Quick Intro to Flask Part III

Version 1.00

This is the third installment of an introduction to web application programming using Flask. The coverage involves access to a NoSQL database.

Prerequisites

You should have gone through the introduction to NoSQL Databases/MongoDB lecture and already have set up the required databases and collections.

You should also already have gone through Parts I and II of this Flask intro series.

Finally, you should already have installed PyMongo on your computers. If you haven't done so yet, run this command from your OS terminal/command prompt:

```
conda install pymongo
```

Run Flask

If you haven't done so yet, run Flask from Terminal (Mac OS users) or from the Command Line/Anaconda Prompt (Windows users).

As we haven't yet covered how to make environment variables more permanent, we will have to go through the next few steps again:

If you are on Mac:
```
$ export FLASK_APP=app.py
$ export FLASK_ENV=development
```

If you are on Windows Anaconda Prompt:
```
C:\path\to\app>set FLASK_APP=app.py
C:\path\to\app>set FLASK_ENV=development
```

Make sure you are in the directory where your app.py file is located.

To run Flask:
```
$ flask run
```

Replace hard-coded database constructs

Up to this point, we relied heavily on hard-coded (mock) database objects from within our Python code. We will be gradually replacing these mock objects with calls to an actual database.

Exercise: include MongoDB-specific code

1. In the file **database.py**, import pymongo. The first few lines of code should look like the following. Your actual code may be different, depending on additional dictionaries you added in your last Take Home Quiz.

   ```
   import pymongo

   products = {
       100: {"name":"Americano","price":125},
       200: {"name":"Brewed Coffee","price":110},
       300: {"name":"Cappuccino","price":120},
       400: {"name":"Espresso","price":120},
       500: {"name":"Latte","price":140},
       600: {"name":"Cold Brew","price":200}
   }

   branches = {
       1: {"name":"Katipunan"},
       2: {"name":"Tomas Morato"},
       3: {"name":"Eastwood"},
       4: {"name":"Tiendesitas"},
       5: {"name":"Arcovia"},
   }
   ```

2. Still in the file **database.py**, we need to establish a database *client* to MongoDB. A *client* in this context is just a consumer of data or services from a *server*; in our specific case, our database server is MongoDB, which listens to port 27017 by default.

   The first few lines of **database.py** should look like the following:

   ```
   import pymongo

   myclient = pymongo.MongoClient("mongodb://localhost:27017/")

   products_db = myclient["products"]
   ```

```
order_management_db = myclient["order_management"]
```

At this point, it is best that you save your changes to the file.

Notes:

- The `products_db` variable is a placeholder and pointer to the MongoDB products database (equivalent to specifying the **`use products`** command in the MongoDB console)
- The `order_management_db` variable is a placeholder and pointer to the MongoDB `order_management` database (equivalent to specifying the **`use order_management`** command in the MongoDB console)

Exercise: Authenticate using user credentials stored in the database

1. In **database.py**, replace the function `get_user(username)` as follows:

```
def get_user(username):
    customers_coll = order_management_db['customers']
    user=customers_coll.find_one({"username":username})
    return user
```

   Save your changes.

2. Still in **database.py**, remove the **`users`** dictionary containing our old hard-coded user information. We will no longer need this because we will be querying our MongoDB database for user information from now on.

3. While the user dictionary structure has slightly changed, the old code in **authentication.py** will still work without requiring any modifications. Test your newly-updated app by logging in. Note that if in a previous exercise you've changed passwords in the database, please login with the new passwords accordingly.

   Are you able to login successfully?

Exercise: Point to our MongoDB database for product information

Warning: For our hard-coded product catalog, we relied on the **code** key to be able to quickly access data in the dictionary. We shall be using a slightly different structure moving forward, so we will have to make modifications in various areas of our Python code.

1. In **database.py**, replace the function `get_products()` as follows:

```
def get_products():
    product_list = []

    products_coll = products_db["products"]

    for p in products_coll.find({}):
        product_list.append(p)

    return product_list
```

2. Still in **database.py**, replace the function `get_product(code)` as follows:

   ```
   def get_product(code):
       products_coll = products_db["products"]

       product = products_coll.find_one({"code":code})

       return product
   ```

   Save your changes.

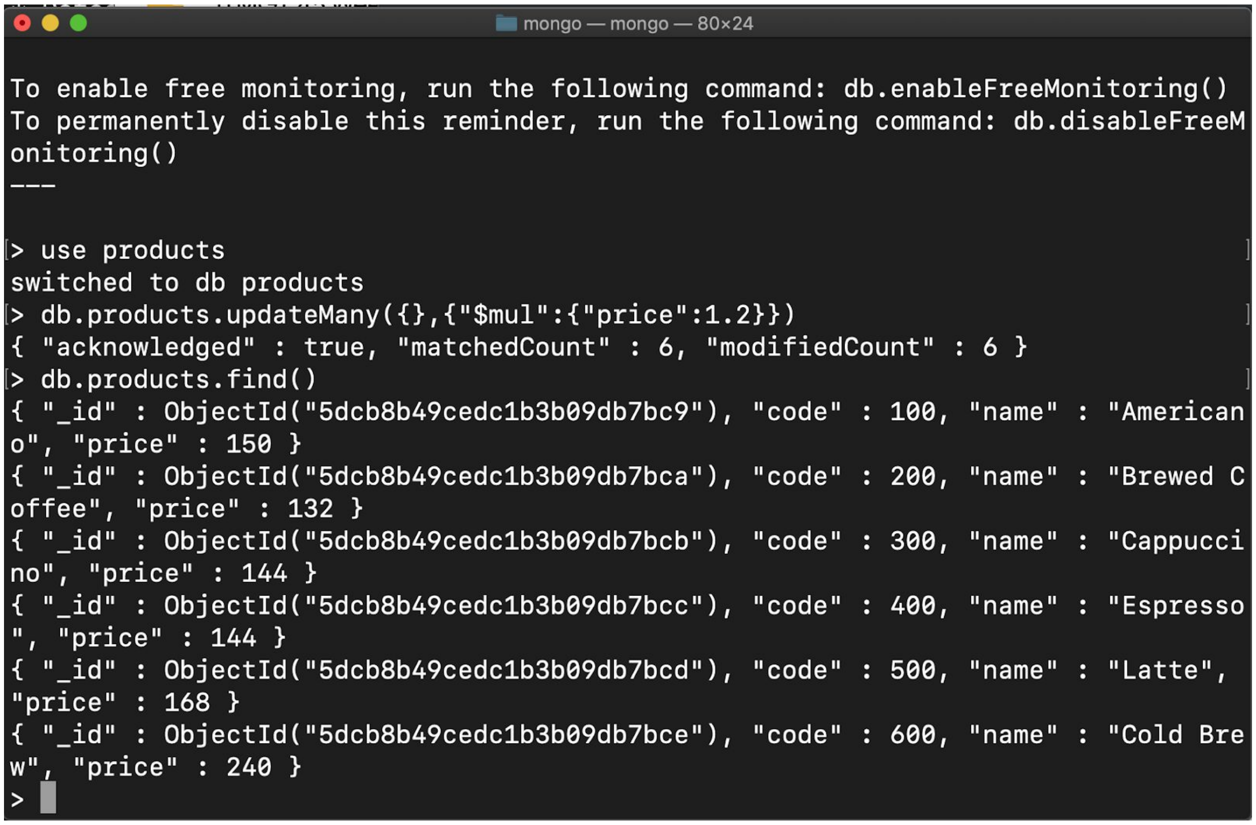
3. Still in **database.py**, remove the **`products`** dictionary containing our old hard-coded products information. We will no longer need this because we will be querying our MongoDB database for product information from now on.

4. Go back to your web browser and try accessing Products and Product Details pages. If things are working, you will not notice any visual changes.

5. Let's simulate a price increase of 20% across all products. For this, we shall go directly to MongoDB using the command line. From your MongoDB session, run the commands as shown below in **bold**.

   ```
   > use products
   switched to db products
   > db.products.updateMany({},{"$mul":{"price":1.2}})
   { "acknowledged" : true, "matchedCount" : 6, "modifiedCount" :
   6 }
   ```

   Your MongoDB terminal session should look something like this:



```
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeM
onitoring()
---

> use products
switched to db products
> db.products.updateMany({},{"$mul":{"price":1.2}})
{ "acknowledged" : true, "matchedCount" : 6, "modifiedCount" : 6 }
> db.products.find()
{ "_id" : ObjectId("5dcb8b49cedc1b3b09db7bc9"), "code" : 100, "name" : "American
o", "price" : 150 }
{ "_id" : ObjectId("5dcb8b49cedc1b3b09db7bca"), "code" : 200, "name" : "Brewed C
offee", "price" : 132 }
{ "_id" : ObjectId("5dcb8b49cedc1b3b09db7bcb"), "code" : 300, "name" : "Cappucci
no", "price" : 144 }
{ "_id" : ObjectId("5dcb8b49cedc1b3b09db7bcc"), "code" : 400, "name" : "Espresso
", "price" : 144 }
{ "_id" : ObjectId("5dcb8b49cedc1b3b09db7bcd"), "code" : 500, "name" : "Latte",
"price" : 168 }
{ "_id" : ObjectId("5dcb8b49cedc1b3b09db7bce"), "code" : 600, "name" : "Cold Bre
w", "price" : 240 }
>
```

Note the price changes.

6. Go back to your web browser and access the Products and Product Details pages.

   Do you see the price increases?

Implement Checkout for our E-Commerce App

Exercise: Add checkout functionality

1. Modify the Cart page to include a link for Checkout. This link should only appear if the cart is not empty. The template file **cart.html** should include an anchor tag to checkout similar to the following:

```
{% include "header.html" %}
        <h1>Cart</h1>

            {% if session["cart"] is defined %}
            <table>

<tr><th>Name</th><th>Quantity</th><th>Subtotal</th></tr>
            {% for item in session["cart"].values() %}
                        <tr><td>{{ item["name"] }}</td><td>{{
item["qty"] }}</td><td>{{ item["subtotal"] }}</td></tr>
            {% endfor %}
            <tr><td colspan=2><b>Total</b></td><td><b>{{
session["cart"].values()|sum(attribute="subtotal")
}}</b></td></tr>
            </table>
            <a href="checkout">Checkout</a>
            {% else %}
            <div>Your cart is empty</div>
            {% endif %}

{% include "footer.html" %}
```

2. Under the **templates** folder, create a new file **ordercomplete.html** with the following code:

```
{% include "header.html" %}
        <h1>Order Complete</h1>

            <div>Thank you for your order.</div>
            <a href="/">Shop for new items.</a>

{% include "footer.html" %}
```

3. Create a new file (directly under the **digitalcafe** folder) named **ordermanagement.py** with the contents below. Note that since we will need direct access to the **session** variable, we will need to import session from flask.

```
import database as db
from flask import session
from datetime import datetime

def create_order_from_cart():
    order = {}
    order.setdefault("username",session["user"]["username"])
    order.setdefault("orderdate",datetime.utcnow())
    order_details = []
    cart = session["cart"]
    for key, value in cart.items():
        order_details.append({"code":key,
                              "name":value["name"],
                              "qty":value["qty"],
                              "subtotal":value["subtotal"]})
    order.setdefault("details",order_details)
    db.create_order(order)
```

4. In the file **database.py**, create a new function called `create_order(order)` as follows:

```
def create_order(order):
    orders_coll = order_management_db['orders']
    orders_coll.insert(order)
```

5. In **app.py**, add an import directive for ordermanagement like so:

```
import ordermanagement as om
```

The first few lines of **app.py** should look like this:

```
from flask import Flask,redirect
from flask import render_template
from flask import request
from flask import session
import database as db
import authentication
import ordermanagement as om
```

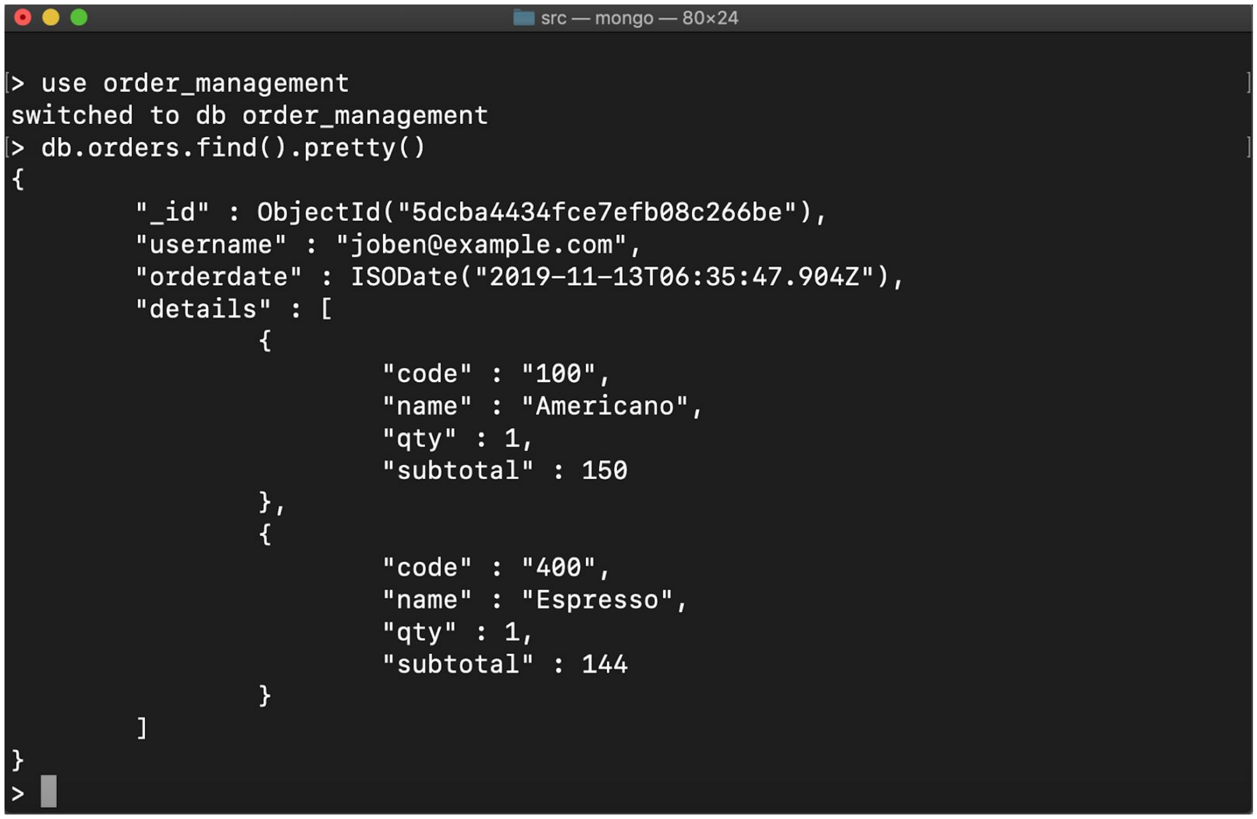
6. Define two new routes in **app.py** for our checkout routine with code found below. Note that we need to clear the cart upon checkout.

```
@app.route('/checkout')
def checkout():
    # clear cart in session memory upon checkout
    om.create_order_from_cart()
    session.pop("cart",None)
    return redirect('/ordercomplete')

@app.route('/ordercomplete')
def ordercomplete():
    return render_template('ordercomplete.html')
```

Save your changes.

7. Go back to your web browser. Go through the motions of placing items in your shopping cart and then check out.
8. To see if your orders have been saved to the database, go to MongoDB from your OS terminal like so and perform a simple query against the **orders** collection:



```
> use order_management
switched to db order_management
> db.orders.find().pretty()
{
        "_id" : ObjectId("5dcba4434fce7efb08c266be"),
        "username" : "joben@example.com",
        "orderdate" : ISODate("2019-11-13T06:35:47.904Z"),
        "details" : [
                {
                        "code" : "100",
                        "name" : "Americano",
                        "qty" : 1,
                        "subtotal" : 150
                },
                {
                        "code" : "400",
                        "name" : "Espresso",
                        "qty" : 1,
                        "subtotal" : 144
                }
        ]
}
>
```

```
> use order_management
switched to db order_management
> db.orders.find().pretty()
{
      "_id" : ObjectId("5dcba4434fce7efb08c266be"),
      "username" : "joben@example.com",
      "orderdate" : ISODate("2019-11-13T06:35:47.904Z"),
          "details" : [
               {
                    "code" : "100",
                    "name" : "Americano",
                    "qty" : 1,
                    "subtotal" : 150
               },
               {
                    "code" : "400",
                    "name" : "Espresso",
                    "qty" : 1,
                    "subtotal" : 144
               }
          ]
     }
```

Take-Home Quiz #5

Total Points: **30**
Due Date: Tuesday, November 26, 6:00PM

This will be a **Group** quiz.

Commit your Flask application folder into your Group GitHub repository. Once done, notify me via email at jbilagan@ateneo.edu.

1. Move branch information to the database. First, insert branch information into the MongoDB collection **branches** in the **products** database.

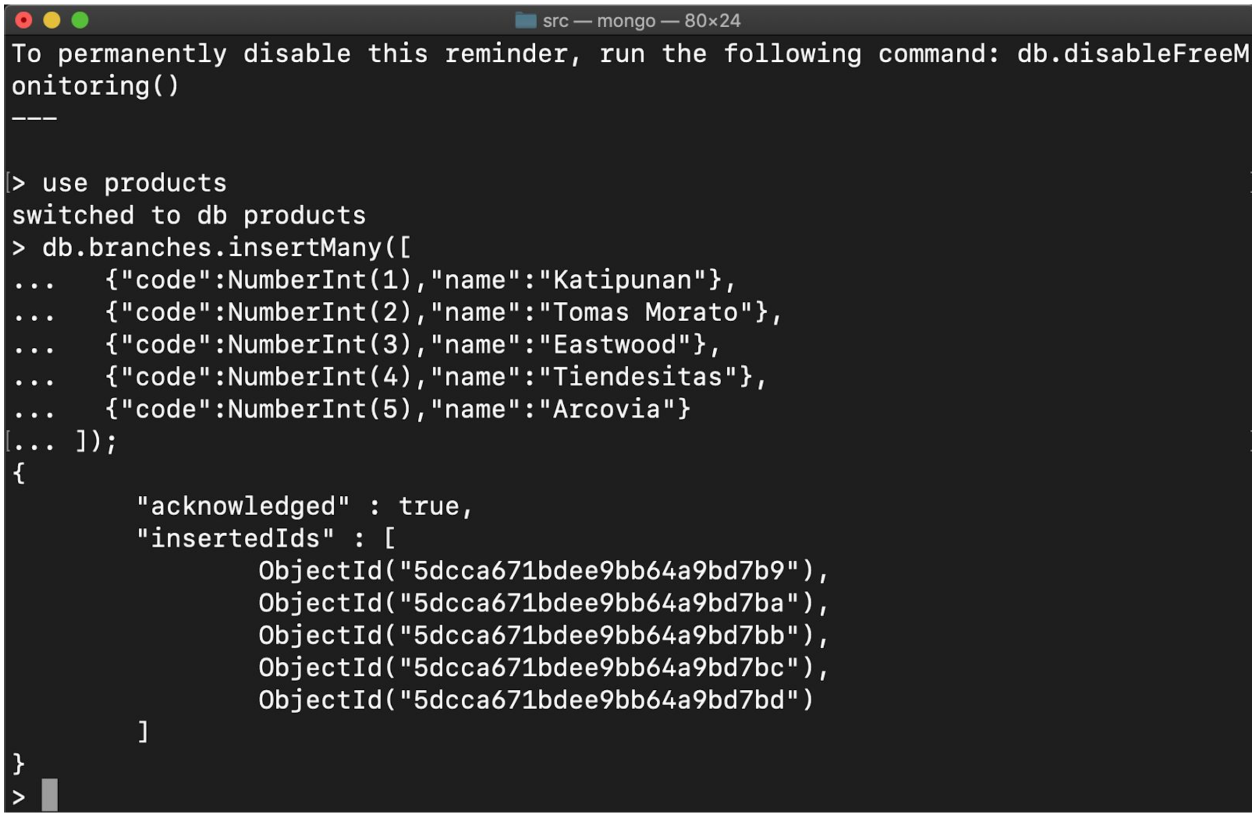
   Use this MongoDB command. IMPORTANT: Make sure you are using the **products** database. Also, if you included more information on branches from previous quizzes, feel free to modify the script.

   ```
   use products

   db.branches.insertMany([
     {"code":1,"name":"Katipunan"},
     {"code":2,"name":"Tomas Morato"},
     {"code":3,"name":"Eastwood"},
   ```

```
  {"code":4,"name":"Tiendesitas"},
  {"code":5,"name":"Arcovia"}
]);
```

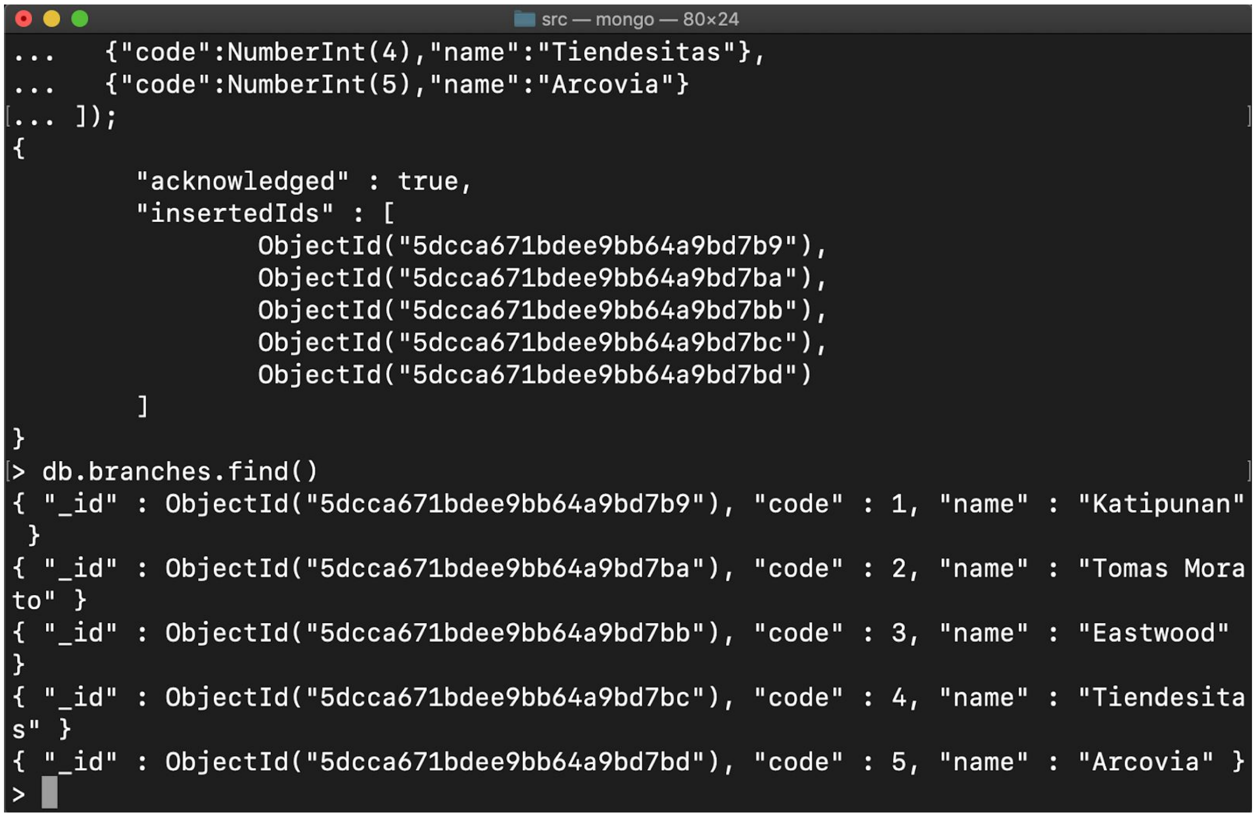
Your MongoDB output should look like this:



```
To permanently disable this reminder, run the following command: db.disableFreeM
onitoring()
---

> use products
switched to db products
> db.branches.insertMany([
...   {"code":NumberInt(1),"name":"Katipunan"},
...   {"code":NumberInt(2),"name":"Tomas Morato"},
...   {"code":NumberInt(3),"name":"Eastwood"},
...   {"code":NumberInt(4),"name":"Tiendesitas"},
...   {"code":NumberInt(5),"name":"Arcovia"}
... ]);
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5dcca671bdee9bb64a9bd7b9"),
                ObjectId("5dcca671bdee9bb64a9bd7ba"),
                ObjectId("5dcca671bdee9bb64a9bd7bb"),
                ObjectId("5dcca671bdee9bb64a9bd7bc"),
                ObjectId("5dcca671bdee9bb64a9bd7bd")
        ]
}
> 
```

Run a query that lists all branches. Take a screenshot of your MongoDB session after.
Your output should look like this: **(5 Points)**



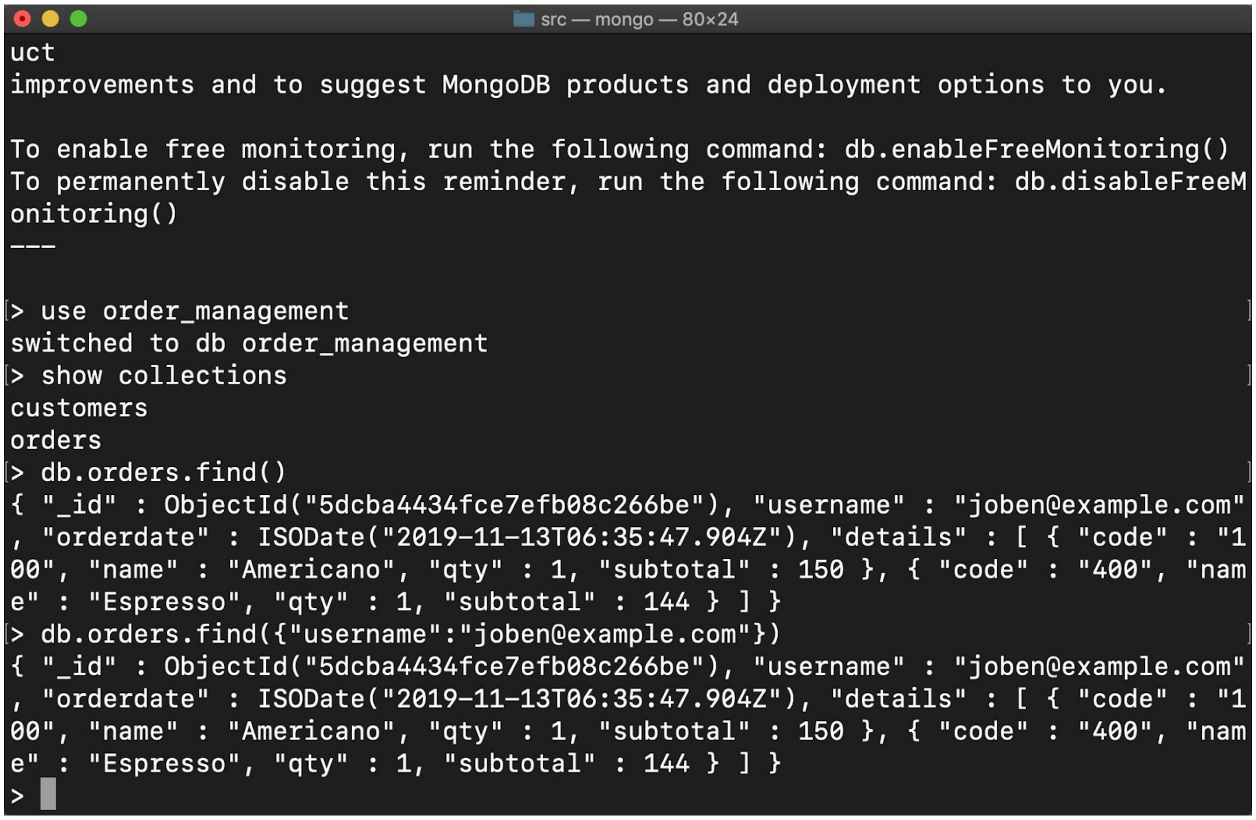
```
...    {"code":NumberInt(4),"name":"Tiendesitas"},
...    {"code":NumberInt(5),"name":"Arcovia"}
... ]);
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5dcca671bdee9bb64a9bd7b9"),
                ObjectId("5dcca671bdee9bb64a9bd7ba"),
                ObjectId("5dcca671bdee9bb64a9bd7bb"),
                ObjectId("5dcca671bdee9bb64a9bd7bc"),
                ObjectId("5dcca671bdee9bb64a9bd7bd")
        ]
}
> db.branches.find()
{ "_id" : ObjectId("5dcca671bdee9bb64a9bd7b9"), "code" : 1, "name" : "Katipunan"
 }
{ "_id" : ObjectId("5dcca671bdee9bb64a9bd7ba"), "code" : 2, "name" : "Tomas Mora
to" }
{ "_id" : ObjectId("5dcca671bdee9bb64a9bd7bb"), "code" : 3, "name" : "Eastwood"
}
{ "_id" : ObjectId("5dcca671bdee9bb64a9bd7bc"), "code" : 4, "name" : "Tiendesita
s" }
{ "_id" : ObjectId("5dcca671bdee9bb64a9bd7bd"), "code" : 5, "name" : "Arcovia" }
> 
```

2. Rewrite your `get_branches()` and `get_branch(code)` functions in **database.py** to access branch information from MongoDB instead. Recall that we created the **branches** collection in the **products** database. Retest your web app. Go to the Branches page to see if everything is still working as before.

   (5 points)

3. Create a page to allow a customer to view past orders. The customer must be logged in to view this page. If there aren't any past orders made, just display a placeholder message, like "No orders have been made yet." Feel free to place your own call-to-action copy.

   Hint: The screen below shows a query to all orders made by joben@example.com:


src — mongo — 80×24

```
uct
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.disableFreeM
onitoring()
---

> use order_management
switched to db order_management
> show collections
customers
orders
> db.orders.find()
{ "_id" : ObjectId("5dcba4434fce7efb08c266be"), "username" : "joben@example.com"
, "orderdate" : ISODate("2019-11-13T06:35:47.904Z"), "details" : [ { "code" : "1
00", "name" : "Americano", "qty" : 1, "subtotal" : 150 }, { "code" : "400", "nam
e" : "Espresso", "qty" : 1, "subtotal" : 144 } ] }
> db.orders.find({"username":"joben@example.com"})
{ "_id" : ObjectId("5dcba4434fce7efb08c266be"), "username" : "joben@example.com"
, "orderdate" : ISODate("2019-11-13T06:35:47.904Z"), "details" : [ { "code" : "1
00", "name" : "Americano", "qty" : 1, "subtotal" : 150 }, { "code" : "400", "nam
e" : "Espresso", "qty" : 1, "subtotal" : 144 } ] }
> 
```

Orders data are stored in the **orders** collection in the **order_management** database. You will have to write the PyMongo equivalent routines to achieve the same query result from within Python and Flask.

Note that past order data are stored in MongoDB, not in session memory. You will, at most, only use session data to get the username of the current user logged in, and you will use that username as part of your query to MongoDB.

(10 Points)

4. Create a page that allows a user to change the password. The screen should ask for the old password and enter the new password twice.
   Check if the new password and the retyped password match. If they don't, then go back to the screen and display an error message. Otherwise, proceed to update the password of the user in MongoDB.

   For reference, use the update example in the previous lesson on MongoDB (which is copied below):

```
db.customers.updateOne({"username":"chums@example.com"},
{"$set":{"password":"n3wp@ssw0rd"}})
```

(10 points)