MAKE SCHOOL

Databases with MongoDB & PyMongo

WEB 1.1

Agenda

MAKE SCHOOL

- Learning Outcomes
- Code Review
- APIs Bonus Topics - virtual environments, environment variables
- 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

Code Review

Warm-Up: Code Review (20 minutes)

MAKE SCHOOL

Break out into pairs and choose who will be the **reviewer** and **reviewee** for your finished Homework 3 (More Forms) assignment.

- **Reviewee**: Share your screen and explain what your code does from top to bottom.
- **Reviewer**: Listen, ask questions, and make suggestions for improvement.

After 7 minutes, switch roles.

** You may choose to do a code review of a different assignment, if you prefer.

MAKE SCHOOL

Virtual Environments

What is a virtual environment?

A **virtual environment** allows us to manage Python packages for different projects. Think of it like a **sandbox or container** where we can install packages and they won't affect the rest of your file system or projects.

For example, for **Project A** you may need **Version 1** of a package, but for **Project B** you may need **Version 2**. Since those versions are incompatible, you'll need a separate virtual environment for each project.

How do I use one?

MAKE SCHOOL

Navigate to your project directory, then:

- `python3 -m venv env` - create a folder called `env` that will hold all installed packages
- `source env/bin/activate` - activate your virtual environment; do this before you run your code or install packages
- `deactivate` - deactivate your virtual environment; do this when switching to work on a different project

What is requirements.txt?

MAKE SCHOOL

In Python projects, we use a file called `requirements.txt` to list all of the project's required packages.

- `pip3 install -r requirements.txt` - install all packages listed in the requirements
- `pip3 freeze > requirements.txt` - populate the requirements.txt file with all packages that are currently installed in the environment

Activity (10 minutes)

MAKE SCHOOL

Navigate to your Homework 3 directory. Use the steps on the previous slides to create a virtual environment, activate it, install Flask, & "freeze" to a requirements file.

MAKE SCHOOL

Environment Variables

What are Environment Variables?

MAKE SCHOOL

Sometimes, there are pieces of information (e.g. API keys) that you want to keep secret from the public. (Why?)

We call these pieces of information **secrets**.

It is very important to hide your secrets!!!

We can do this by saving each "secret" as an **environment variable**.

What are Environment Variables?

MAKE SCHOOL

An **environment variable** is a key-value pair that is saved to your operating system's environment.

We can read in environment variables using the **dotenv** Python package.

How do we use it?

MAKE SCHOOL

1. Install the dotenv package: `pip3 install python-dotenv`
2. Create a file called `.env` and enter your key-value pairs (do not use quotes):

```
# .env
SECRET_KEY=ilikebananas
```

3. In your Python code, enter the following:

```
import os
from dotenv import load_dotenv
load_dotenv()

secret_key = os.getenv('SECRET_KEY')
print(secret_key) # 'ilikebananas'
```

Activity (10 minutes)

MAKE SCHOOL

In your Homework 3 directory, create a `.env` file.

Follow the steps on the previous slide to create a key-value pair for the **GIF Search API key** instead of hard-coding it.

The `datetime` Library

datetime

The `datetime` library is built into Python, so you don't need to install it.

This library provides us access to `datetime` objects - which represent a specific date and time in object form.

```
from datetime import datetime

now_date_obj = datetime.now()

print(now_date_obj)

# 2020-09-02 21:17:19.511047
```

datetime

MAKE SCHOOL

The datetime object has many useful fields:

```
print(now_date_obj.day) # 2

print(now_date_obj.month) # 9

print(now_date_obj.year) # 2020

print(now_date_obj.hour) # 21

print(now_date_obj.minute) # 20
```

Check out the datetime documentation for more info.

Converting datetime

We can convert a datetime to a string using `strftime`:

```
print(now_date_obj.strftime('%Y-%m-%d'))
# '2020-09-02'
```

format codes

And we can convert a string to a datetime using `strptime`:

```
another_date = datetime.strptime('2020-09-03', '%Y-%m-%d')
print(another_date) # 2020-09-03 00:00:00
```

Time since epoch

MAKE SCHOOL

The "**epoch time**" of any datetime object is the number of seconds that have elapsed since January 1st, 1970 (considered to be the "epoch" of the Internet).

```
print(now_date_obj.strftime('%s'))
# 1599082605
```

Activity

Read over this Format Code List for a complete list of codes. Try using some to convert a datetime to a string, or vice versa!

What is **one question** you still have about date objects?

- Time durations
- Timezones
- UTC vs. local time

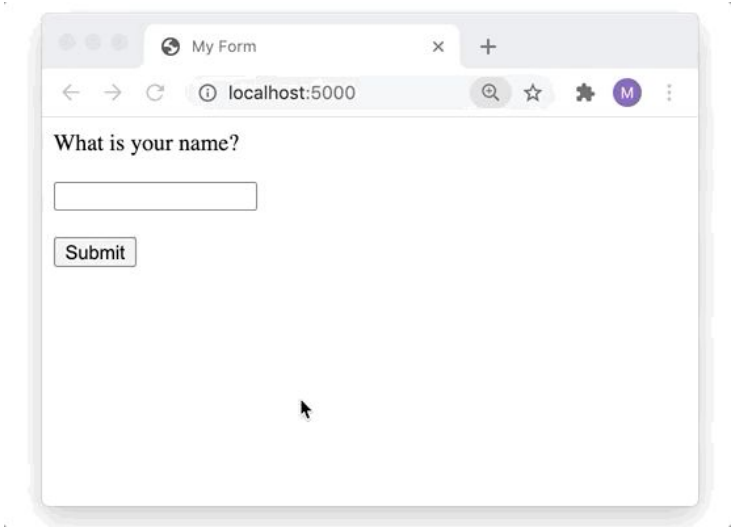
MAKE SCHOOL

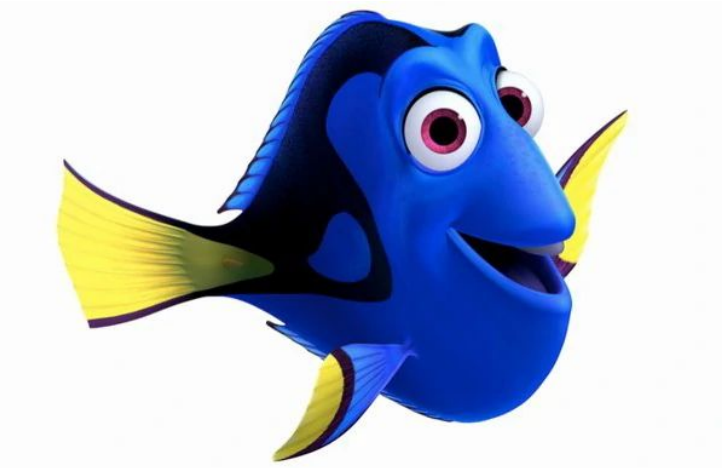
Why Databases?

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!



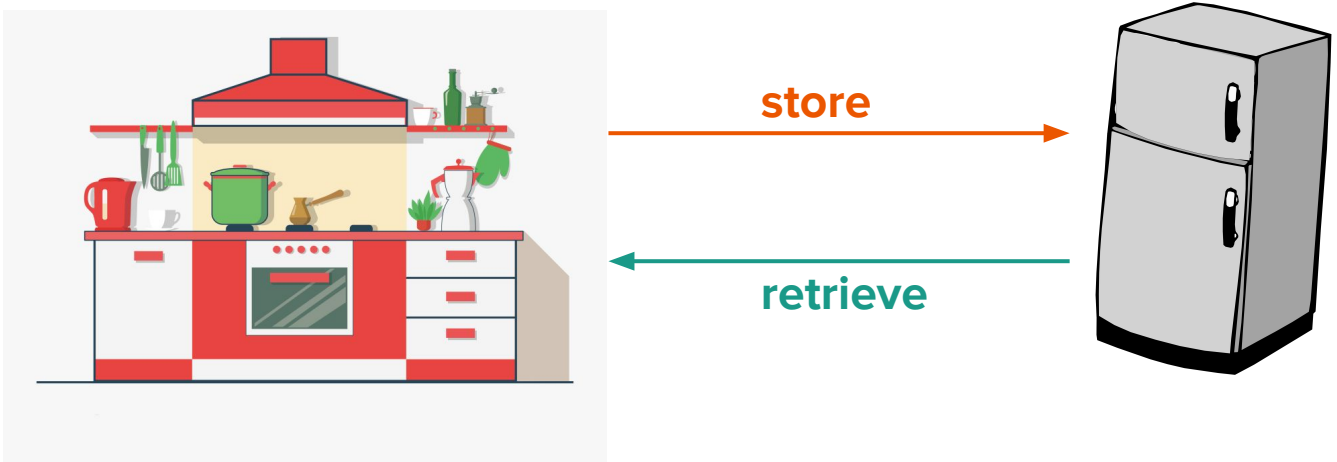


Databases

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)

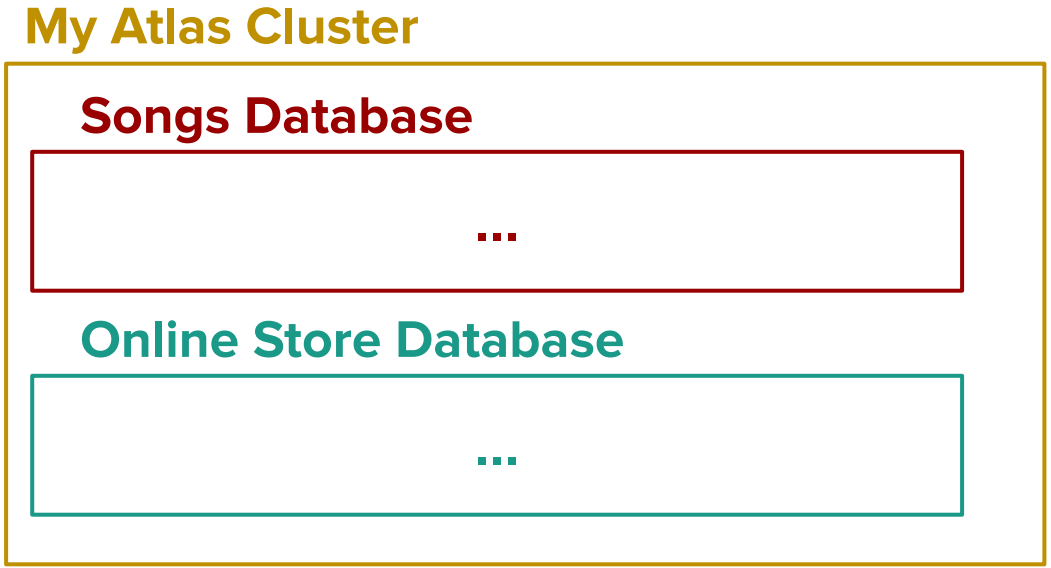
MAKE SCHOOL

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

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')}
```

Activity

MAKE SCHOOL

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

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
```

MAKE SCHOOL

Lab Time

Homework

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