

10.4.1 Store the Data

Robin is pretty excited about all of the data we've managed to scrape. And the code is designed to grab the most recent data, so if it's run at a later time, all of the results will have been updated—without us needing to alter the code.

Now that she has the results she wants, she needs to store them in a spot where they can be easily accessed and retrieved as needed. SQL isn't a good option because it works with tabular data, and only one of the items we scraped is presented in that format. Even then, it's all condensed into a block of HTML code.

What Robin will need to use is a database that works differently from SQL with its neatly ordered tables and relationships. Mongo, a NoSQL database, is designed for exactly this task. While Robin is familiar with SQL databases, Mongo is completely new and we'll need to practice with it before loading in our scraped data.

The data Robin has gathered for her web app is great. She's been able to pull a great image, the most recent news article summary, and even an HTML table. Each data type is different, though, with text and images and HTML all together. Compared to SQL's orderly relational system, where each table is linked to at least one other by a key, the data we've helped Robin gather is a bit chaotic. This is where a non-relational database comes in.

MongoDB (Mongo for short) is a non-relational database that stores data in Binary JavaScript Object Notation (JSON), or BSON format. We'll access data stored in Mongo the same way we access data stored in JSON files. This method of data storage is far more flexible than SQL's model. If you'd

like to dig into MongoDB at a deeper level, check out the [official documentation](https://docs.mongodb.com/) [\(https://docs.mongodb.com/\)](https://docs.mongodb.com/).

REWIND

JSON, JavaScript Object Notation, is a method that sorts and presents data in the form of key:value pairs. It looks much like a Python dictionary and can be traversed through using list notation.

A Mongo database contains collections. These collections contain documents, and each document contains fields, and fields are where the data is stored.

While Mongo and SQL are both databases, that's where the similarities end. They handle documents differently, the storage model isn't even close, and we even interact with them in very different ways.

1. To get started with Mongo, first open a new terminal window, but make sure your working environment is activated. Note that your environment does not need to have the same name as the one in the image.

A screenshot of a terminal window with a dark background. The title bar at the top shows a home icon, the text 'arwen - -bash - 80x24', and three colored window control buttons (red, yellow, green). The terminal content shows two lines of green text: '(base) L10465-C02ZM2FBLVDL:~ arwen\$ conda activate work' and '(work) L10465-C02ZM2FBLVDL:~ arwen\$'.

```
(base) L10465-C02ZM2FBLVDL:~ arwen$ conda activate work
(work) L10465-C02ZM2FBLVDL:~ arwen$
```

2. Then, to start an instance, type `mongod` into the first line of your terminal and press return or enter on your keyboard. Some Mac users may not need to run this command as Mongo is already running in the background

We need to keep this tab open and active so that the Mongo instance continues to run. While Mongo does have a GUI, similar to pgAdmin

for Postgres, we'll be using a command line interface (CLI) to make connections within the database.

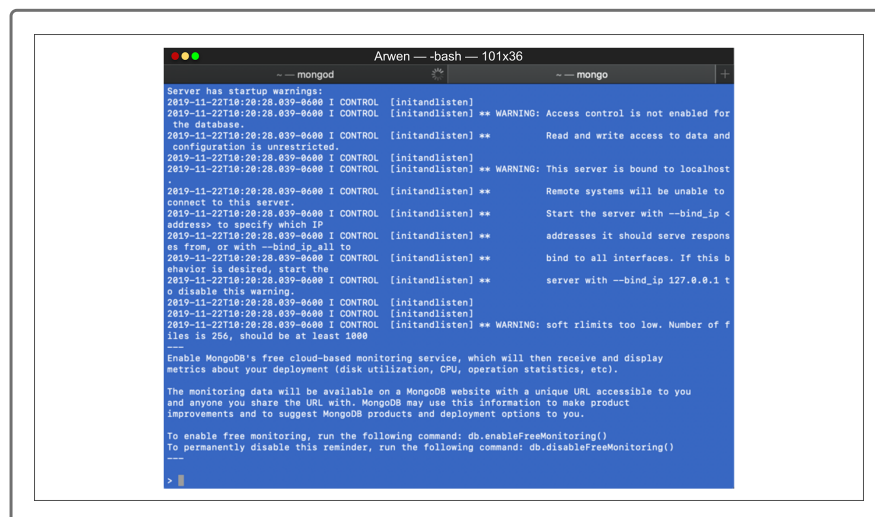


```
arwen -- -bash -- 80x24
(base) L10465-C02ZM2FBLVDL:~ arwen$ conda activate work
(work) L10465-C02ZM2FBLVDL:~ arwen$ mongod_
```

3. In our terminal, create a second window or tab to use for working in Mongo. Again, make sure your environment is active.

On the first line of this new window, type "mongo." This is done in a new window because, after you execute the command, you cannot use the terminal for other tasks—only to send information to and from the database.

After executing the command, your terminal will show a right angle bracket and a blinking cursor. This indicates that the database is active and ready for use.



```
Arwen -- -bash -- 101x36
-- mongod
Server has startup warnings:
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten]
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten]
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten]
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten]
2019-11-22T10:20:28.039-0600 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

SHOW PRO TIP

Create a Database

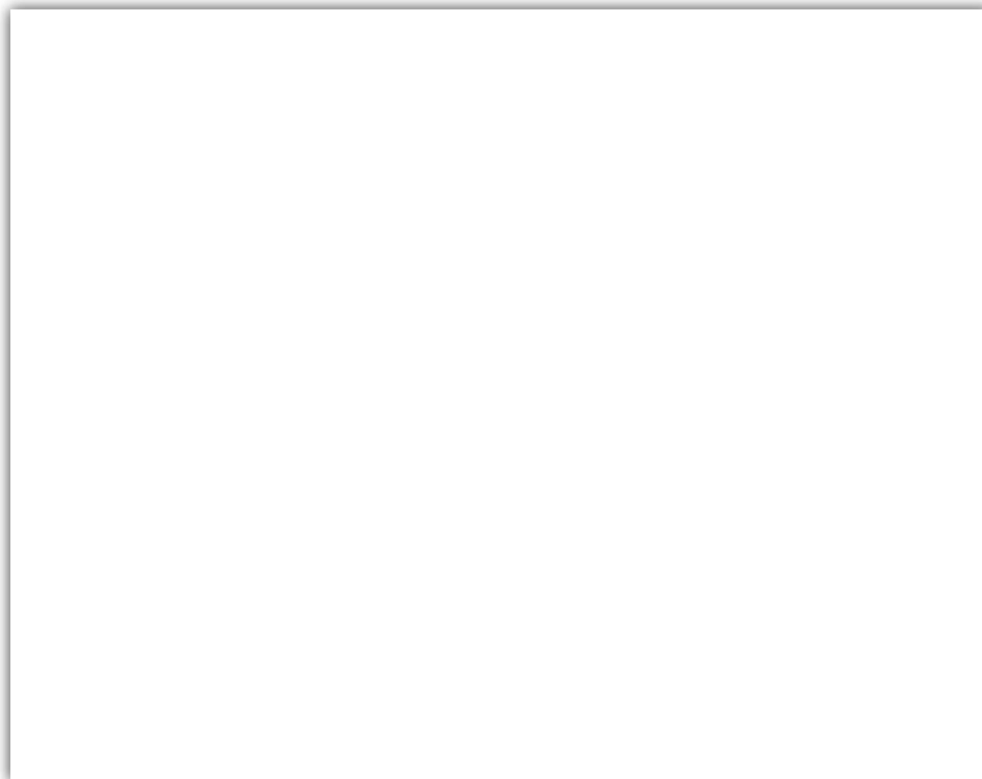
We don't have a fancy GUI to use while navigating through creating a database and inserting data, but that doesn't mean that our commands will be very complex. Let's create a new practice database to get used to some of the more common commands.

In the terminal where Mongo is active and awaiting instruction, type "use practicedb" and then press Enter. This creates a new database named "practicedb" and makes it our active database.

Our New Database

```
> use practicedb
switched to db practicedb
>
```

If you're not sure which database you're using, type "db" in the terminal and press Enter.



After typing "db" into the terminal and pressing Enter, the name of the current active database is returned. This is a quick check to make sure

we'll be saving data to the right spot.

Current Active Database

```
> db
practicedb
>
```

You can also see how many databases are stored locally by typing "show dbs" in your terminal. There should be a few already there by default, so don't be alarmed if more than one appears that you didn't create yourself.

There is also a way to check to see what data, or collections, are already in the database. Type "show collections" into the shell, or terminal, then press Enter.

Nothing came up after that, right? That's a good thing. We haven't entered any data yet. We'll practice doing that next.

Insert Data

Now that we've confirmed we're in the right database, we can practice the commands to insert data or a document.

The syntax follows: `db.collectionName.insertOne({key:value})`. Its components do the following:

- `db` refers to the active database, practicedb.
- `collectionName` is the name of the new collection we're creating (we'll customize it when we practice).
- `.insertOne({ })` is how MongoDB knows we're inserting data into the collection.
- `key:value` is the format into which we're inserting our data; its construction is very similar to a Python dictionary.

In short, we're saying, "Hey, Mongo, use the database we've already specified, and insert a document into this collection. If there's not a collection named that, then create one."

Let's explore how this works a bit by adding some zoo animals to our collection.

In the shell, type:

```
db.zoo.insertOne({name: 'Cleo', species: 'jaguar', age: 12, hobbies: ['sleep
```

After pressing Enter, the next line in your terminal should confirm the write. This means that we've successfully inserted Cleo into the database.

Now let's add another animal. In your shell, type the following:

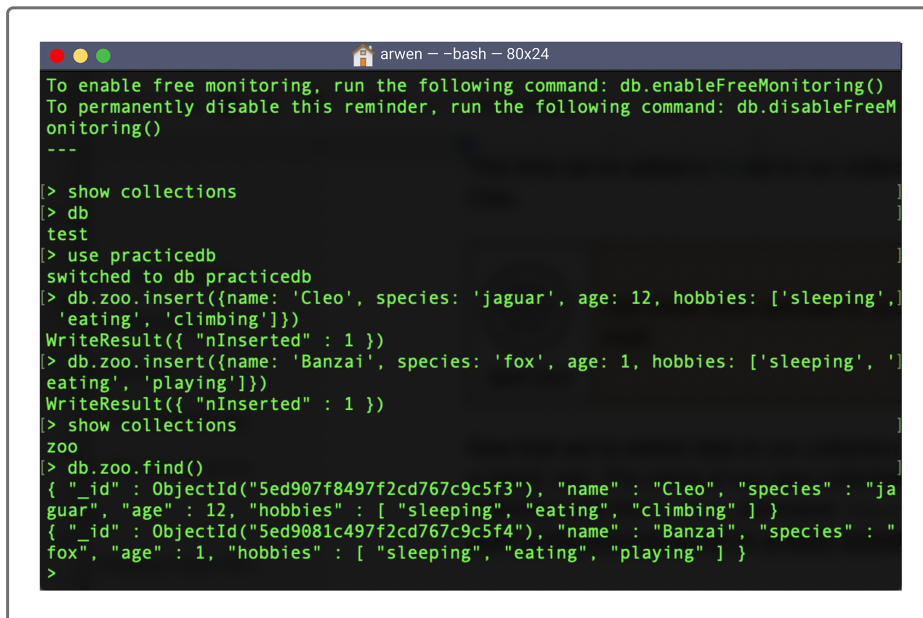
```
db.zoo.insertOne({name: 'Banzai', species: 'fox', age: 1, hobbies: ['sleeping', 'eating', 'playing']}).
```

This time we've added a fox to our collection, but the code is very similar to when we added Cleo.

SKILL DRILL

Add three more animals to your database, then type "show collections" in your shell.

Now that we've added data to our collection, when we type "show collections" we'll actually see a result: zoo. The name of our new collection is returned. We can also view what's inside a collection with the `find()` command. Executing `db.zoo.find()` in the terminal will return each of the documents we've already added.



```

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show collections
> db
test
> use practicedb
switched to db practicedb
> db.zoo.insert({name: 'Cleo', species: 'jaguar', age: 12, hobbies: ['sleeping', 'eating', 'climbing']})
WriteResult({ "nInserted" : 1 })
> db.zoo.insert({name: 'Banzai', species: 'fox', age: 1, hobbies: ['sleeping', 'eating', 'playing']})
WriteResult({ "nInserted" : 1 })
> show collections
zoo
> db.zoo.find()
{ "_id" : ObjectId("5ed907f8497f2cd767c9c5f3"), "name" : "Cleo", "species" : "jaguar", "age" : 12, "hobbies" : [ "sleeping", "eating", "climbing" ] }
{ "_id" : ObjectId("5ed9081c497f2cd767c9c5f4"), "name" : "Banzai", "species" : "fox", "age" : 1, "hobbies" : [ "sleeping", "eating", "playing" ] }
>

```

Documents can also be deleted or dropped. The syntax to do so follows:

```
db.collectionName.deleteOne({}).
```

So, if we wanted to remove Cleo from the database, we would update that line of code to:

```
db.zoo.deleteOne({name: 'Cleo'}).
```

We can also empty the collection at once, instead of one document at a time. For example, to empty our pets collection, we would type:

```
db.zoo.remove({}).
```

Because the inner curly brackets are empty, Mongo will assume that we want everything in our pets collection to be removed.

Additionally, to remove a collection all together, we would use

```
db.zoo.drop().
```

After running that line in the shell, our pets collection will no longer exist at all.

And to remove the test database, we will use this line of code:

```
db.dropDatabase().
```

SKILL DRILL

Remove the animals you added earlier with the `deleteOne({})` method, then drop the database.

Now test your code as you answer the following question:



You can quit the Mongo shell by using keyboard commands: Command + C for Mac or CTRL + C for Windows. This stops the processes that are actively running and frees up your terminal. Remember to quit both the server and the shell when you're done practicing. Otherwise, they'll continue to run in the background and use system resources, such as memory, and slow down the response time of your computer.

IMPORTANT

Create a new database named "mars_app" to hold the Mars data we scrape. It is essential that this database exists before we start running our web app outside of Jupyter Notebook, otherwise we'll encounter errors.