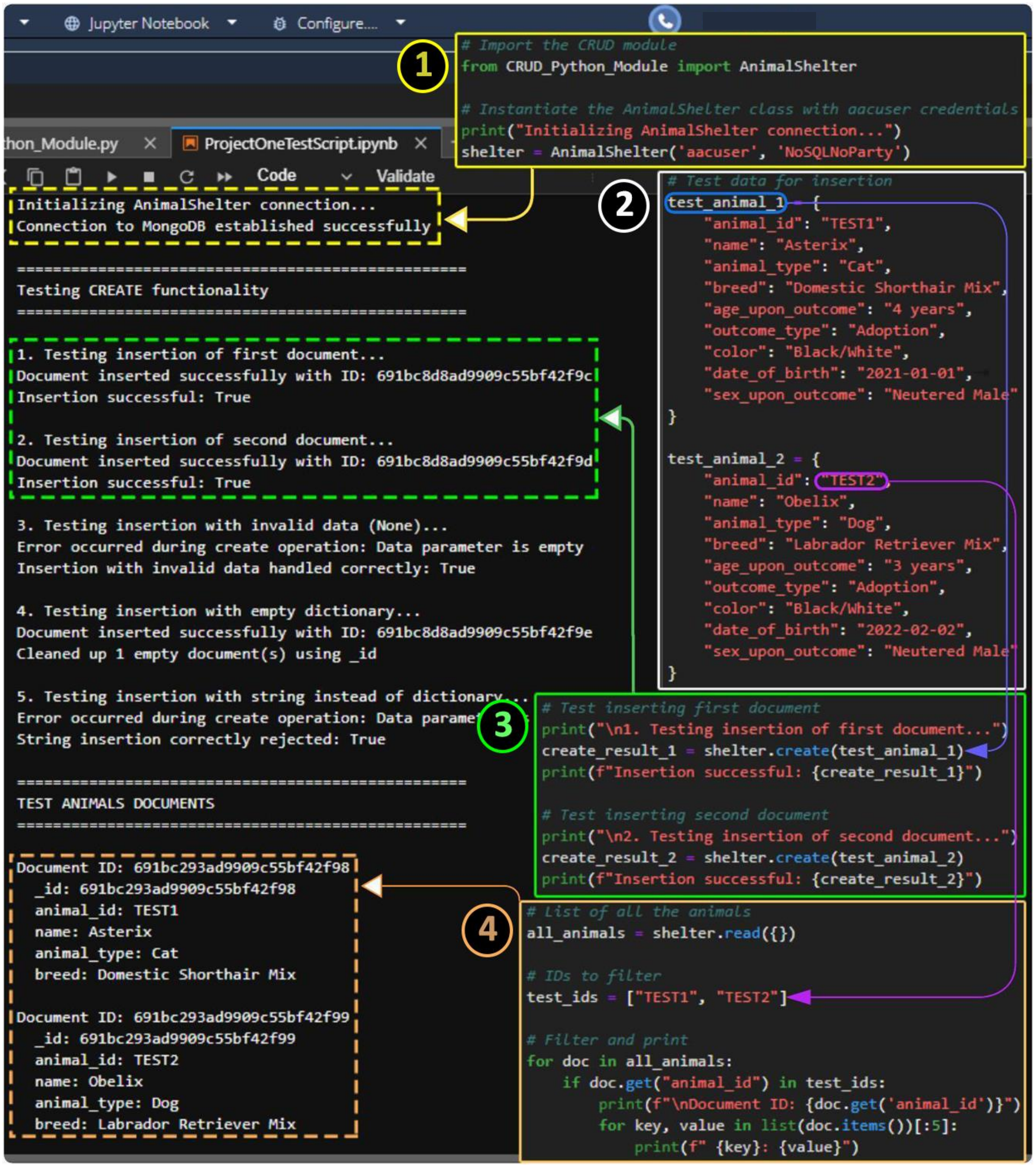
CRUD functionality test execution

(1) Imports the AnimalShelter CRUD class and then creates an instance using the aacuser credentials

(2) Defines two sample animal documents to be inserted into the database

(3) create(): Inserts both test documents into the database and prints if the insertion was successful

(4) Retrieves all records, filters for the two test IDs, and prints key details from the matching documents



```
# Import the CRUD module
from CRUD_Python_Module import AnimalShelter

# Instantiate the AnimalShelter class with aacuser credentials
print("Initializing AnimalShelter connection...")
shelter = AnimalShelter('aacuser', 'NoSQLNoParty')
```

```
Initializing AnimalShelter connection...
Connection to MongoDB established successfully

==================================================
Testing CREATE functionality
==================================================

1. Testing insertion of first document...
Document inserted successfully with ID: 691bc8d8ad9909c55bf42f9c
Insertion successful: True

2. Testing insertion of second document...
Document inserted successfully with ID: 691bc8d8ad9909c55bf42f9d
Insertion successful: True

3. Testing insertion with invalid data (None)...
Error occurred during create operation: Data parameter is empty
Insertion with invalid data handled correctly: True

4. Testing insertion with empty dictionary...
Document inserted successfully with ID: 691bc8d8ad9909c55bf42f9e
Cleaned up 1 empty document(s) using _id

5. Testing insertion with string instead of dictionary...
Error occurred during create operation: Data parame
String insertion correctly rejected: True

==================================================
TEST ANIMALS DOCUMENTS
==================================================

Document ID: 691bc293ad9909c55bf42f98
  _id: 691bc293ad9909c55bf42f98
  animal_id: TEST1
  name: Asterix
  animal_type: Cat
  breed: Domestic Shorthair Mix

Document ID: 691bc293ad9909c55bf42f99
  _id: 691bc293ad9909c55bf42f99
  animal_id: TEST2
  name: Obelix
  animal_type: Dog
  breed: Labrador Retriever Mix
```

```
# Test data for insertion
test_animal_1 = {
    "animal_id": "TEST1",
    "name": "Asterix",
    "animal_type": "Cat",
    "breed": "Domestic Shorthair Mix",
    "age_upon_outcome": "4 years",
    "outcome_type": "Adoption",
    "color": "Black/White",
    "date_of_birth": "2021-01-01",
    "sex_upon_outcome": "Neutered Male"
}

test_animal_2 = {
    "animal_id": "TEST2",
    "name": "Obelix",
    "animal_type": "Dog",
    "breed": "Labrador Retriever Mix",
    "age_upon_outcome": "3 years",
    "outcome_type": "Adoption",
    "color": "Black/White",
    "date_of_birth": "2022-02-02",
    "sex_upon_outcome": "Neutered Male"
}
```

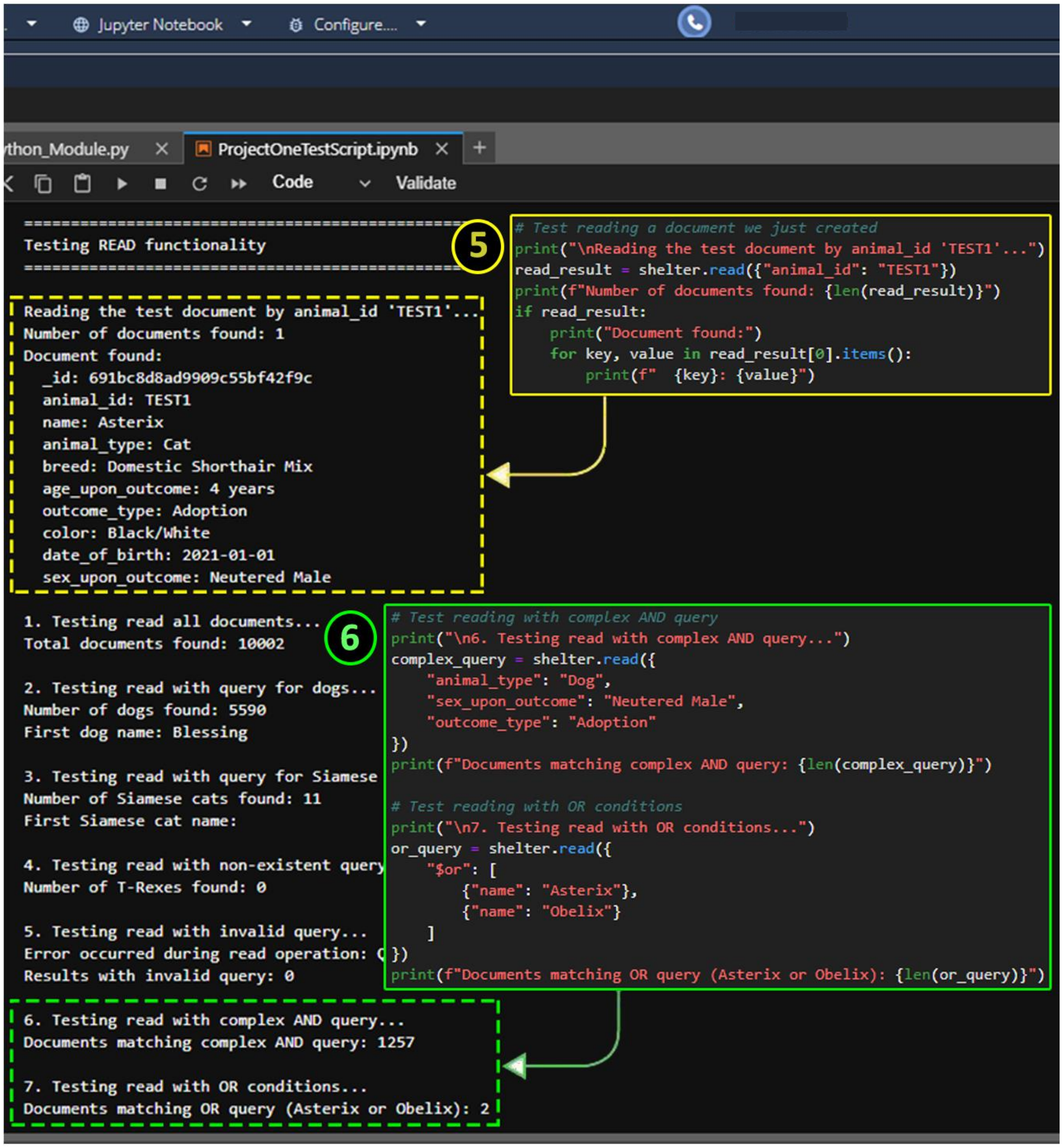
```
# Test inserting first document
print("\n1. Testing insertion of first document...")
create_result_1 = shelter.create(test_animal_1)
print(f"Insertion successful: {create_result_1}")

# Test inserting second document
print("\n2. Testing insertion of second document...")
create_result_2 = shelter.create(test_animal_2)
print(f"Insertion successful: {create_result_2}")
```

```
# List of all the animals
all_animals = shelter.read({})

# IDs to filter
test_ids = ["TEST1", "TEST2"]

# Filter and print
for doc in all_animals:
    if doc.get("animal_id") in test_ids:
        print(f"\nDocument ID: {doc.get('animal_id')}")
        for key, value in list(doc.items())[:5]:
            print(f"  {key}: {value}")
```

(5) read(): Reads the document with animal_id = "TEST1" from the database and prints all of its fields if found

(6) read(): Searches for documents matching all specified fields (animal_type, sex_upon_outcome, outcome_type) and prints how many were found

read(): Reads all documents where the name is "Asterix" or "Obelix" using a MongoDB $or condition



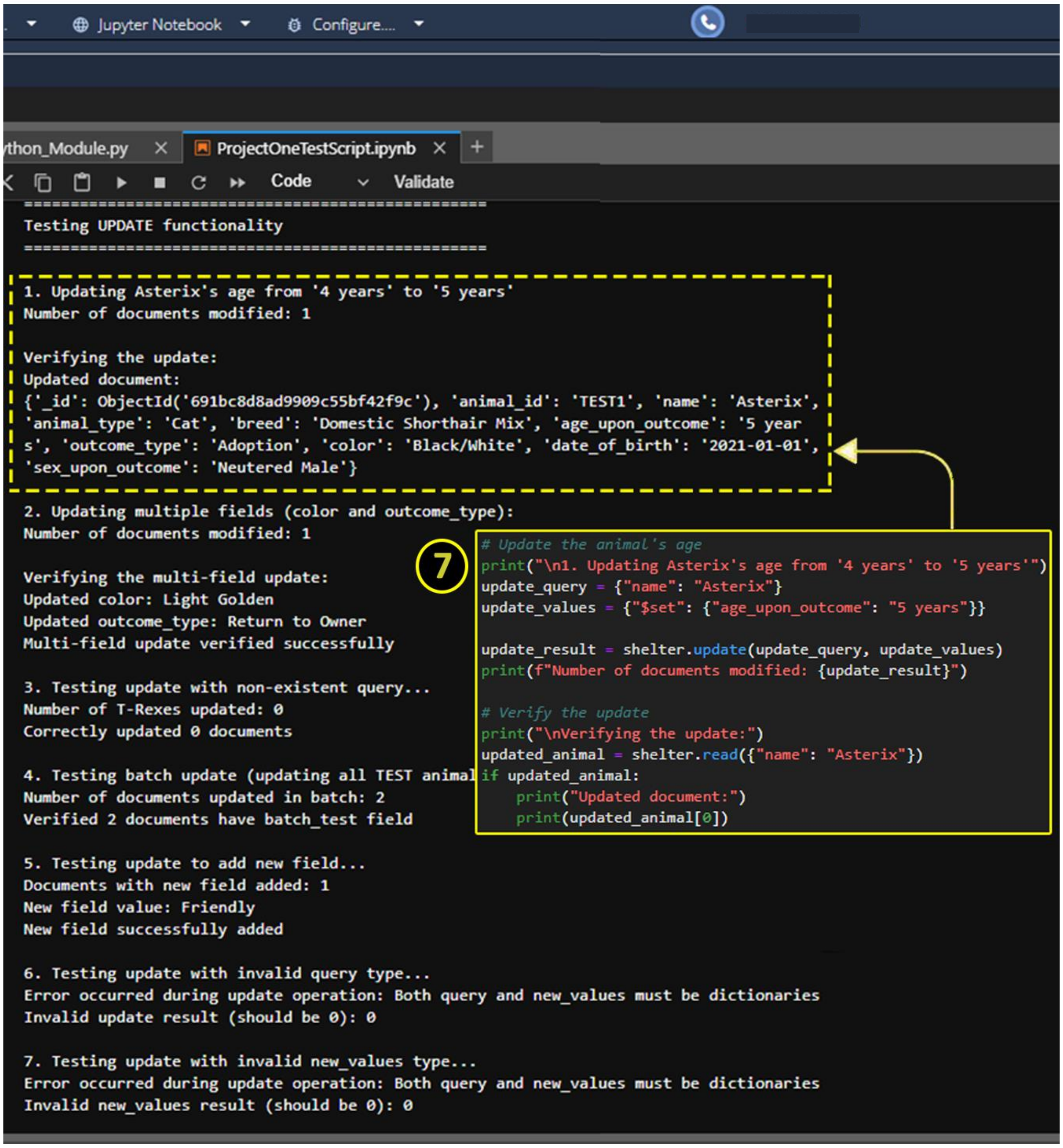
```
# Test reading a document we just created
print("\nReading the test document by animal_id 'TEST1'...")
read_result = shelter.read({"animal_id": "TEST1"})
print(f"Number of documents found: {len(read_result)}")
if read_result:
    print("Document found:")
    for key, value in read_result[0].items():
        print(f"  {key}: {value}")
```

```
================================================
Testing READ functionality
================================================

Reading the test document by animal_id 'TEST1'...
Number of documents found: 1
Document found:
  _id: 691bc8d8ad9909c55bf42f9c
  animal_id: TEST1
  name: Asterix
  animal_type: Cat
  breed: Domestic Shorthair Mix
  age_upon_outcome: 4 years
  outcome_type: Adoption
  color: Black/White
  date_of_birth: 2021-01-01
  sex_upon_outcome: Neutered Male

1. Testing read all documents...
Total documents found: 10002

2. Testing read with query for dogs...
Number of dogs found: 5590
First dog name: Blessing

3. Testing read with query for Siamese
Number of Siamese cats found: 11
First Siamese cat name:

4. Testing read with non-existent query
Number of T-Rexes found: 0

5. Testing read with invalid query...
Error occurred during read operation: (
Results with invalid query: 0
```

```
# Test reading with complex AND query
print("\n6. Testing read with complex AND query...")
complex_query = shelter.read({
    "animal_type": "Dog",
    "sex_upon_outcome": "Neutered Male",
    "outcome_type": "Adoption"
})
print(f"Documents matching complex AND query: {len(complex_query)}")

# Test reading with OR conditions
print("\n7. Testing read with OR conditions...")
or_query = shelter.read({
    "$or": [
        {"name": "Asterix"},
        {"name": "Obelix"}
    ]
})
print(f"Documents matching OR query (Asterix or Obelix): {len(or_query)}")
```

```
6. Testing read with complex AND query...
Documents matching complex AND query: 1257

7. Testing read with OR conditions...
Documents matching OR query (Asterix or Obelix): 2
```

(7) update(): Updates Asterix's age_upon_outcome from "4 years" to "5 years", then reads the document again to verify that the change was successfully applied

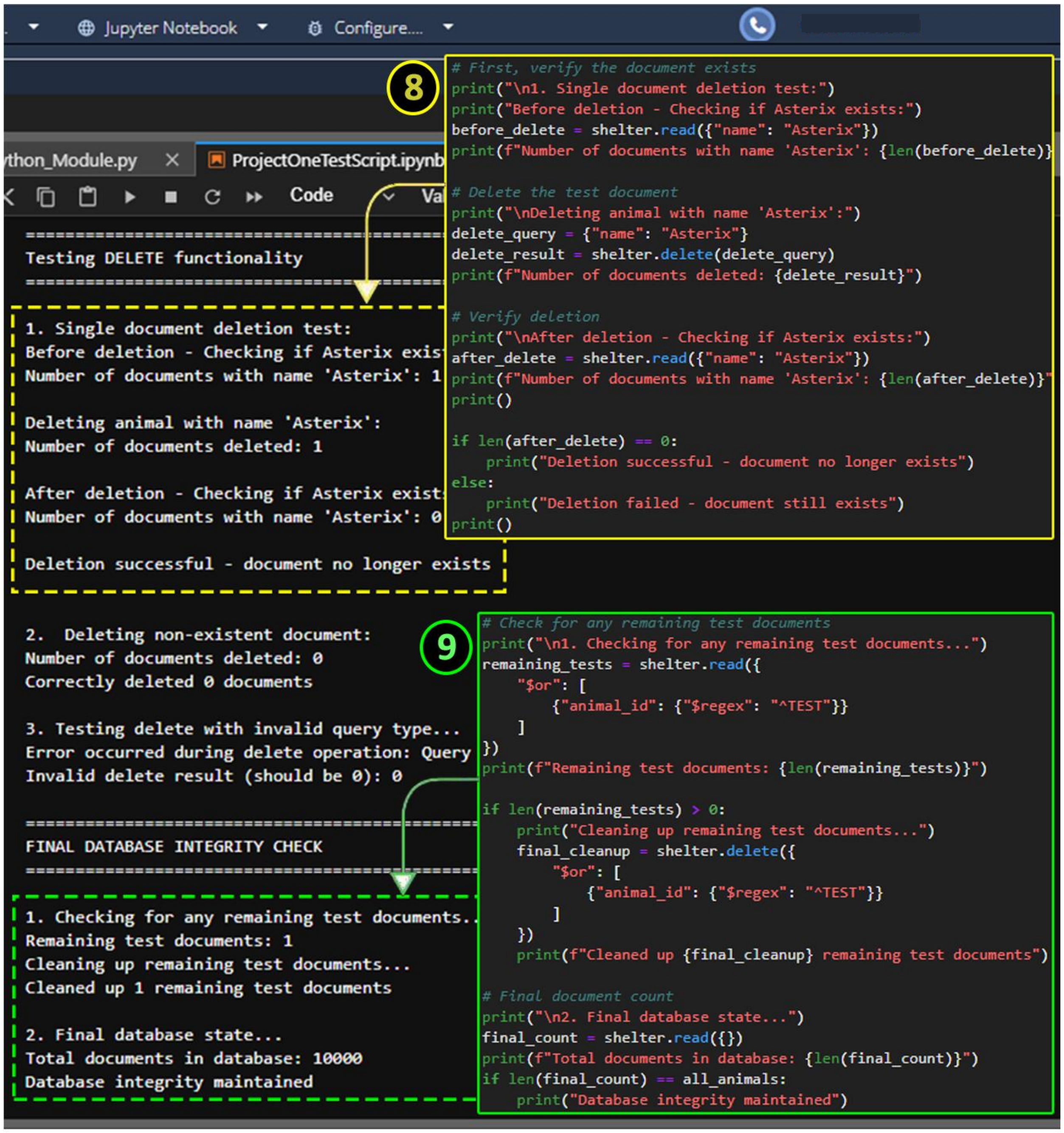


```
==================================================
Testing UPDATE functionality
==================================================

1. Updating Asterix's age from '4 years' to '5 years'
Number of documents modified: 1

Verifying the update:
Updated document:
{'_id': ObjectId('691bc8d8ad9909c55bf42f9c'), 'animal_id': 'TEST1', 'name': 'Asterix',
'animal_type': 'Cat', 'breed': 'Domestic Shorthair Mix', 'age_upon_outcome': '5 year
s', 'outcome_type': 'Adoption', 'color': 'Black/White', 'date_of_birth': '2021-01-01',
'sex_upon_outcome': 'Neutered Male'}

2. Updating multiple fields (color and outcome_type):
Number of documents modified: 1

Verifying the multi-field update:
Updated color: Light Golden
Updated outcome_type: Return to Owner
Multi-field update verified successfully

3. Testing update with non-existent query...
Number of T-Rexes updated: 0
Correctly updated 0 documents

4. Testing batch update (updating all TEST animal
Number of documents updated in batch: 2
Verified 2 documents have batch_test field

5. Testing update to add new field...
Documents with new field added: 1
New field value: Friendly
New field successfully added

6. Testing update with invalid query type...
Error occurred during update operation: Both query and new_values must be dictionaries
Invalid update result (should be 0): 0

7. Testing update with invalid new_values type...
Error occurred during update operation: Both query and new_values must be dictionaries
Invalid new_values result (should be 0): 0
```

```python
# Update the animal's age
print("\n1. Updating Asterix's age from '4 years' to '5 years'")
update_query = {"name": "Asterix"}
update_values = {"$set": {"age_upon_outcome": "5 years"}}

update_result = shelter.update(update_query, update_values)
print(f"Number of documents modified: {update_result}")

# Verify the update
print("\nVerifying the update:")
updated_animal = shelter.read({"name": "Asterix"})
if updated_animal:
    print("Updated document:")
    print(updated_animal[0])
```

(8) delete(): Single-document deletion test. Checks that the "Asterix" record exists, deletes it using a name-based query, and then verifies that the document was successfully removed.

(9) Cleanup of remaining test documents. Searches for any leftover test records (animal_id starting with "TEST"), deletes them if present, and then prints the final total number of documents to confirm database integrity (it must be equal to the document count "all_animals" we created at the start of the test).

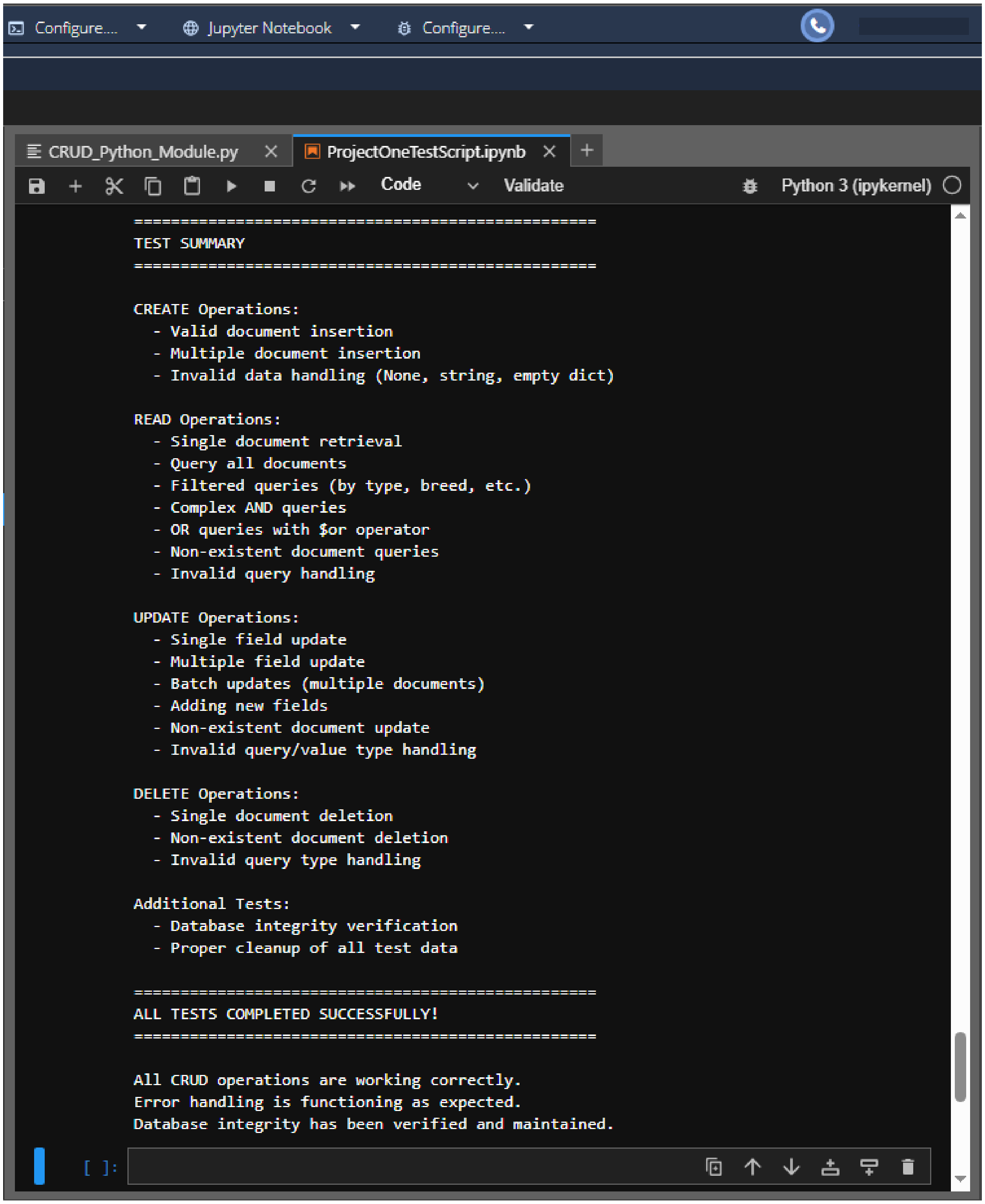


```
# First, verify the document exists
print("\n1. Single document deletion test:")
print("Before deletion - Checking if Asterix exists:")
before_delete = shelter.read({"name": "Asterix"})
print(f"Number of documents with name 'Asterix': {len(before_delete)}

# Delete the test document
print("\nDeleting animal with name 'Asterix':")
delete_query = {"name": "Asterix"}
delete_result = shelter.delete(delete_query)
print(f"Number of documents deleted: {delete_result}")

# Verify deletion
print("\nAfter deletion - Checking if Asterix exists:")
after_delete = shelter.read({"name": "Asterix"})
print(f"Number of documents with name 'Asterix': {len(after_delete)}"
print()

if len(after_delete) == 0:
    print("Deletion successful - document no longer exists")
else:
    print("Deletion failed - document still exists")
print()
```

```
=================================================
Testing DELETE functionality
=================================================

1. Single document deletion test:
Before deletion - Checking if Asterix exis
Number of documents with name 'Asterix': 1

Deleting animal with name 'Asterix':
Number of documents deleted: 1

After deletion - Checking if Asterix exist
Number of documents with name 'Asterix': 0

Deletion successful - document no longer exists

2.  Deleting non-existent document:
Number of documents deleted: 0
Correctly deleted 0 documents

3. Testing delete with invalid query type...
Error occurred during delete operation: Query
Invalid delete result (should be 0): 0

=================================================
FINAL DATABASE INTEGRITY CHECK
=================================================

1. Checking for any remaining test documents..
Remaining test documents: 1
Cleaning up remaining test documents...
Cleaned up 1 remaining test documents

2. Final database state...
Total documents in database: 10000
Database integrity maintained
```

```
# Check for any remaining test documents
print("\n1. Checking for any remaining test documents...")
remaining_tests = shelter.read({
    "$or": [
        {"animal_id": {"$regex": "^TEST"}}
    ]
})
print(f"Remaining test documents: {len(remaining_tests)}")

if len(remaining_tests) > 0:
    print("Cleaning up remaining test documents...")
    final_cleanup = shelter.delete({
        "$or": [
            {"animal_id": {"$regex": "^TEST"}}
        ]
    })
    print(f"Cleaned up {final_cleanup} remaining test documents")

# Final document count
print("\n2. Final database state...")
final_count = shelter.read({})
print(f"Total documents in database: {len(final_count)}")
if len(final_count) == all_animals:
    print("Database integrity maintained")
```

The final screen displays a summary of all the tests that were performed.



```
==================================================
TEST SUMMARY
==================================================

CREATE Operations:
  - Valid document insertion
  - Multiple document insertion
  - Invalid data handling (None, string, empty dict)

READ Operations:
  - Single document retrieval
  - Query all documents
  - Filtered queries (by type, breed, etc.)
  - Complex AND queries
  - OR queries with $or operator
  - Non-existent document queries
  - Invalid query handling

UPDATE Operations:
  - Single field update
  - Multiple field update
  - Batch updates (multiple documents)
  - Adding new fields
  - Non-existent document update
  - Invalid query/value type handling

DELETE Operations:
  - Single document deletion
  - Non-existent document deletion
  - Invalid query type handling

Additional Tests:
  - Database integrity verification
  - Proper cleanup of all test data

==================================================
ALL TESTS COMPLETED SUCCESSFULLY!
==================================================

All CRUD operations are working correctly.
Error handling is functioning as expected.
Database integrity has been verified and maintained.
```