CSCI E-50 WEEK 9

TERESA LEE
[teresa.lee@alumni.utoronto.ca]
October 29, 2017

Agenda

- Flask - decorators and routes
- Jinja/templating
- SQL

Flask

- **Python** is not just used for command-line programming. It also contains native functionality to support networking and more, enabling site backends to be written in it.

- Web frameworks make this easier, instead of implementing every detail. Some of the most popular include Django, Pyramid, and Flask. In this course, we use Flask.

```
from flask import Flask
from datetime import datetime
from pytz import timezone

app = Flask(__name__)

@app.route("/")
def time():
    now = datetime.now(timezone('America/New_York'))
    return f"The current date and time is {now}."
```

- It’s rather easy to get started using Flask within CS50 IDE.

```
from flask import Flask
```

- After importing the Flask module, we need to initiate a Flask application.

```
app = Flask(__name__)
```

- From there, we need only write functions to define the behavior of our application.

```
def time():
```

```
def index():
   return "You are at the index page!"


def sample():
   return "You are on the sample page!"
```

```
@app.route("/")
def index():
    return "You are at the index page!"

@app.route("/sample")
def sample():
    return "You are on the sample page!"
```

- The lines just added are called ***decorators***. They are used in Flask to associate the behavior of a particular function with a particular URL. They also have more general use in Python, but that goes beyond this course.

- Data can be passed in via URLs, much like using HTTP GET.

```
@app.route("/show/<number>")
def show(number):
   return f"You passed in {number}."
```

- Data can be passed in via HTML forms as well, as via HTTP POST, but we need to clarify things a bit more for Flask if we do:

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    # if the username field of form missing
    if not request.form.get("username"):
        return apology("need a username!")
```

- Or we could vary our function's behavior depending on what kind of HTTP request we got.

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == "POST":
        # do one thing
    else:
        # do a different thing
```

- Flask has a number of different built-in methods you'll find useful, particularly for Problem Set 7 but also perhaps for Problem Set 6.

```
url_for()

redirect()

session()

render_template()
```

- For more information on Flask:

  `http://flask.pocoo.org/docs/0.12/quickstart`

- For more information on Jinja, the Python templating engine:

  `http://jinja.pocoo.org`

Jinja

- The folks behind Flask are also the developers of a popular *templating engine* called **Jinja**.

- The primary motivation for a templating engine like this is for us to be able to procedurally generate HTML based on the value of some variable(s) in our programs.

- This allows us to mix Python and HTML!

- Let’s build a simple web application to create a multiplication table, where the size of the table is determined by the user, and we generate the HTML for the table based on the size the user wants.

- We will be making use of Flask’s `render_template()` method rather extensively in this application.

- First, the basic app.

- First, the basic app.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/")
def mult_table():
```

- Let’s start by just displaying a simple form to the user.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/")
def mult_table():
```

- Let’s start by just displaying a simple form to the user.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/")
def mult_table():
    return render_template("form.html")
```

- By default, Flask will look in the `templates/` directory to try to find a template with that name, so first we create that subdirectory, and then we toss a very simple form in there. (No Jinja in this one, since it’s static.)

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Multiplication Table
        </title>
    </head>
    <body>
        <form action="/" method="post">
            <input name="size" type="number" placeholder="dimension"/>
            <input name="submit" type="submit" />
        </form>
    </body>
</html>
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Multiplication Table
        </title>
    </head>
    <body>
        <form action="/" method="post">
            <input name="size" type="number" placeholder="dimension"/>
            <input name="submit" type="submit" />
        </form>
    </body>
</html>
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Multiplication Table
        </title>
    </head>
    <body>
        <form action="/" method="post">
            <input name="size" type="number" placeholder="dimension"/>
            <input name="submit" type="submit" />
        </form>
    </body>
</html>
```

- Okay, so now we're good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
```

- Okay, so now we're good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
    # our form is set up to submit via POST
    elif request.method == "POST":
        return render_template("table.html")
```

- Time to create another template. We know that HTML tables consist of <tr> tags for each row, consisting of a set of <td> tags for columns. So that lets us craft the super-basic idea for a template.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>

          <tr>

              <td>

              </td>

          </tr>

        </table>
  </body>
</html>
```

- Okay, so now we're good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
    # our form is set up to submit via POST
    elif request.method == "POST":
        return render_template("table.html", dim=request.form.get("size"))
```

- Okay, so now we're good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
    # our form is set up to submit via POST
    elif request.method == "POST":
        return render_template("table.html", dim=request.form.get("size"))
```

- Time to create another template. We know that HTML tables consist of <tr> tags for each row, consisting of a set of <td> tags for columns. So that lets us craft the super-basic idea for a template.

- Next, we need to somehow convey to this template the number of rows the user supplied. We can do this by altering our call to `render_template()`.

- Effectively, “dim” is now a variable within Jinja, and Jinja allows us to use a Python-like syntax interspersed within our HTML.

- Jinja is introduced in the template in one of two ways:
  - `{% ... %}`
    - These delimiters indicate that what is between them is control-flow or logic.
  - `{{ … }}`
    - These delimiters indicate that what is between them should be evaluated and effectively “printed” as HTML.

- Exactly what you can do with Jinja is an exercise for home, but its syntax is generally Python like, with a couple of quirks due to the way it is interspersed with HTML. Let’s see how we can use it to generate the HTML we need.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>

          <tr>

              <td>

              </td>

          </tr>

        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
        // loop to repeat “dim” times (“dim” # of rows)
          <tr>
        // loop to repeat “dim” times (“dim” # of columns)
              <td>
        // print out that value of the cell between <td>s
              </td>

          </tr>

        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim) %}
          <tr>
        // loop to repeat “dim” times (“dim” # of columns)
              <td>
        // print out that value of the cell between <td>s
              </td>

          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim) %}
          <tr>
              {% for j in range(dim) %}
              <td>
        // print out that value of the cell between <td>s
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim) %}
          <tr>
              {% for j in range(dim) %}
              <td>
                 {{ (i + 1) * (j + 1) }}
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim|int) %}
          <tr>
              {% for j in range(dim|int) %}
              <td>
                 {{ (i + 1) * (j + 1) }}
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table border=1>
          {% for i in range(dim|int) %}
          <tr>
              {% for j in range(dim|int) %}
              <td>
                 {{ (i + 1) * (j + 1) }}
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

20 Submit

20 Submit

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108	114	120
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152	160
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135	144	153	162	171	180
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	192	204	216	228	240
13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
14	28	42	56	70	84	98	112	126	140	154	168	182	196	210	224	238	252	266	280
15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255	270	285	300
16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	272	288	304	320
17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
18	36	54	72	90	108	126	144	162	180	198	216	234	252	270	288	306	324	342	360
19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	320	340	360	380	400

SQL

- In order to build increasingly complex websites, we depend on a **database** to store information long-term. The simplest form of a database with which we are all likely familiar is a basic spreadsheet, organized into rows and columns, tabs (tables), and individual files (databases).

- SQL is a programming language whose purpose is to *query* databases (perform operations on them).

- After you create a database, you create one or more **tables**.
- For each table, you specify all of the **columns** in the table.
- When new information is added to the database, the new information (typically) goes into a new **row**.
- There are many data types that can be stored in a SQL database. This is just a small sample.

INT	SMALLINT	TINYINT	MEDIUMINT	BIGINT
DECIMAL	FLOAT	BIT	DATE	TIME
DATETIME	TIMESTAMP	CHAR	VARCHAR	BINARY
BLOB	TEXT	ENUM	GEOMETRY	LINESTRING

- After you create a database, you create one or more **tables**.
- For each table, you specify all of the **columns** in the table.
- When new information is added to the database, the new information (typically) goes into a new **row**.
- In SQLite, which we'll use in this course, we can consolidate these various datatypes into a few more general classes (though underlying types still exist)

NULL	INTEGER	REAL	TEXT	BLOB

- Another consideration is choosing a column to be a **primary key**, guaranteed to be unique across rows. A good primary key makes subsequent table operations much easier.
    - You can also have a *joint primary key*, a combination of two or more columns where the combination is guaranteed to be unique.

- SQL is a programming language, but it has a limited vocabulary that we'll use.

- Another consideration is choosing a column to be a **primary key**, guaranteed to be unique across rows. A good primary key makes subsequent table operations much easier.
    - You can also have a *joint primary key*, a combination of two or more columns where the combination is guaranteed to be unique.

- SQL is a programming language, but it has a limited vocabulary that we'll use.

```
INSERT
SELECT
UPDATE
DELETE
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

- An `INSERT` query adds information to a table.

```
INSERT INTO
<table>
(<columns>)
VALUES
(<values>)
```

- An `INSERT` query adds information to a table.

```
INSERT INTO
users
(username, password, fullname)
VALUES
('newman', 'USMAIL', 'Newman')
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

- When defining the column that ultimately is your primary key, it’s usually a good idea for that column to be an integer.

- Moreover, you can configure that column to **autoincrement**, so it will pre-populate that column for you automatically when rows are added, eliminating the risk that you’ll accidentally try to insert something with a duplicate value.

- An `INSERT` query adds information to a table.

```
INSERT INTO
moms
(username, mother)
VALUES
('kramer', 'Babs Kramer')
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

- A `SELECT` query extracts information from a table.

```
SELECT
<columns>
FROM
<table>
WHERE
<predicate>
```

- A `SELECT` query extracts information from a table.

```
SELECT
idnum, fullname
FROM
users
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

- A `SELECT` query extracts information from a table.

```
SELECT
password
FROM
users
WHERE
idnum < 12
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

- A `SELECT` query extracts information from a table.

```
SELECT
*
FROM
moms
WHERE
username = 'jerry'
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

- Databases empower us to organize information into tables efficiently.

- We don’t always need to store every possible relevant piece of information in the same table, but rather we can use relationships across tables to connect all the pieces of data we need.

- Let’s imagine we need to get a user’s full name (from the *users* table) and their mother’s name (from the *moms* table).

- A `SELECT (JOIN)` query extracts information from multiple tables.

```
SELECT
<columns>
FROM
<table1>
JOIN
<table2>
ON
<predicate>
```

- A `SELECT (JOIN)` query extracts information from multiple tables.

```
SELECT
users.fullname, moms.mother
FROM
users
JOIN
moms
ON
users.username = moms.username
```

- A `SELECT (JOIN)` query extracts information from multiple tables.

```
SELECT
users.fullname, moms.mother
FROM
users
JOIN
moms
ON
users.username = moms.username
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users & moms

users.idnum	users.username moms.username	users.password	users.fullname	moms.mother
10	jerry	fus!ll!	Jerry Seinfeld	Helen Seinfeld
11	gcostanza	b0sc0	George Costanza	Estelle Costanza

users & moms

users.idnum	users.username moms.username	users.password	users.fullname	moms.mother
10	jerry	fus!ll!	Jerry Seinfeld	Helen Seinfeld
11	gcostanza	b0sc0	George Costanza	Estelle Costanza

- An `UPDATE` query modifies information in a table.

```
UPDATE
<table>
SET
<column> = <value>
WHERE
<predicate>
```

- An `UPDATE` query modifies information in a table.

```
UPDATE
users
SET
password = 'yadayada'
WHERE
idnum = 10
```

users

idnum	username	password	fullname
10	jerry	yadayada	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

- A `DELETE` query removes information from a table.

```
DELETE FROM
<table>
WHERE
<predicate>
```

- A `DELETE` query removes information from a table.

```
DELETE FROM
users
WHERE
username = 'newman'
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer