Web Database Applications

CSCI 220: Database Management and Systems Design

START RECORDING

Practice Quiz: Docker

- Discuss with a neighbor

- First, explain the difference between Docker images and containers
- Next, describe what each of the following commands does:
  - docker run
  - docker stop
  - docker start
  - docker rm
  - docker exec
  - docker logs
  - docker ps

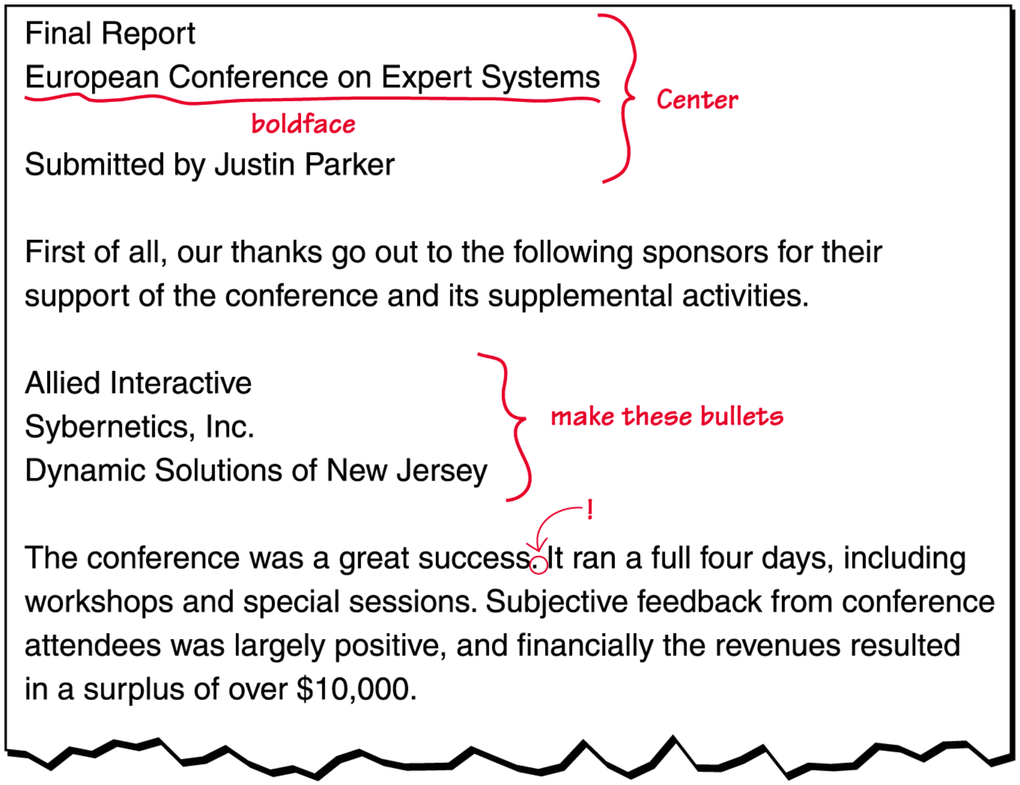
Today you will learn…

- How to build dynamic web pages using databases and HTML
- You will **not** learn:
  - How to create secure or maintainable web applications!

Hyper Text Markup Language (HTML)

HyperText Markup Language (HTML)

- **HyperText:** Links allow instant access to related documents
  - A revolutionary idea at the time!
- **Markup language:** The language annotates content to describe how to render it



Final Report
European Conference on Expert Systems

Submitted by Justin Parker

First of all, our thanks go out to the following sponsors for their support of the conference and its supplemental activities.

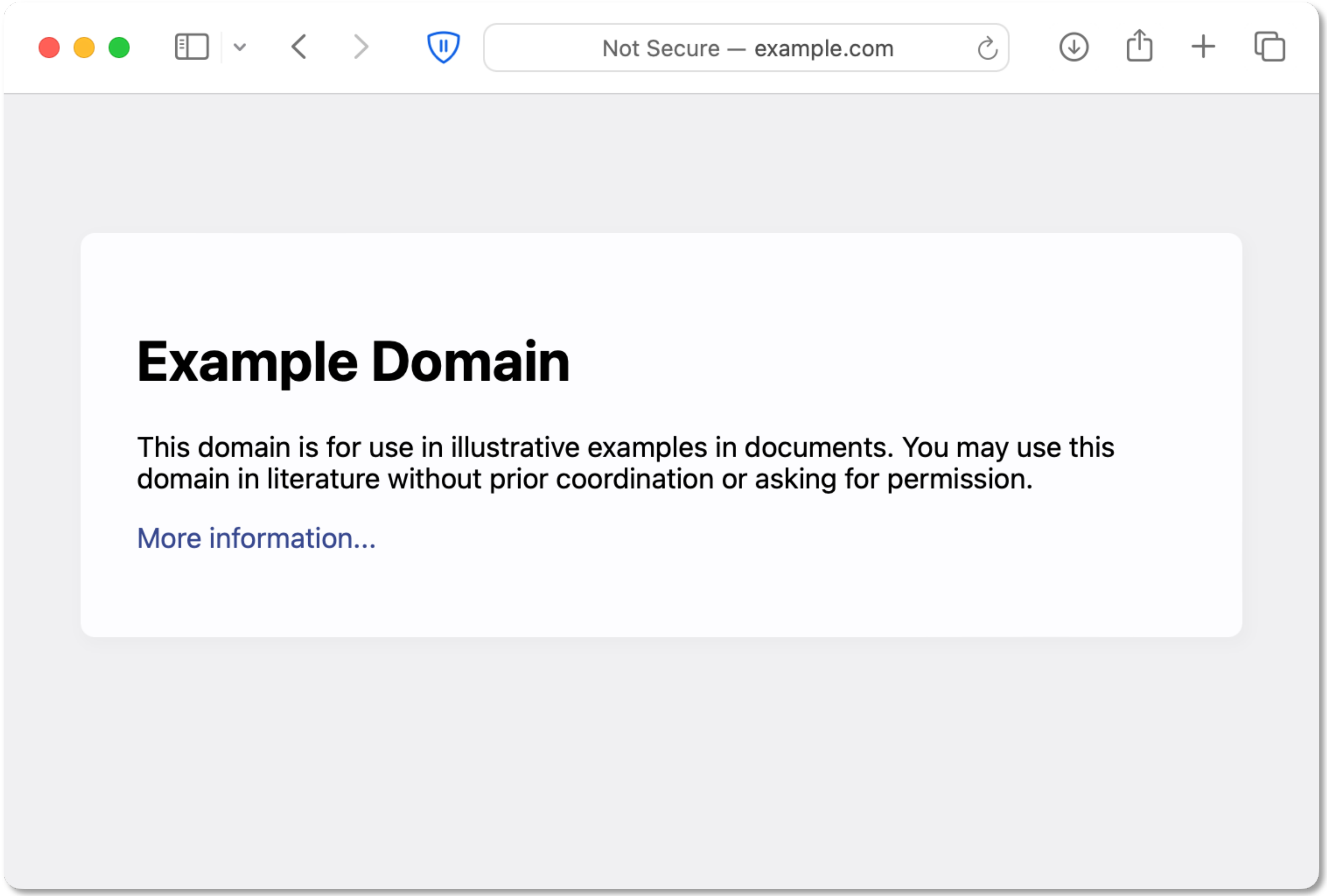
Allied Interactive
Sybernetics, Inc.
Dynamic Solutions of New Jersey

The conference was a great success. It ran a full four days, including workshops and special sessions. Subjective feedback from conference attendees was largely positive, and financially the revenues resulted in a surplus of over $10,000.

HTML Source Code

```
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
    ...
</style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

Decoded and Rendered



Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...

HTML History

- First standardized in 1993
  - Continuously updated since then
- Plain text source code
- HTML tags define HTML elements

HTML Tags

```
<html></html>
<head></head>
<body></body>
<title></title>
<!-- Comment -->
```

```
<!doctype html>
<html>
<head><title>Hello World!</title></head>
<body>
This is my first web page.
<!-- Under construction -->
</body>
</html>
```

HTML Tags

`<p></p>`	Define a paragraph
`<br>`	Create a line break
`<h1></h1>`	Create a heading (also, try <h2>,…)
`<b></b>`	Create bold text
`<i></i>`	Create italicized text

```
<!doctype html>
<html>
<head><title>Hello World!</title></head>
<body>
<h1>Hello World!</h1>
<p>This is my first web page.</p>
</body>
</html>
```

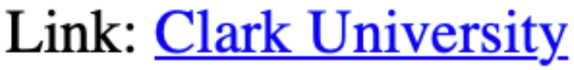
Anchor Tag

Hyperlinks are created using the `<a>` tag.

The href property gives a URL for the link.

Example:

```
Link: <a href="https://www.clarku.edu/">Clark University</a>
```


Link: Clark University

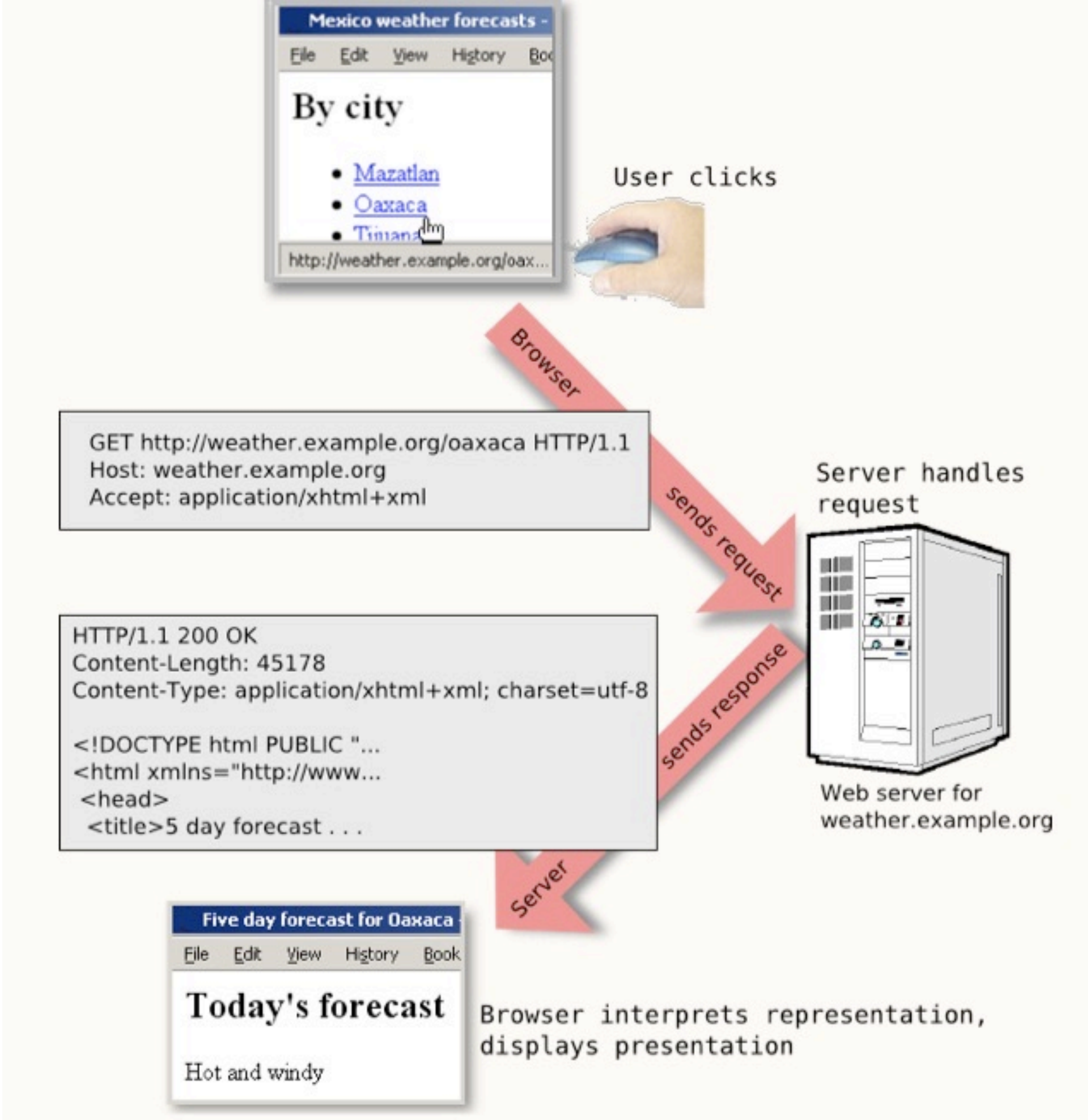
Hyper Text Transfer Protocol (HTTP)

HyperText Transfer Protocol (HTTP)

- HTTP specifies requests and responses between clients and servers
  - For reliable transport, TCP/IP is typically used
  - The client (called a browser) connects to a web server, by default on port 80 (HTTP) or port 443 (HTTPS)
- Any kind of data can be transferred:
  - HTML
  - JSON data
  - Images
  - Video streams
  - …

HTTP

HyperText Transfer Protocol
http://www.ltg.ed.ac.uk/~ht/WhatAreURIs/


Mexico weather forecasts -
File Edit View History Boo
By city
• Mazatlan
• Oaxaca
• Tijuana
http://weather.example.org/oax...
User clicks
Browser
sends request
Server handles request
Web server for weather.example.org
sends response
Server
Five day forecast for Oaxaca
File Edit View History Book
Today's forecast
Hot and windy
Browser interprets representation, displays presentation

```
GET http://weather.example.org/oaxaca HTTP/1.1
Host: weather.example.org
Accept: application/xhtml+xml
```

```
HTTP/1.1 200 OK
Content-Length: 45178
Content-Type: application/xhtml+xml; charset=utf-8

<!DOCTYPE html PUBLIC "...
<html xmlns="http://www...
 <head>
  <title>5 day forecast . . .
```

HTTP Request Messages

- An HTTP request has two main parts: a **method** (action) and a **URI** (uniform resource identifier) upon which to perform the action.
- HTTP methods are OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT.
- The URI must specify the path to a resource (e.g. a file or directory).

HTTP Request Example

Suppose the user types the URL:
http://www.clarku.edu/

The browser will find the IP address for the host www.clarku.edu and create a TCP/IP connection to it on port 80

HTTP Request Example

The HTTP request message looks like:

```
GET / HTTP/1.1

Host: www.clarku.edu:80

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-
US; rv:1.8.1.5) Gecko/20070713 Firefox/2.0.0.5

Accept:
text/xml,application/xml,application/xhtml+xml,text/html
;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive
```

HTTP Response Messages

An HTTP response has three main parts: a **status**, some **headers**, and the **message body**.

HTTP status codes include OK, FORBIDDEN, NOT FOUND, INTERNAL SERVER ERROR, …

For successful requests, the status of OK is followed by headers which explain how to decode the message body.

HTTP Response Example

An HTTP response message looks like:

```
HTTP/1.1 200 OK
Date: Wed, 01 Aug 2007 17:33:41 GMT
Server: Apache/1.3.37 (Unix) mod_ssl/2.8.28
OpenSSL/0.9.7l
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

77b
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
...
```

Content Type

Notice the header field called **Content-Type**

```
HTTP/1.1 200 OK
Date: Wed, 01 Aug 2007 17:33:41 GMT
Server: Apache/1.3.37 (Unix) mod_ssl/2.8.28
OpenSSL/0.9.7l
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
...
```

This indicates the that the message can be interpreted as encoded in plain text and/or HTML.

Web Application Programming

Dynamic Web Pages

- The earliest web pages were static – fixed content which did not change unless edited by a human editor.
- A dynamic web page is generated by a computer program, based on some transaction between client and server.
  - Example: shopping at amazon.com

Dynamic Web Pages

There are many ways to develop dynamic web pages.

- **Java Servlet:** A server-side Java application which processes HTTP requests and generates HTTP responses
- **Java Server Pages:** A programming language which mixes HTML tags, plain text, and Java code scriptlets

Dynamic Web Pages

- **PHP: Hypertext Preprocessor:** A programming language embedded in HTML documents
- **Active Server Pages:** Microsoft’s server side language, based on Visual Basic Script
- **JavaScript frontend and backend:** Pages built dynamically in users' browsers using JavaScript, served by a backend also running JavaScript code
- **CGI Scripting:** A script (Python, Bash, etc.) can read HTTP headers, and use STDOUT to generate HTML output

Dynamic Web Pages

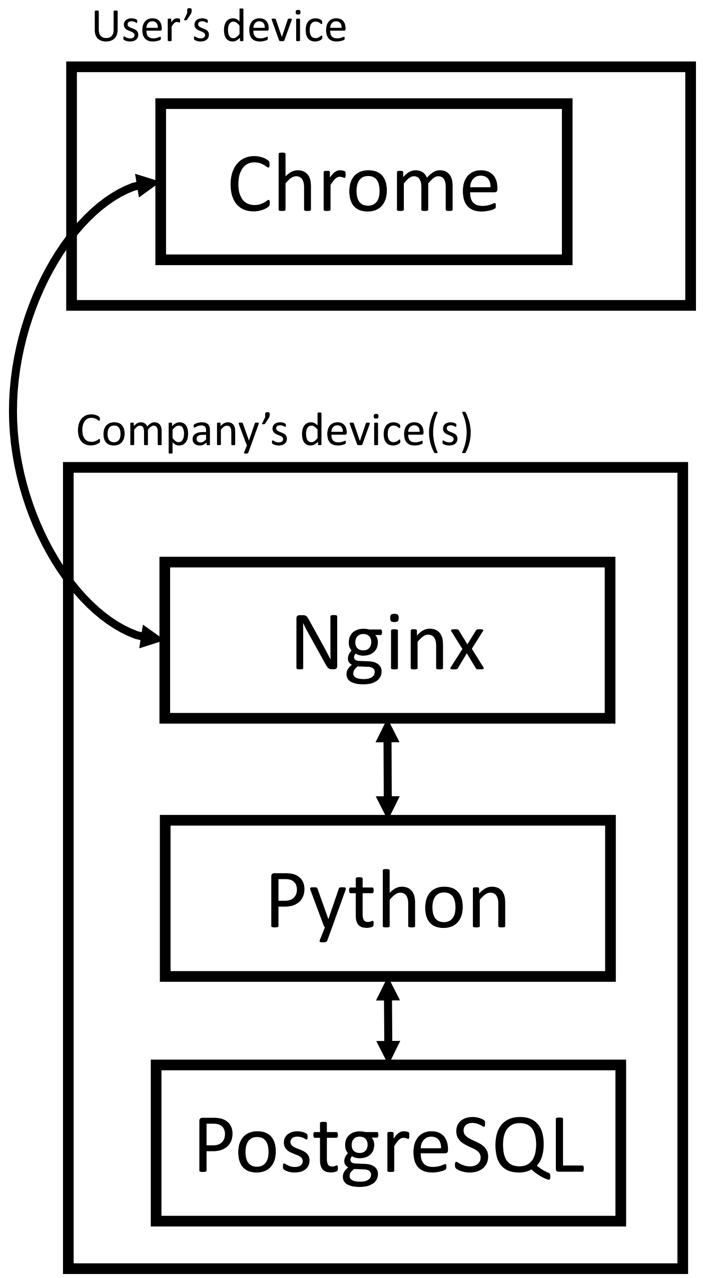
- **WSGI Applications:** One step up from CGI scripting, includes higher-level protocols for communication between a web server and application code
- **Django:** High-level framework that encourages a model-view-controller design pattern, secure coding. Python backend.
  - Similar to Ruby on Rails

Server Stacks

- **Operating system:** Windows, Linux, macOS (less common), etc.
- **Web server:** Apache, nginx, etc.
- **Application code:** Java server pages, PHP, node.js, Django, etc.
- **Database:** MySQL, PostgreSQL, SQLite, MariaDB, etc.

Example Stack

- Web browser (Chrome) requests pages and renders the application's graphics
- Web server (Nginx) passes data between the browser and the application code
- Application code (Python) builds the HTML for dynamic pages, based on data from the database
- The database (PostgreSQL) manages physical storage of the data



WSGI Example with Python

hello_world.py

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type','text/html')])
    return [b"Hello World"]
```

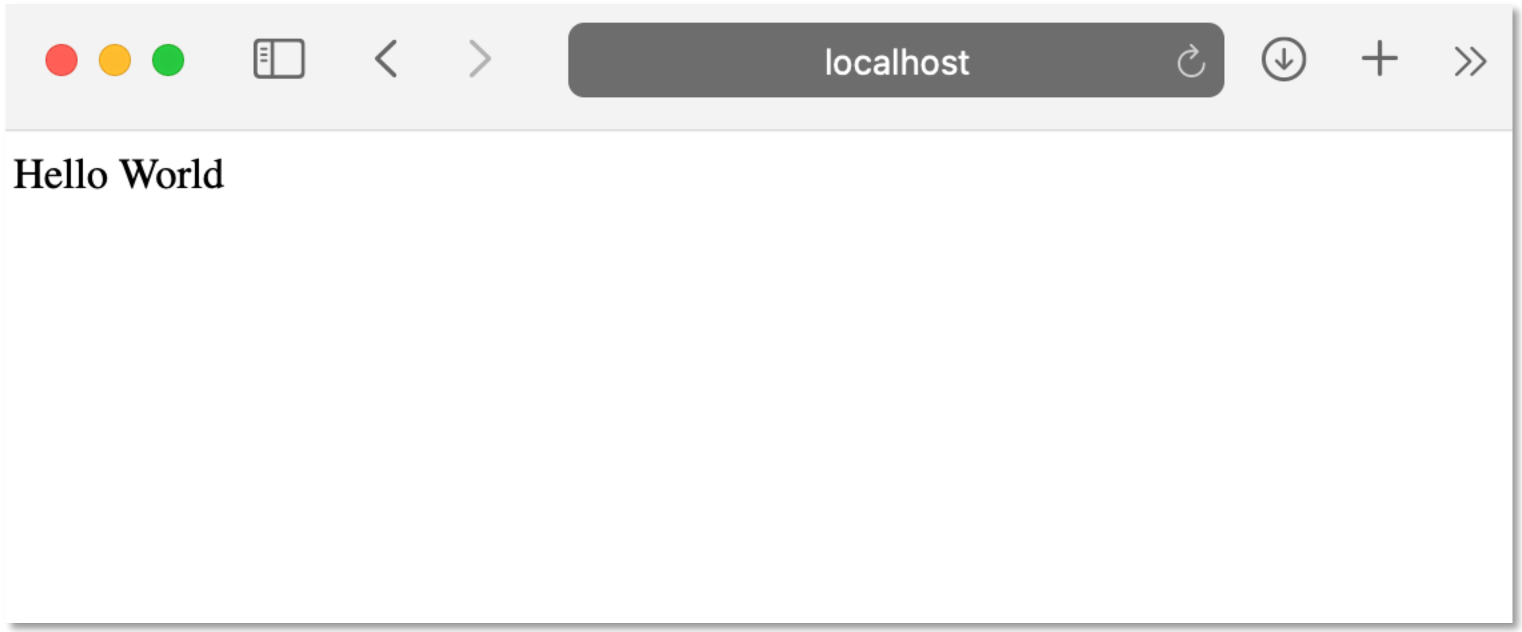
Run using:

```
uwsgi \
  --socket /var/www/uwsgi/uwsgi.sock \
  --chmod-socket=666 \
  --workers 4 \
  --logto /uwsgi/uwsgi.log \
  --wsgi-file /uwsgi/hello_world.py
```

NGINX Configuration Example

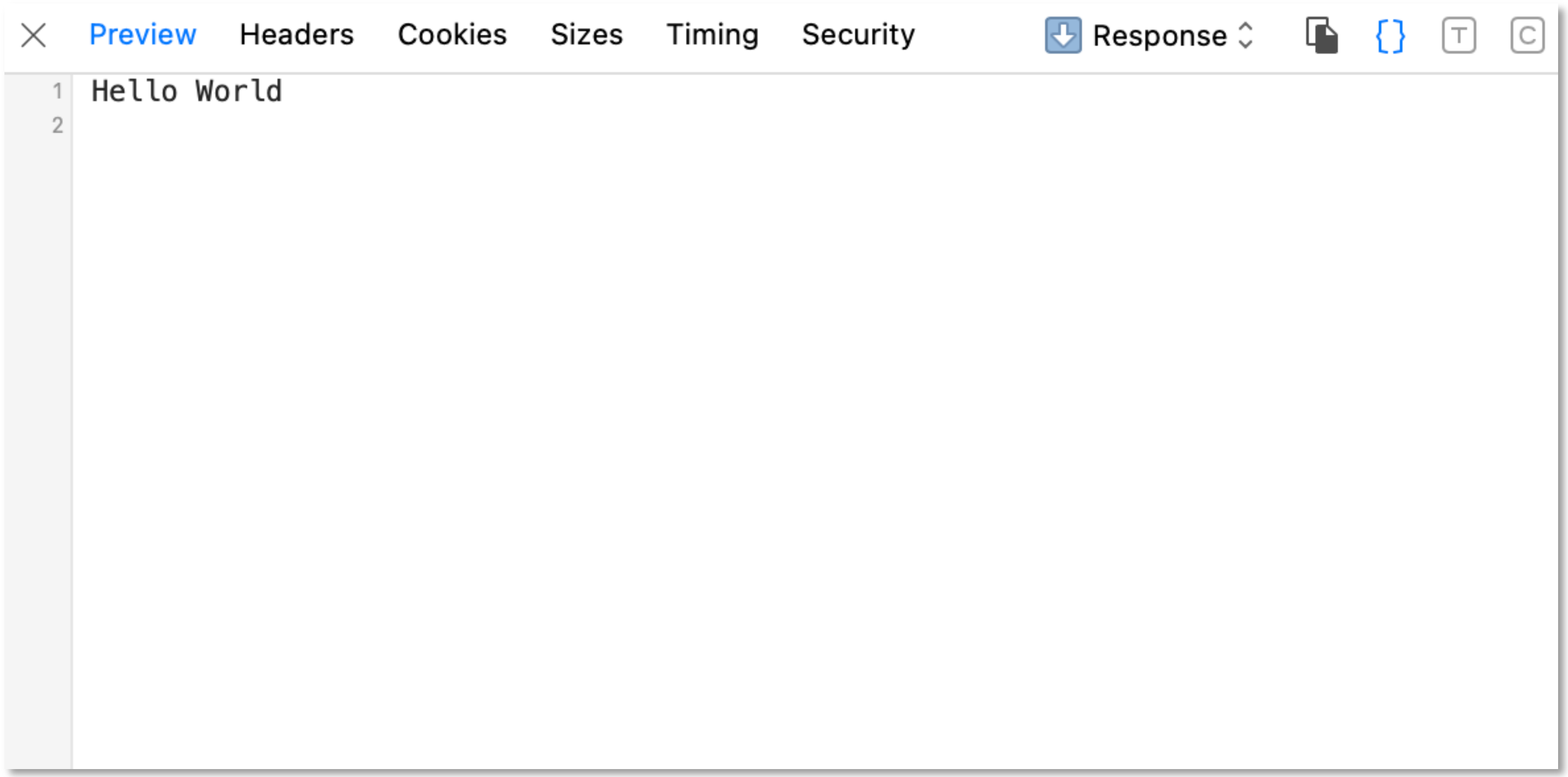
```
server {
    listen      80;
    server_name localhost;
    charset     utf-8;
    client_max_body_size 75M;

    location / { try_files $uri @yourapplication; }
    location /static {
        alias /var/www/static;
    }
    location @yourapplication {
        include uwsgi_params;
        uwsgi_read_timeout 60s;
        uwsgi_pass unix:/var/www/uwsgi/uwsgi.sock;
    }
}
```

hello_world.py in Loaded in the Browser

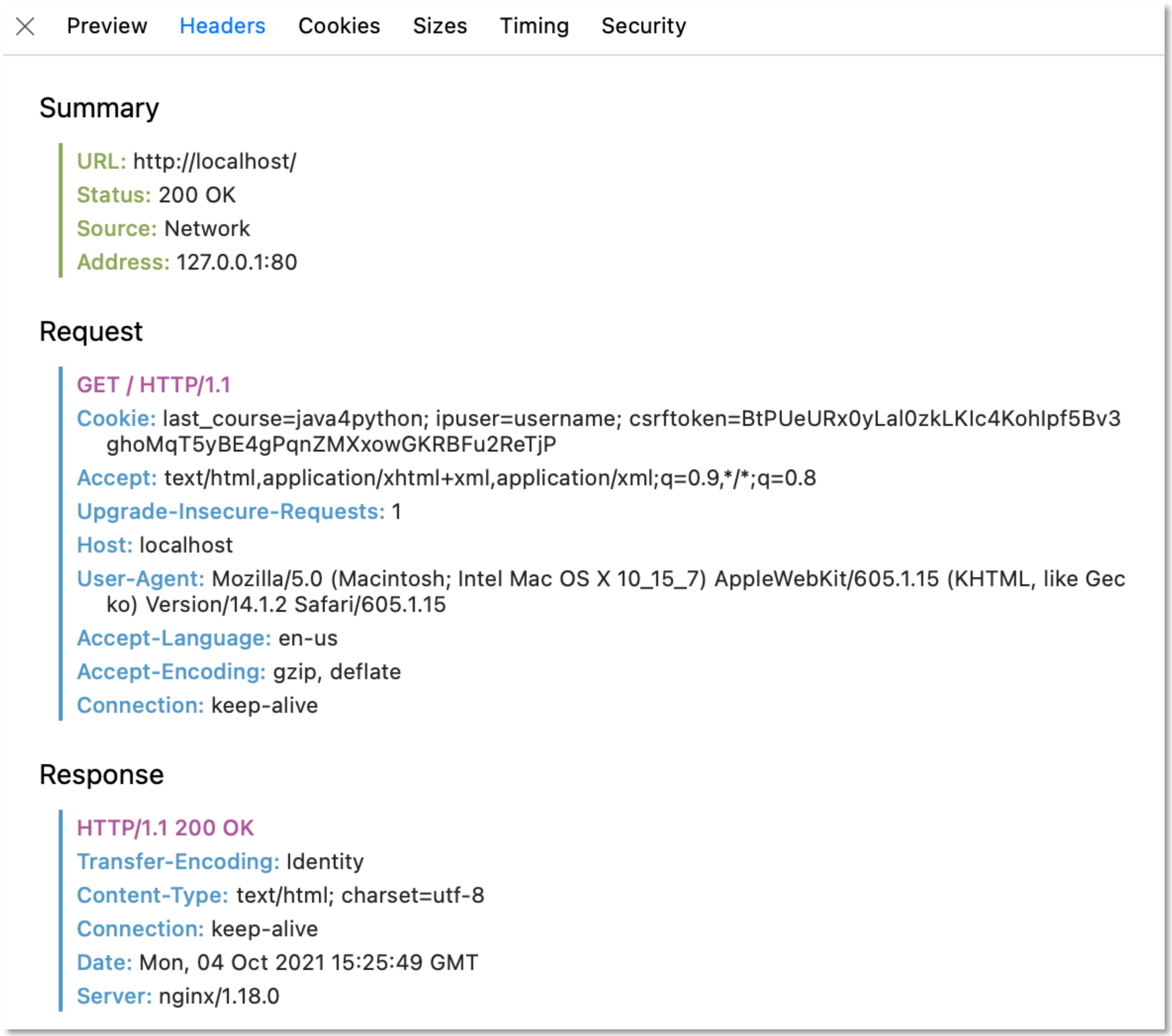


```
Hello World
```

Response in Web Inspector



```
Hello World
```



Preview Headers Cookies Sizes Timing Security

Summary

```
URL: http://localhost/
Status: 200 OK
Source: Network
Address: 127.0.0.1:80
```

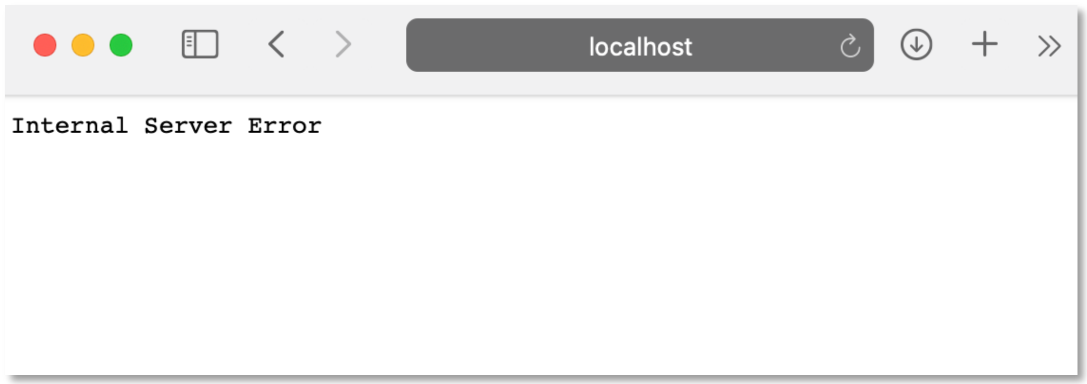
Request

```
GET / HTTP/1.1
Cookie: last_course=java4python; ipuser=username; csrftoken=BtPUeURx0yLal0zkLKIc4Kohlpf5Bv3
    ghoMqT5yBE4gPqnZMXxowGKRBFu2ReTjP
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gec
    ko) Version/14.1.2 Safari/605.1.15
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Transfer-Encoding: Identity
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Date: Mon, 04 Oct 2021 15:25:49 GMT
Server: nginx/1.18.0
```

Internal Server Error



Internal Server Error

Check logs for the source of the error

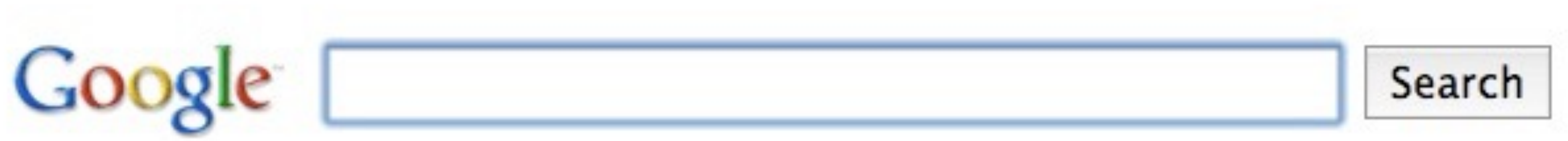
```
 File "/uwsgi/hello_world.py", line 3
    return [b"Hello World]
                         ^
SyntaxError: EOL while scanning string literal
failed to parse file /uwsgi/hello_world.py
unable to load app 0 (mountpoint='') (callable not found or import error)
```

HTML Forms

Sending data to the web application

HTML Forms

- HTML forms provide input to the webserver



- Forms specify:
  - **Where** to send data
  - **What** to send

Google Search Form

Google Search responds to HTTP requests with query data:

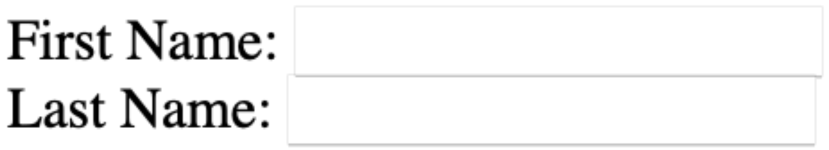
http://www.google.com/search?q=<value>

```
<form action="https://www.google.com/search">
<input name="q">
<input type="submit" value="Search">
</form>
```

HTML Form Fields

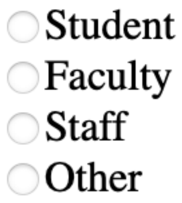
- We can specify each form field’s **name**, **type**, and (default) **value**

```
<form>
First Name: <input name="firstname"><br>
Last Name: <input name="lastname" type="text" value="">
</form>
```


First Name:
Last Name:

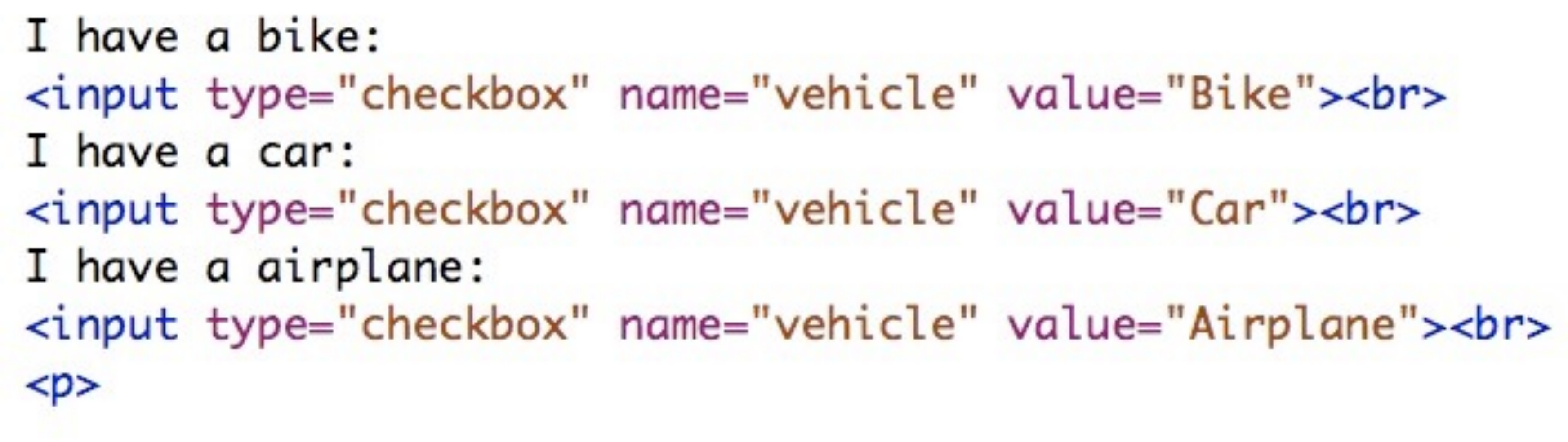
type=radio

```
<form>
<input type="radio" name="role" value="student">Student<br>
<input type="radio" name="role" value="faculty">Faculty<br>
<input type="radio" name="role" value="staff">Staff<br>
<input type="radio" name="role" value="other">Other<br>
</form>
```

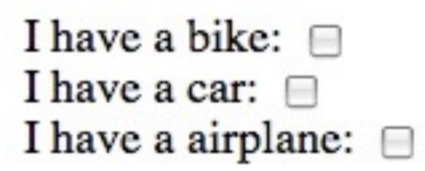


◯ Student
◯ Faculty
◯ Staff
◯ Other

type=checkbox



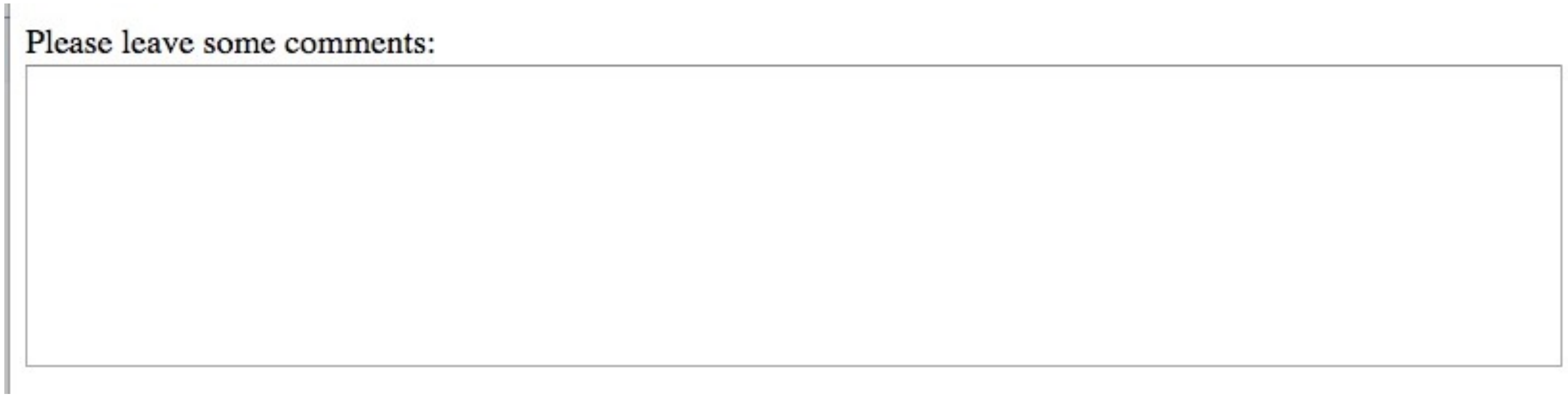
```
I have a bike:
<input type="checkbox" name="vehicle" value="Bike"><br>
I have a car:
<input type="checkbox" name="vehicle" value="Car"><br>
I have a airplane:
<input type="checkbox" name="vehicle" value="Airplane"><br>
<p>
```



I have a bike: ☐
I have a car: ☐
I have a airplane: ☐

`<textarea>`

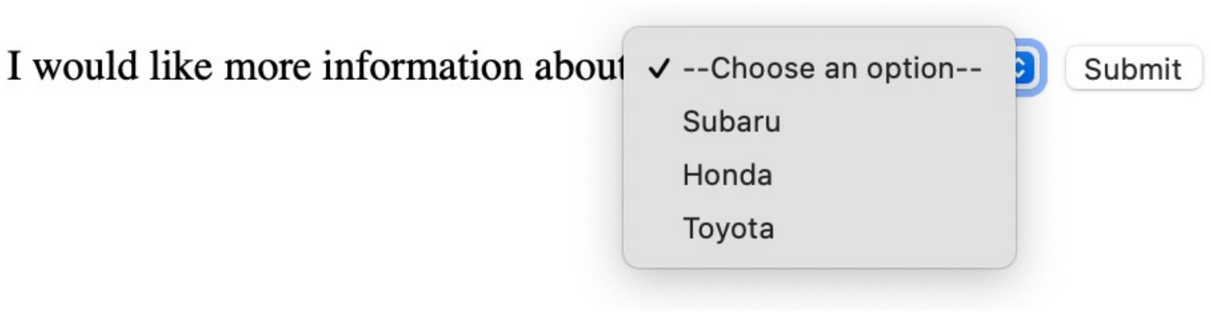
```
Please leave some comments:<br>
<textarea rows=10 cols=100>
</textarea>
```



Please leave some comments:

<select>

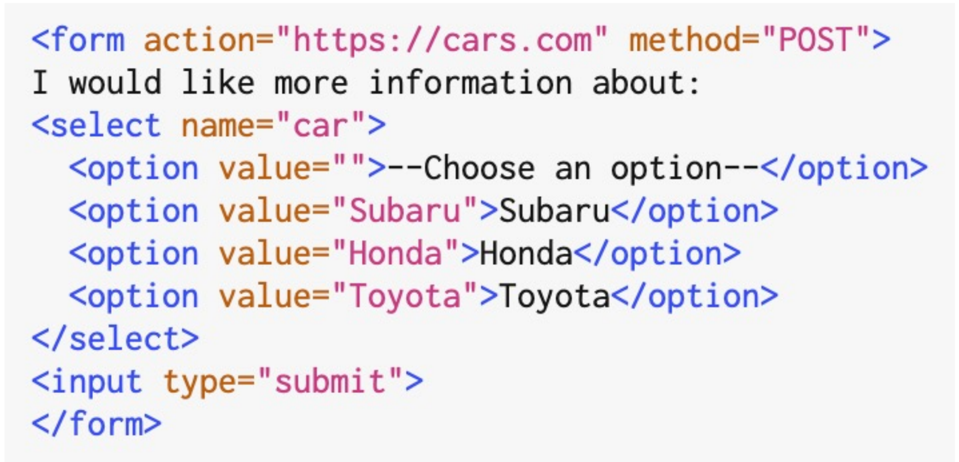
```
<form action="https://cars.com">
I would like more information about:
<select name="car">
  <option value="">--Choose an option--</option>
  <option value="Subaru">Subaru</option>
  <option value="Honda">Honda</option>
  <option value="Toyota">Toyota</option>
</select>
<input type="submit">
</form>
```


I would like more information about

GET vs POST

- GET encodes form contents in the URL:
  - https://cars.com/?car=Subaru
- POST encodes form contents into the body of the HTTP request, so the form content won't be shown in the URL:
  - https://cars.com/

- Security implications:
  - GET should never be used for requests that cause server-side modifications (e.g., updating the database)
  - GET should never be used when a form contains sensitive information



```html
<form action="https://cars.com" method="POST">
I would like more information about:
<select name="car">
  <option value="">--Choose an option--</option>
  <option value="Subaru">Subaru</option>
  <option value="Honda">Honda</option>
  <option value="Toyota">Toyota</option>
</select>
<input type="submit">
</form>
```

Server-side Processing of Form Data

```python
def get_qs_post(env):
    """
    :param env: WSGI environment
    :returns: A tuple (qs, post), containing the query string and post data,
              respectively
    """
    # the environment variable CONTENT_LENGTH may be empty or missing
    try:
        request_body_size = int(env.get("CONTENT_LENGTH", 0))
    except (ValueError):
        request_body_size = 0
    # When the method is POST the variable will be sent
    # in the HTTP request body which is passed by the WSGI server
    # in the file like wsgi.input environment variable.
    request_body = env["wsgi.input"].read(request_body_size).decode("utf-8")
    post = parse_qs(request_body)
    return parse_qs(env["QUERY_STRING"]), post


def application(env, start_response):
    qs, post = get_qs_post(env)
```

Future Topics

- Session state using cookies
- Automated testing
- Web development frameworks
- Common attacks and mitigations:
  - Cross Site Request Forgery (CSRF)
  - Cross Site Scripting (XSS)
  - Clickjacking
  - …

Django

- Python-based framework for web development
- Pros:
  - Less code to write
  - More maintainable
  - More secure
- Cons:
  - More to learn

miniFacebook Demo

Explore these files in the csci220-uwsgi repo:

docker-compose.yml

Dockerfile

miniFacebook.sql

miniFacebook.py

Resources

- Sending Form Data
- Input Element documentation has good examples of input types
- csci220-uwsgi GitHub repository
- WSGI Tutorial

Project Advice

- Schedule multiple weekly meetings
- Establish a shared system for communication (e.g., Discord, GitHub, etc.)
- Assign tasks