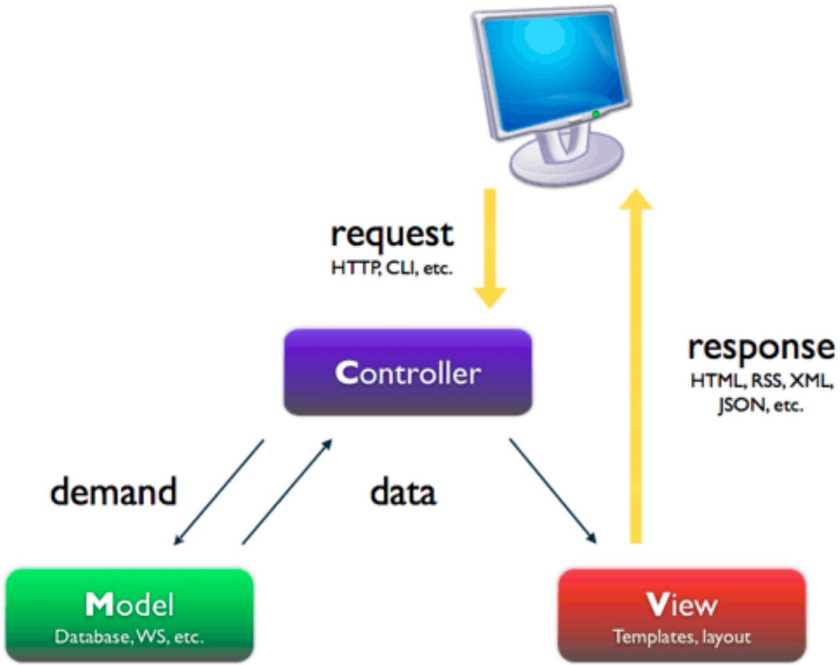
CSCI E-50 WEEK 9

TERESA LEE
[teresa.lee@alumni.utoronto.ca]
October 30, 2017

Agenda

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


request
HTTP, CLI, etc.
Controller
response
HTML, RSS, XML,
JSON, etc.
demand
data
Model
Database, WS, etc.
View
Templates, layout

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():
```

Let’s Look at an Example

application.py

```
@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.

Let’s Look at an Example

decorator.py

- 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()
```

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

- 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"))
```

- 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.

```
<!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>
```

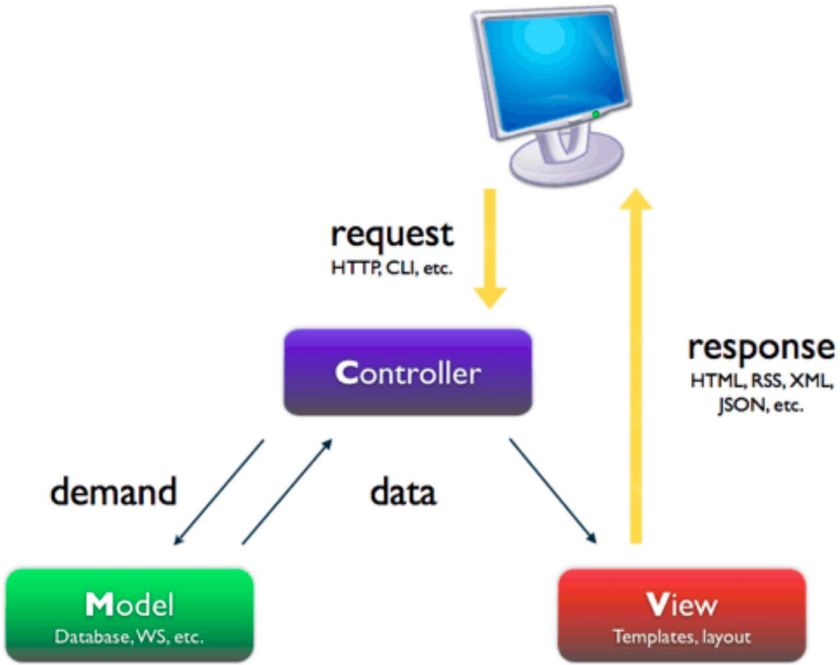
```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim) %}
          <tr>
        // loop to repeat “dim” times (“dim” # of columns)
              <td>
        // print out that value of the cell between <td>s
              </td>

          </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>
```


request
HTTP, CLI, etc.
Controller
response
HTML, RSS, XML,
JSON, etc.
demand
data
Model
Database, WS, etc.
View
Templates, layout

SQL

Often times, in order for us to build the most functional website we can, we depend on a database to store information.

Database: a hierarchically organized set of tables, each of which contains a set of rows and columns.

SQL

Structured Query Language

A programming language which is used for managing and manipulating data in relational databases.

In CS50, you will use a SQLite database.

SQLlite databases are stored in a `.db` file.

In order to insert, delete or modify the data in the database you will use SQL queries.

4 basic queries:

`UPDATE`

`INSERT INTO`

`SELECT`

`DELETE`

SQL: Creating a Table

- Create Fields
- Choose data type from options such as `INTEGER`, `REAL` (a floating-point number), `TEXT` (a string), `BLOB` (binary data), `NUMERIC` (numbers that can be either integers or floats), `BOOLEAN`, `DATETIME` (to store dates and times in a standard way).
- `Primary Key` indicates whether that field is the key that uniquely identifies all the rows in that table. But it's possible that two people share the same name and dorm, so we won't check that.
- `Autoincrement` allows us to have an integer field that increments itself every time a new row is added (like for an ID number), so we'll leave that unchecked too.
- `Not NULL` means that the field cannot be empty, or null. Since we want both fields to filled for every row, we'll check this for both.
- Specify some `Default Value` if no value is provided, but we won't use that either.

SQL: UPDATE

`UPDATE`: Update data in a database table

```
# update table, changing values in particular columns
UPDATE table SET col1 = val1, col2 = val2, ...
```

```
# update table, changing col1 to val1 where "name" equals "identifier"
UPDATE table SET col1 = val1 WHERE identifier = "name"
```

SQL: INSERT INTO

`INSERT INTO`: Insert certain values into a table

```
 # insert into table the row of values
INSERT INTO table VALUES values
```

```
# insert into table under columns col1 & col2, val1 & val2
INSERT INTO table (col1, col2) VALUES (val1, val2)
```

SQL: SELECT

`SELECT`: Select values to view

```
# select a column from table to compare/ view

SELECT col FROM table WHERE col = "identifier"
```

```
# select all columns from a table

SELECT * FROM table
```

SQL: DELETE

`DELETE`: Delete from table

```
# delete a row from table

DELETE FROM table WHERE col = "identifier"
```

SQL

To execute a SQL query in Flask you can use the provided `db.execute()` method.

```
# query database for username

rows = db.execute("SELECT * FROM users WHERE username = :username",
username=request.form.get("username"))
```

To manipulate a database directly (this will come in handy when setting your database up) you will use phpliteadmin, which is a web-based interface.

To run phpliteadmin run the following command in the IDE, substituting `mydatabase.db` for the name of your database file:

```
$ phpliteadmin finance.db
Running phpLiteAdmin at https://ide50-teresa-lee.cs50.io:8081/?pin=qq7KjylNOzo9fQnh
Exit with ctrl-c...
```

Let’s Look at an Example

users

userid	username	password	fullname
1	jerry	fus!ll!	Jerry Seinfeld
2	gcostanza	b0sc0	George Costanza

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

Create Moms table

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

```
Insert 'kramer', 'Babs Kramer'
```

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

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

users

userid	username	password	fullname
1	jerry	fus!ll!	Jerry Seinfeld
2	gcostanza	b0sc0	George Costanza
3	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

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

```
SELECT
userid, fullname
FROM
users
```

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

```
SELECT
password
FROM
users
WHERE
Userid < 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

userid	username	password	fullname
1	jerry	fus!ll!	Jerry Seinfeld
2	gcostanza	b0sc0	George Costanza
3	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