MAKE SCHOOL

Databases with MongoDB & PyMongo

WEB 1.1

Agenda

MAKE SCHOOL

- Learning Outcomes
- Warm-Up: Quiz Review
- Document-based Databases
- **BREAK**
- MongoDB in Python
- Wrap-Up

Learning Outcomes

MAKE SCHOOL

By the end of today, you should be able to...

1. **Describe** the structure of a Document-based Database.
2. **Identify** the operations used to create, read, update, & delete objects from a MongoDB database using Flask-PyMongo.

MAKE SCHOOL

Warm-Up: Quiz Review

Quiz Review

MAKE SCHOOL

As we go through each of the following quiz answers, think about:

- What mistake did this student make? How would you correct it?
- What would you say to that student to explain your reasoning?

Key/value pairs

MAKE SCHOOL

Q2.2 Retrieve key/value pairs

3 Points

I am writing a Flask route for the drink order page which accepts a POST request from the form shown above. You will fill in the TODOs to complete the function.

```
@app.route('/order', methods=['POST'])
def order_page():
    nickname = '' # TODO 1
    drink_type = '' # TODO 2

    # ... make order here

    return redirect(url_for('thank_you', name=nickname, drink=drink_type))
```

What should I put in place of `TODO 1` to define the `nickname` variable?

request.args.nickname()

What should I put in place of `TODO 2` to define the `drink_type` variable?

request.args.drink_type()

GET/POST

MAKE SCHOOL

Q3.1 Contact Me form

2 Points

I am writing a website for my personal blog. One of the features is a Contact Me form that will send me an email when the user enters their information.

When the user submits the contact form, should it make a GET or POST request?

- (•) GET
- ( ) POST

Justify your answer:

You use GET because you are receiving information inputed by the user from the server.

Templating

MAKE SCHOOL

Q4 Templating

6 Points

I am writing a Lunch Tracker application that will show the user what they have packed for lunch each day. Since I don't yet have a database set up, I am testing my app with some sample data. Here is what I have so far in my route function in `app.py`:

```
@app.route('/lunch')
def lunch_tracker():
    """Show the user what they have packed for lunch."""
    context = {
        'is_sunny': True,    # Could be either True or False

        'lunch_items': [    # Could contain any lunch items
            'apple',
            'sandwich',
            'crackers'
        ]
    }
    return render_template('lunch.html', **context)
```

Here is what I have so far for `lunch.html`:

```
<!-- lunch.html -->

<!-- TODO 1: Use a Jinja if statement to display the first line if it's sunny,
and the second line otherwise. -->

You should eat outside!

Let's eat indoors.

<!-- TODO 2: Use a Jinja for loop to display each lunch item on a separate
bullet point. -->

You packed the following for lunch:
<ul>

    <li>Lunch Item Goes Here</li>

</ul>
```

Below, write the **ENTIRE** contents of the `lunch.html` file, with the `TODO`s completed. No need to include base HTML tags (doctype, html, head, body).

```
<!-- lunch.html -->

<!-- TODO 1: Use a Jinja if statement to display the first line if it's sunny,
and the second line otherwise. -->

You should eat outside!
{% if 'is_sunny': True  %}
Let's eat indoors.
{% if 'is_sunny': Flase  %}
<!-- TODO 2: Use a Jinja for loop to display each lunch item on a separate
bullet point. -->
{% for item in seq -%}
   {{ 'apple' }}
   {{ 'sandwich' }}
   {{ 'crackers' }}
{%- endfor %}
You packed the following for lunch:
<ul>

   <li> 'lunch_items' </li>

</ul>
```

MAKE SCHOOL

GET vs. POST: An Analogy

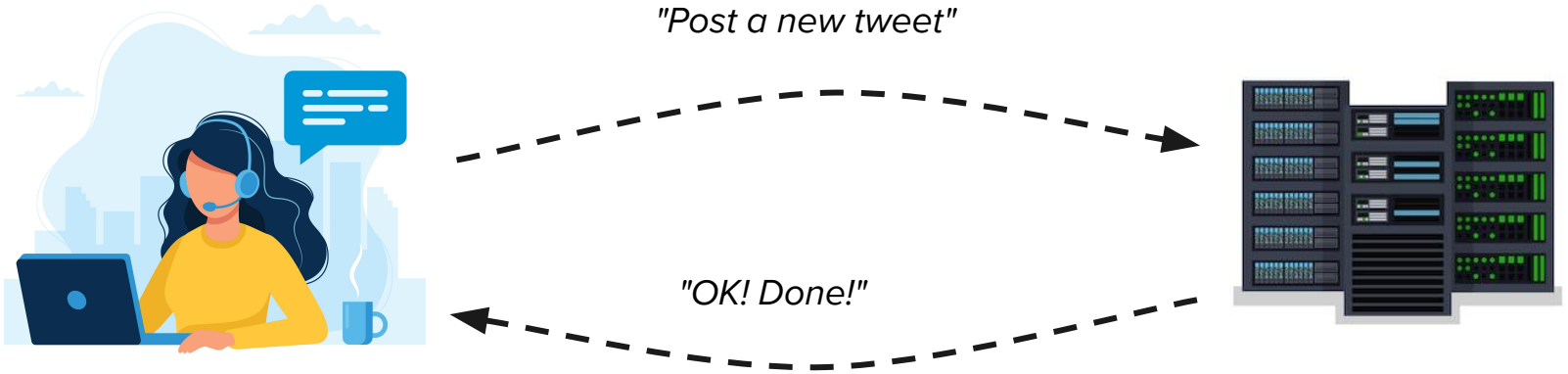
GET & POST requests

Whenever you make a **request** on the web, there is always a **response**. This is true of both GET & POST requests.

GET request (& response)

GET & POST requests

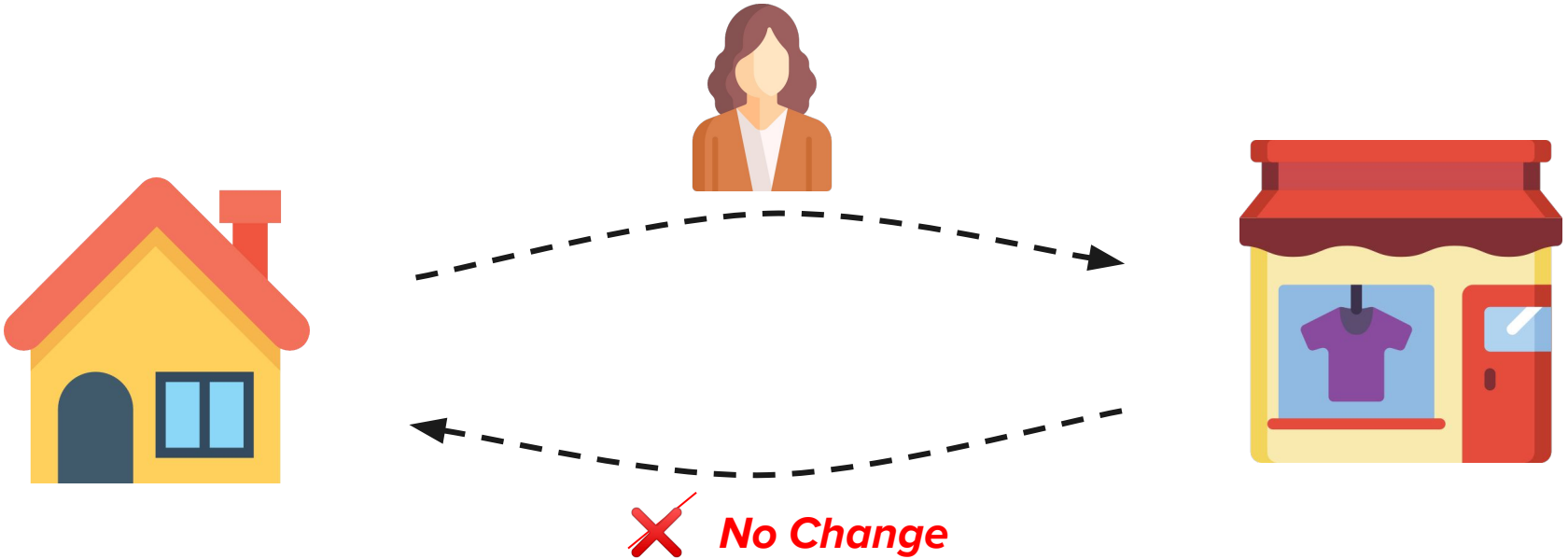
Whenever you make a **request** on the web, there is always a **response**. This is true of both GET & POST requests.



POST request (& response)

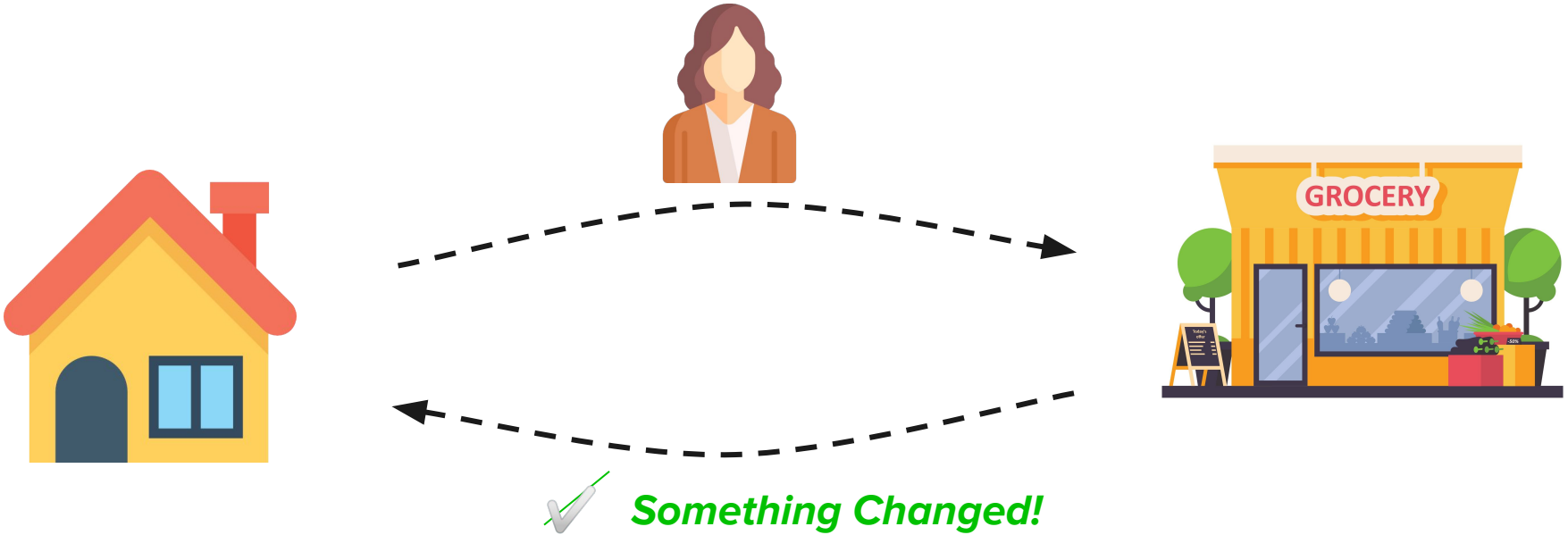
GET - Window Shopping

Making a **GET** request is like going **window shopping**. When I get back home, I have more information about what's in the store, but my bank account balance (and the store's inventory) are the same.



POST - Grocery Shopping

Making a **POST** request is like going **grocery shopping**. When I get back home, my bank account balance has changed, the store's inventory decreased, and my pantry contents increased.



MAKE SCHOOL

Why Databases?

What is a database? What is it useful for?

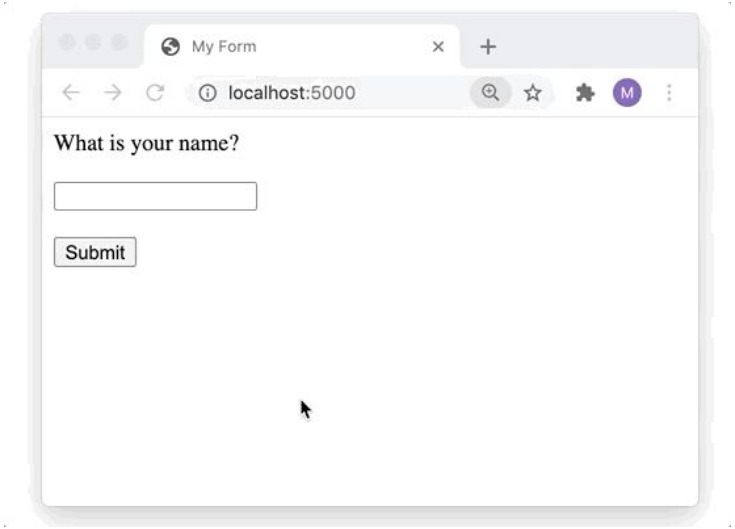
MAKE SCHOOL

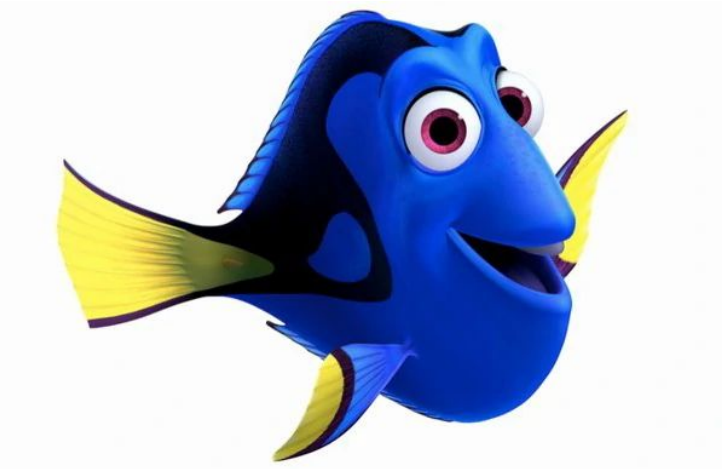
- Referencing user data
- Store user data
- Create relationships between data - e.g. patient ID is related to doctor ID
- Relate an ID to a piece of data

Databases

We can collect information about a user, and save it to a variable. But what happens when we restart the server?

All of that information is forgotten!



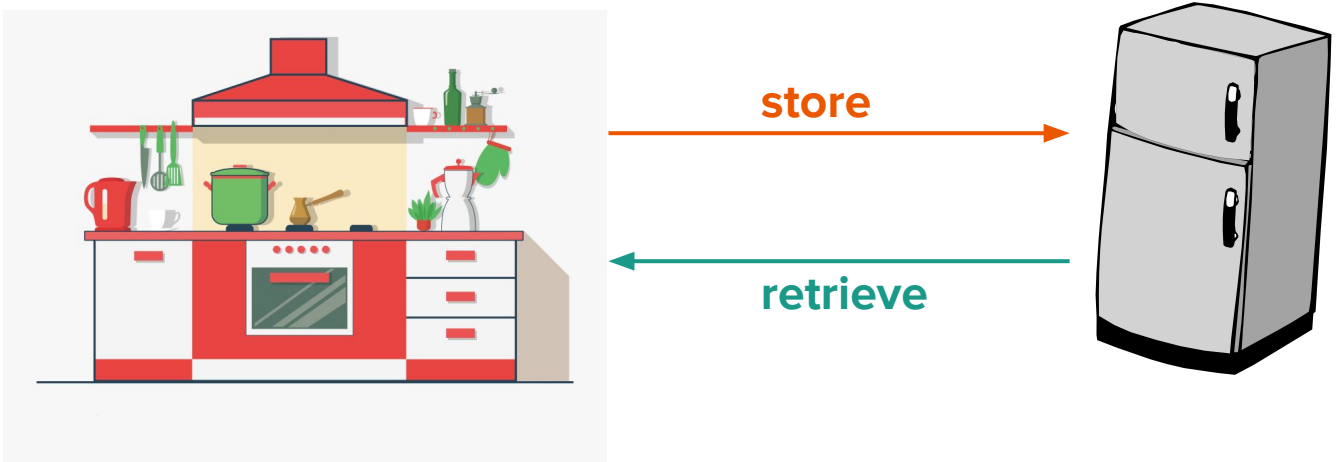


Databases

Think about a site that requires user login, like Facebook.

What would happen if it didn't have a database?

If our web application is like a kitchen, then the database is like a refrigerator that keeps our data safe until we need to use it again.



What is MongoDB?

MAKE SCHOOL

In this class, we'll be using **MongoDB** as our database, and **PyMongo** as the Python library to connect to it.

MongoDB is called a **NoSQL**, or **Document-based** database, because we don't have to specify ahead of time what our data will look like. Instead, we can store **any key-value pairs** in a database item.

Activity (10 minutes)

MAKE SCHOOL

With a partner, follow the steps in the MongoDB Setup Tutorial to install MongoDB and run a local server on your computer.

MAKE SCHOOL

So... What is a Document-based Database?

MongoDB Vocabulary

MAKE SCHOOL

A **resource** refers to **one type of data** that can be saved to a database. For example, in a Music application, we may have the resources of **Song**, **Artist**, **Album**, and **Playlist**.

A resource usually has **attributes** (sometimes called **fields**). For example, the **Song** resource may have attributes of: song name, artist id, rating, and publish date.

What **resources** might we have for an **Online Store** application? Name at least 3.

MongoDB Vocabulary

A **document** is a JSON object containing data related to **one single object** of a resource. E.g. we may have a document for the song “Single Ladies” by Beyonce.

```
{
    'name': 'Can\'t Buy Me Love',
    'artistId': ObjectId('12345'),
    'avgRating': 4.7
}
```

```
{
    'name': 'Single Ladies',
    'artistId': ObjectId('12347'),
    'avgRating': 4.7
}
```

```
{
    'name': 'Teardrops on My Guitar',
    'artistId': ObjectId('12342'),
    'avgRating': 4.7
}
```

```
{
    'name': 'Roar',
    'artistId': ObjectId('12343'),
    'avgRating': 4.7
}
```

MongoDB Vocabulary

A **collection** is a container for documents specific to a certain type of object, or resource. E.g. we may have one collection for **songs**, another for **albums**, another for **artists**, another for **playlists**, etc. A collection contains many **documents**.

MongoDB Vocabulary

A **database** is a container for many collections. Typically, we will use one database for one project.

Songs Database

Songs

```
{
    'name': 'Can\'t Buy Me Love',
    'artistId': ObjectId('12345'),
    'avgRating': 4.7
}
```

```
{
    'name': 'Single Ladies',
    'artistId': ObjectId('12347'),
    'avgRating': 4.7
}
```

...

Albums

```
{
    'title': 'Hey Jude',
    'artistId': ObjectId('12347'),
    'published': '2/26/1970'
}
```

```
{
    'title': 'Lemonade',
    'artistId': ObjectId('12347'),
    'published': '4/23/2016'
}
```

...

Artists

```
{
    'name': 'The Beatles',
    'members': [ 'John Lennon',
        'Paul McCartney',
        'George Harrison',
        'Ringo Starr' ]
}
```

```
{
    'name': 'Beyonce',
    'born': '9/4/1981'
}
```

...

Activity (10 minutes)

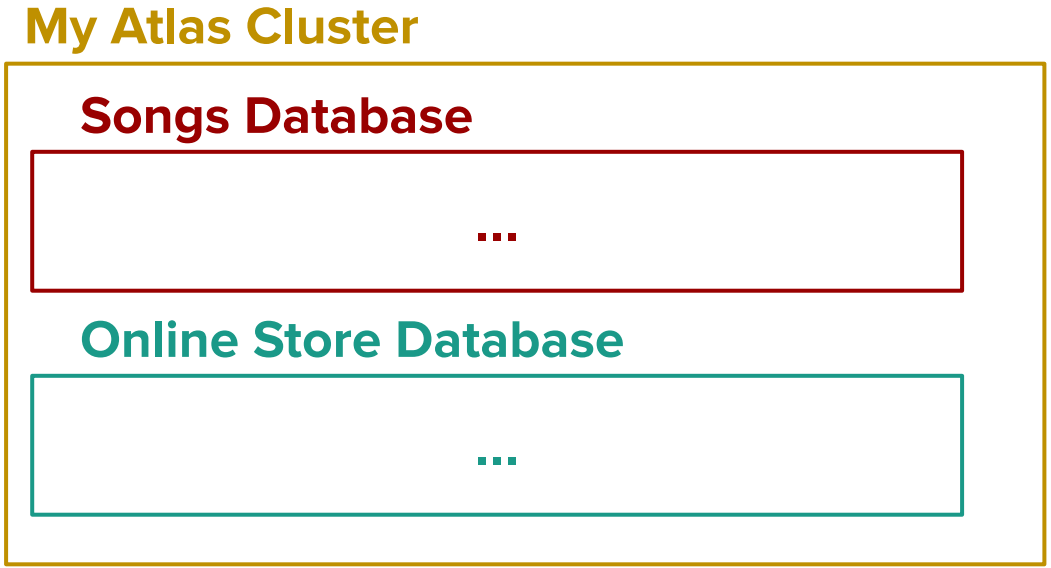
MAKE SCHOOL

In a group of 3, discuss which collections you would use to create an online store where users can sign up, view items, add items to a shopping cart, and purchase items. Write down 3 possible collections and give an example document for each collection.

Use this worksheet to organize your thoughts.

MongoDB Vocabulary

A **cluster** is specific to MongoDB Atlas, and gives computing power to hosting your data. A cluster can contain many **databases** (but doesn't have to).



MAKE SCHOOL

Break - 10 min

MAKE SCHOOL

How do I use it in Python?

Set Up

MAKE SCHOOL

We can import the PyMongo module in our code:

```
from flask import Flask
from flask_pymongo import PyMongo

app = Flask(__name__)
app.config["MONGO_URI"] = "mongodb://localhost:27017/myDatabase"

mongo = PyMongo(app)
```

Then, we can use the **mongo.db** object directly in our routes:

```
@app.route("/")
def home_page():
    online_users = mongo.db.users.find({"online": True})
    return render_template("index.html",
        online_users=online_users)
```

MongoDB Operations

MAKE SCHOOL

There are **four operations** we can do on a database document. You can remember them with the acronym **C.R.U.D.**

C reate

R ead

U pdate

D elete

MongoDB Operations - Insert One

MAKE SCHOOL

We can **create a new document** using the operation `insert_one`.

```
new_user = {
    'first_name': 'Meredith',
    'role': 'instructor',
    'num_pets': 1
}

result = mongo.db.users.insert_one(new_user)
```

This means that we are creating a new document in the **users** collection of the **db** database.

What is an ObjectId?

All new objects in the database are automatically given a field **_id** which is an randomly-generated 24-digit hexadecimal string.

However, this field is stored as type **bson.objectid.ObjectId**, not as a string.

```
{
    'name': 'Meredith',
    'role': 'instructor',
    'num_pets': 1,
}
```

```
{
    'name': 'Meredith',
    'role': 'instructor',
    'num_pets': 1,
    '_id': ObjectId('5f5f871c9bca94a49d6e8956')
}
```

MongoDB Operations - Find

MAKE SCHOOL

We can get all objects in a collection by using **find**:

```
all_users = mongo.db.users.find()
for user in all_users:
   print(user['name'])
```

We can also get all objects matching some constraint(s):

```
all_instructors = mongo.db.users.find({'role': 'instructor'})
for user in all_instructors:
   print(user['name'])
```

Activity (5 minutes)

MAKE SCHOOL

Use the MongoDB Playground to experiment with using the **find** operation:

- What happens if you create two identical objects? How can you tell them apart?
- What happens if you include nested data (lists/objects) inside of a document? How could you retrieve that information?

MongoDB Operations - Find One

MAKE SCHOOL

We can get one single object using **find_one**:

```
user1 = mongo.db.users.find_one({'first_name': 'Meredith'})
print(user1)


>>> {'name': 'Meredith', 'role': 'instructor', 'num_pets': 1,
'_id': ObjectId('5f5f871c9bca94a49d6e8956')}
```

MongoDB Operations - Update One

MAKE SCHOOL

We can update an existing entry using **update_one** and setting a field called **$set** in the second parameter.

```
user1 = mongo.db.users.update_one({
   'first_name': 'Meredith'
},
{
   '$set': { 'num_pets': 2 }
})
print(user1)

>>> {'name': 'Meredith', 'role': 'instructor', 'num_pets': 2,
'_id': ObjectId('5f5f871c9bca94a49d6e8956')}
```

MongoDB Operations - Delete One

MAKE SCHOOL

We can delete an entry by using **delete_one**.

```
result = mongo.db.users.delete_one({
    'first_name': 'Meredith'
})
print(result.deleted_count)


>>> 1
```

Activity

MAKE SCHOOL

Run the code for the Fruits Database Repl. Notice what happens after each database operation is run.

MAKE SCHOOL

Lab Time

Homework

Homework 4: Databases will require you to use a MongoDB database to create, read, update, & delete objects.