CS50 Section 9

Flask Week

SQL Review

2 Tables

users

a. id (INTEGER)
b. username (TEXT)
c. hash (TEXT)
d. cash (NUMERIC)

transactions

a. id (INTEGER)
b. user_id (INTEGER)
c. symbol (TEXT)
d. shares (INTEGER)
e. price (NUMERIC)
f. time (DATETIME)

SQL Review

users

a. id (INTEGER)
b. username (TEXT)
c. hash (TEXT)
d. cash (NUMERIC)

id	username	hash	cash
0	phyllis	good_pw	1
1	albert	no_knee	1000
2	luca	math_god	1000000000

SQL Review – Insert

in SQL...

INSERT INTO users (username, hash) VALUES(“phyllis”, “hashed_pw”)

in Python...

SQL Review – Insert

in SQL...

INSERT INTO users (username, hash) VALUES(“phyllis”, “hashed_pw”)

in Python...

```
db.execute("INSERT INTO users (username, hash) VALUES(?, ?)",
                    username_variable, hash_variable)
```

SQL Review – Update

in SQL...

UPDATE users SET cash = cash - (some value) WHERE id = (some id)

in Python...

SQL Review – Update

in SQL...

UPDATE users SET cash = cash - (some value) WHERE id = (some id)

in Python...

```
db.execute("UPDATE users SET cash = cash - ? WHERE id = ?",
                variable_to_subtract_by, id_variable)
```

SQL Review – Select

in SQL...

SELECT * FROM transactions WHERE user_id = (some id)

in Python...

SQL Review – Select

in SQL...

SELECT * FROM transactions WHERE user_id = (some id)

in Python...

db.execute("SELECT * FROM transactions WHERE user_id = ?", id_variable)

What does db.execute really do?

returns a list of dictionaries.

Query: SELECT col1, col2 FROM table1 WHERE condition1

What does db.execute really do?

returns a list of dictionaries.

Query: SELECT col1, col2 FROM table1 WHERE condition1

Resulting list of dictionaries: each row in the SQL table is an item in the list.

[{“col1”: val1, “col2”: val2},

{“col1”: another_val1, “col2”: another_val2},

{“col1”: yet_another_val1, “col2”: yet_another_val2}

,...]

What does db.execute really do?

returns a list of dictionaries.

Query: SELECT username, cash FROM users

id	username	hash	cash
0	phyllis	good_pw	1
1	albert	no_knee	1000
2	luca	math_god	1000000000

What does db.execute really do?

returns a list of dictionaries.

Query: SELECT username, cash FROM users

Resulting list of dictionaries:
[{“username”: “phyllis”, “cash”: 1},
{“username”: “albert”, “cash”: 1000},
{“username”: “luca”, “cash”: 1000000000}]

id	username	hash	cash
0	phyllis	good_pw	1
1	albert	no_knee	1000
2	luca	math_god	1000000000

What does db.execute really do?

returns a list of dictionaries.

Query: SELECT username, cash FROM users
WHERE username = “phyllis”

id	username	hash	cash
0	phyllis	good_pw	1
1	albert	no_knee	1000
2	luca	math_god	1000000000

What does db.execute really do?

returns a list of dictionaries.

Query: SELECT username, cash FROM users
WHERE username = “phyllis”

Resulting list of dictionaries:
[{“username”: “phyllis”, “cash”: 1}]

id	username	hash	cash
0	phyllis	good_pw	1
1	albert	no_knee	1000
2	luca	math_god	1000000000

What does db.execute really do?

returns a list of dictionaries.

results = db.execute(“SELECT username, cash FROM users WHERE username = ‘phyllis’”)

this sets results = [{“username”: “phyllis”, “cash”: 1}] # this is a **list**

results[0] = {“username”: “phyllis”, “cash”: 1} # this is a **dictionary**

results[0][“username”] = “phyllis”

results[0][“cash”] = 1

So that was a lot of backend stuff...

all your SQL queries should be written in a python file, and this python file + database serve as the “backend” of the project.

Let’s move to some frontend stuff! The frontend is where users actually interact with your project.

HTML Forms

```
<form>
    <div>
        <input name="symbol" placeholder="Symbol" type="text">
    </div>
    <div>
        <input min="1" name="shares" placeholder="Shares" type="number">
    </div>
    <button type="submit">Buy</button>
</form>
```

HTML Tables

```
<table>
  <thead>
    <tr>
      <th>Symbol</th>
      <th>Shares</th>
      <th>Price</th>
      <th>Time</th>
    </tr>
  </thead>
  <tbody>
      <tr>
        <td>AAPL</td>
        <td>2</td>
        <td>$5.00</td>
        <td>1:00</td>
      </tr>
  </tbody>
</table>
```

HTML Tables

```
<table>
  <thead>
    <tr>
      <th>Symbol</th>
      <th>Shares</th>
      <th>Price</th>
      <th>Time</th>
    </tr>
  </thead>
  <tbody>
      <tr>
        <td>AAPL</td>
        <td>2</td>
        <td>$5.00</td>
        <td>1:00</td>
      </tr>
  </tbody>
</table>
```

What if there are a lot of rows and we don't want to type them all? What if the values are constantly changing?

How do the HTML and Python files communicate?

1. app.py
2. app.route
3. jinja syntax
4. request.form.get
5. render_template

app.py

Everything lives here!

Flask, by default, looks for an app.py to run, so you always need something named app.py!

app.route

Get started with flask by importing!

```
from flask import Flask
app = Flask(__name__)
```

app.route

Get started with flask by importing!

```
from flask import Flask
app = Flask(__name__)

@app.route("/buy", methods=["GET", "POST"])
def buy():

  if request.method == "POST":
      # when person submits something, do this
  else:
      # when person first clicks this page, do this.
```

jinja syntax

layout.html

```
<html>
    <!-- code that is similar between all html files -->
    <!-- could include the <head></head>, navbar -->

    {% block title %}{% endblock %}

    <!-- code that is similar between all html files -->

    {% block main %}{% endblock %}

    <!-- code that is similar between all html files -->
</html>
```

jinja syntax

history.html

```
{% extends “layout.html” %}

{% block title %}
    History
{% endblock %}

{% block main %}
     <table>
         <!-- table contents →
    </table>
{% endblock %}
```

jinja syntax

```
<table>
    <thead>
        <tr>
            <th>Symbol</th>
            <th>Shares</th>
            <th>Price</th>
            <th>Time</th>
        </tr>
    </thead>
    <tbody>
        {% for data in data_list %}
        <tr>
            <td>{{ data.column1 }}</td>
            <td>{{ data.column2 }}</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

jinja syntax

buy.html (in particular, forms)

```
<form action="/buy" method="post">
    <div>
        <input name="symbol" placeholder="Symbol" type="text">
    </div>
    <div>
        <input min="1" name="shares" placeholder="Shares" type="number">
    </div>
    <button type="submit">Buy</button>
</form>
```

request.form.get

```
<form action=”/buy” method=”post”>
    <div>
        <input name="symbol" placeholder="Symbol" type="text">
    </div>
    <button type="submit">Buy</button>
</form>
```

request.form.get

```
<form action=”/buy” method=”post”>
    <div>
        <input name="symbol" placeholder="Symbol" type="text">
    </div>
    <button type="submit">Buy</button>
</form>
```

in app.py...

```
def buy():
    if request.method == “POST”:
        # use the name attribute of the input tag to get the value the user has inputted
        if not request.form.get(“symbol”): # validate input, make sure it is not empty
            return (whatever you’re supposed to return)
```

request.form.get

May consider doing something like this, when combining SQL with request.form.get...

```
db.execute("INSERT INTO table (col1, col2) VALUES(?, ?)",
                    request.form.get(<some input by user>),
                    request.form.get(<some other input by user>))
```

render_template

```
<table>
    <thead>
        <tr>
            <th>Symbol</th>
            <th>Shares</th>
            <th>Price</th>
            <th>Time</th>
        </tr>
    </thead>
    <tbody>
        {% for data in data_list %}
        <tr>
            <td>{{ data.column1 }}</td>
            <td>{{ data.column2 }}</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

render_template

We get data to use in our HTML files from app.py.

Want to give **data_list** to history.html?

Do **return render_template(“history.html”, data_list=<desired_variable>)**

render_template

We get data to use in our HTML files from app.py.

Want to give **data_list** to history.html?
Do **return render_template(“history.html”, data_list=<desired_variable>)**

For example...

```
@app.route(“/history”)
def history():
    list = db.execute(<my SQL query>)

    # list is the variable used in app.py, data_list is the variable used in history.html
    return render_template(“history.html”, data_list=list)
```

render_template

Don’t want to pass in any data?

Do **return render_template(“<page>.html”)**, same as before but without arguments.

render_template

Don’t want to pass in any data?

Do **return render_template(“<page>.html”)**, same as before but without arguments.

Want to pass in a lot of data?

Do **return render_template(“<page>.html”, data1 = data1_var, data2 = data2_var)**,

same as before but with multiple arguments.

render_template

Don’t want to pass in any data?

Do **return render_template(“<page>.html”)**, same as before but without arguments.

Want to pass in a lot of data?

Do **return render_template(“<page>.html”, data1 = data1_var, data2 = data2_var)**,

same as before but with multiple arguments.

For example...

```
@app.route(“/buy”, methods=[“GET”, “POST”])
def buy():
    if request.method == “POST”:
        # process some data
    else: # just show them a form!
            return render_template(“buy.html”)
```

useful things

Want to return your user back to the index page?
Instead of returning the index.html template, we **return redirect(“/”)**. Why?

For a form with multiple options, you can do a for loop, creating as many options as you need. In order to see which option the user selected after submitting the form, you want to set the **value attribute** of the option.

<option value=“hi”> hi option </option>

Selecting this option inside a form with name = “form-name” will set request.form.get(“symbol”) equal to “hi”.

Other Hints + Debugging Tips

Refer to the Finance + Exam Hints document.

Lab

app.py

```
if request.method == "POST":
    name = request.form.get("name")
    month = ""
    day = ""

    db.execute("INSERT into birthdays (name, month, day)
                (?,?,?)", name, month, day)
    return redirect("/")
else:
    #

bdays = db.execute("SELECT * FROM birthdays")

return render_template("index.html", birthdays = bdays)
```

Within index.html, use birthdays somehow!

index.html

First, create a form

Create a table

```
{% for birthday in birthdays %}
<td> birthday.name </td>
```