

Server-Side Web Programming: Python (Part 2)

Copyright © 2026 by
Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
 - HTTP query strings
 - The HTTP POST method
 - GET vs. POST
 - A fundamental example

Agenda

- **HTTP query strings**
- HTTP POST method
- GET vs. POST
- Fundamental example

HTTP Query Strings

- **So far:**
 - Client/browser sends request to HTTP WSGI server
 - URL specifies protocol, host, port, route
 - HTTP WSGI server passes route to Python app
- **Problem: Often we want:**
 - Client/browser sends request to HTTP WSGI server
 - URL specifies protocol, host, port, route, **arguments**
 - HTTP WSGI server passes route **and arguments** to Python app

HTTP Query Strings

- **Solution:**
 - *Query strings*
 - `key=value` pairs appended to URL

HTTP Query Strings

- **Question:** How does browser pass arguments to HTTP server?
- **Answer 1:** User enters URL at top of browser with a query string appended

HTTP Query Strings

- Answer 1 elaboration

```
protocol://host:port/someroute?key1=value1&key2=value2
```

HTTP Query Strings

- **Question:** How does browser pass arguments to HTTP server?
- **Answer 1:** User enters URL at top of browser with a **query string** appended
- **Answer 2:** User clicks on page link whose `href` attribute has a **query string** appended

HTTP Query Strings

- Answer 2 elaboration

```
<a href="protocol://host:port/someroute?  
    key1=value1&key2=value2">  
    ...  
</a>
```

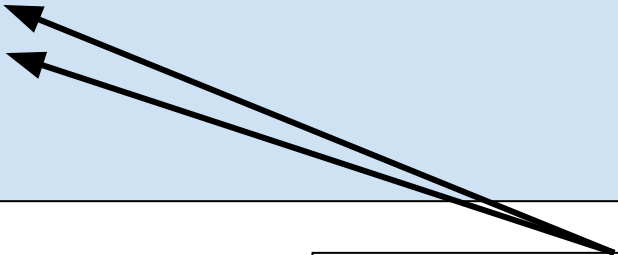
HTTP Query Strings

- **Question:** How does browser pass arguments to HTTP server?
- **Answer 1:** User enters URL at top of browser with a query string appended
- **Answer 2:** User clicks on page link whose `href` attribute has a **query string** appended
- **Answer 3:** User submits form that has `input` elements of type `text`

HTTP Query Strings

- Answer 3 elaboration

```
<form action="protocol://host:port/somerooute" method="get">  
  <input type="text" name="key1">  
  <input type="text" name="key2">  
  <input type="submit">  
</form>
```

Two black arrows originate from a text box on the right and point to the two red-colored input elements in the HTML code block.

User enters *value1* and *value2* in input elements

Sends HTTP request as if generated by clicking on page link with href
protocol://host:port/somerooute?key1=value1&key2=value2

HTTP Query Strings

- Answer 3 elaboration (cont.)

```
<form action="protocol://host:port/someroute" method="get">  
  <input type="text" name="key1" value="initvalue1">  
  <input type="text" name="key2" value="initvalue2">  
  <input type="submit" value="somelabel">  
</form>
```

Aside: URL Encoding

- **Problem:**
 - Sometimes we want query string to contain spaces or special characters
 - URLs cannot contain spaces or special characters (" , ' , = , & , etc.)
- **Solution:**
 - *URL encoding*

Aside: URL Encoding

- *URL encoding*

- A way of encoding a string such that it may appear within a URL
- Each space is encoded as +
- Each special character is encoded as:
 - %nn where nn are hex digits as per UTF-8, or
 - %nn%nn, or
 - %nn%nn%nn, or
 - %nn%nn%nn%nn

Aside: URL Encoding

- Python support for URL encoding
 - `urllib.parse.quote_plus()`
 - string → URL encoded string
 - `urllib.parse.unquote_plus()`
 - URL encoded string → string

Aside: URL Encoding

```
$ python
Python 3.14.6 (main, Jun 11 2026,
12:32:48) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import urllib.parse
>>> urllib.parse.quote_plus('one')
'one'
>>> urllib.parse.quote_plus('t wo')
't+wo'
>>> urllib.parse.quote_plus('th"r ee')
'th%22r+ee'
>>> quit()
$
```

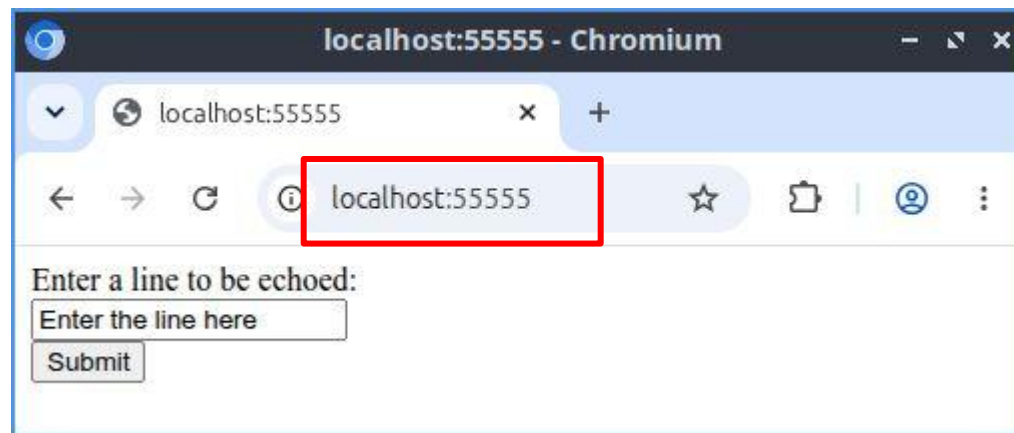
HTTP Query Strings

- See **EchoWsgiGet** app

```
$ python runserver.py 55555  
Listening on port 55555
```

HTTP Query Strings

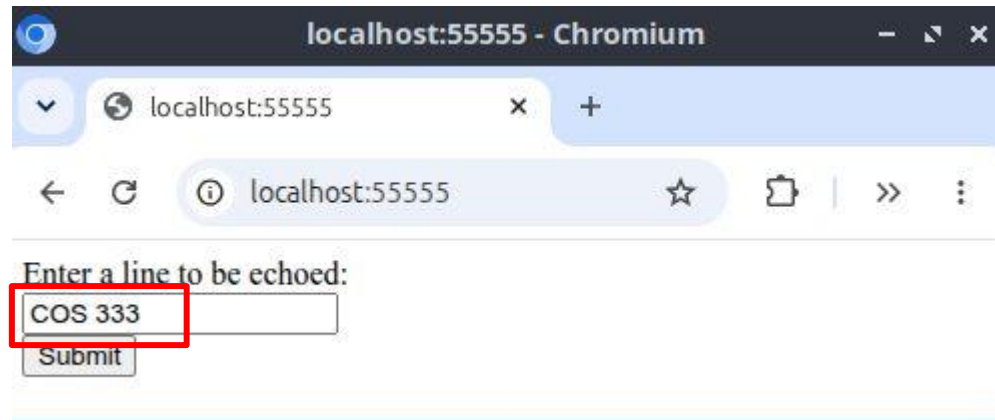
- See EchoWsgiGet app (cont.)



index page

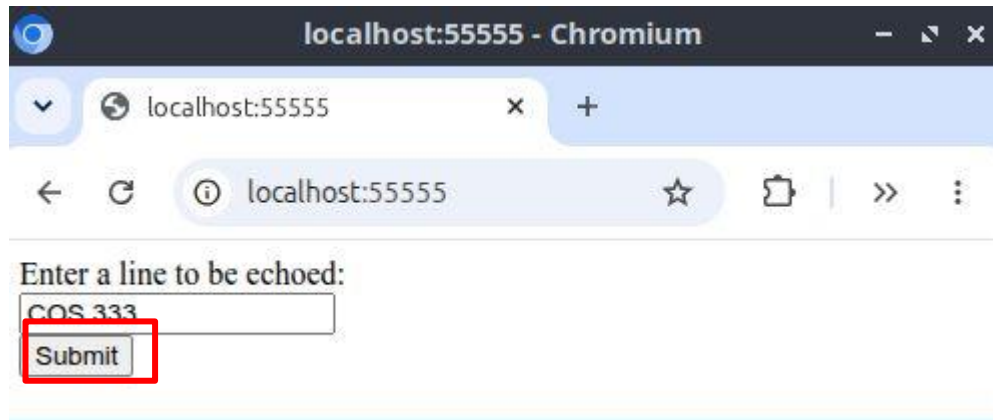
HTTP Query Strings

- See EchoWsgiGet app (cont.)



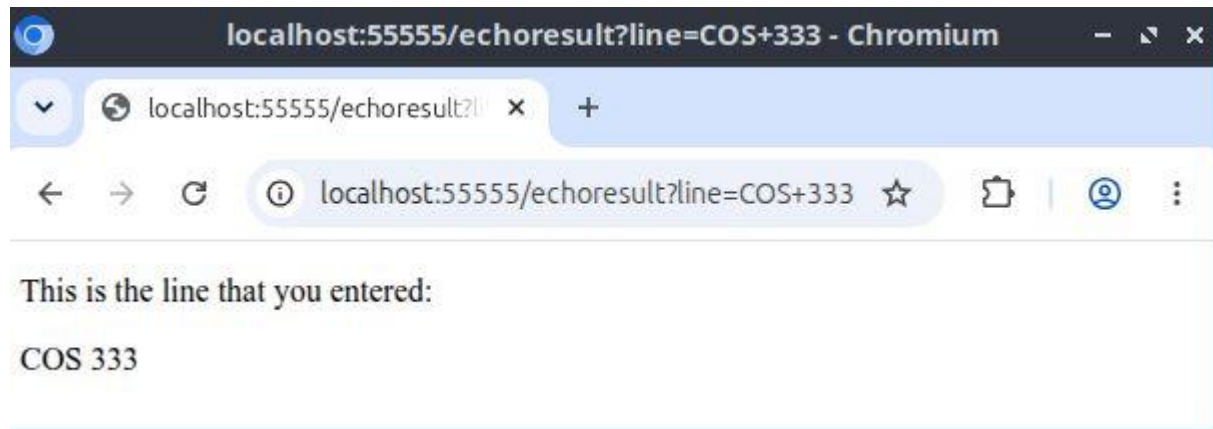
HTTP Query Strings

- See **EchoWsgiGet** app (cont.)



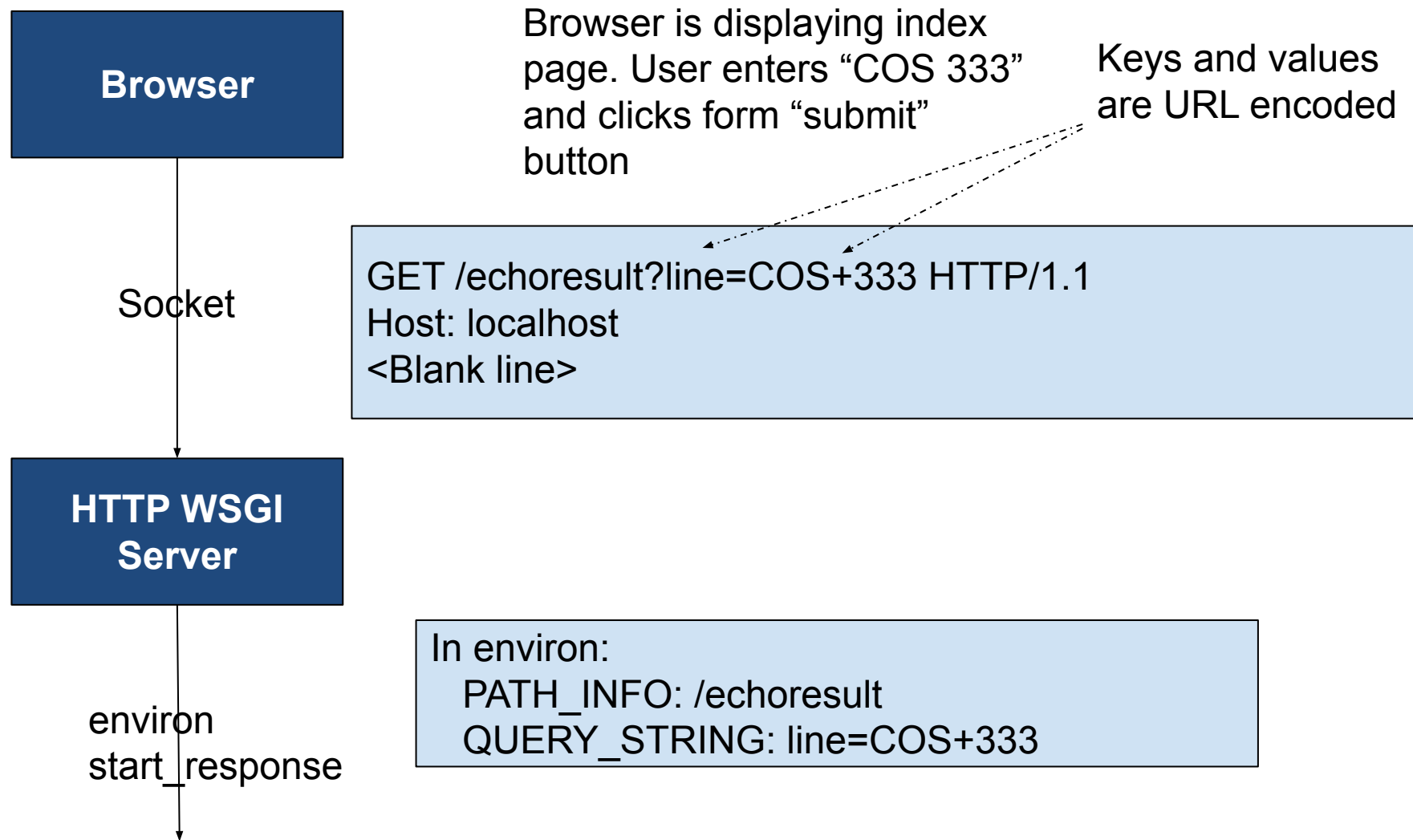
HTTP Query Strings

- See **EchoWsgiGet** app (cont.)

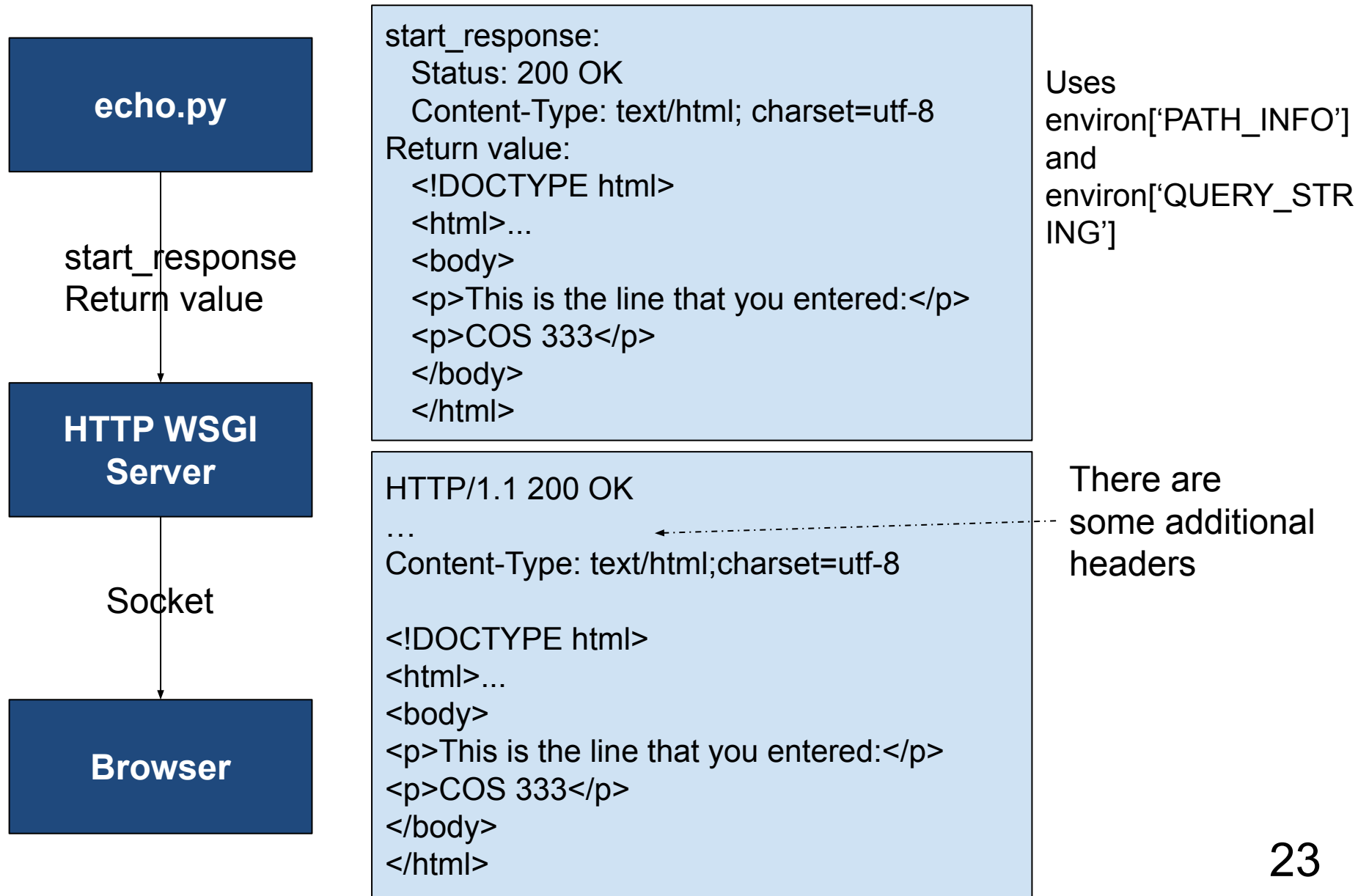


echoresult page

HTTP Query Strings



HTTP Query Strings



HTTP Query Strings

- See **EchoWsgiGet** app
 - **runserver.py**
 - **echo.py**

Agenda

- HTTP query strings
- **HTTP POST method**
- GET vs. POST
- Fundamental example

HTTP POST Method

- **So far:** Browser communicates with HTTP server using the **GET** method
- **Question:** How does user command browser to communicate with HTTP server using the **POST** method?
- **Answer:** User submits form

HTTP POST Method

- Answer elaboration

```
<form action="protocol://host:port/somerooute" method="post">  
  <input type="text" name="key1" value="initvalue1">  
  <input type="text" name="key2" value="initvalue2">  
  <input type="submit" value="somelabel">  
</form>
```

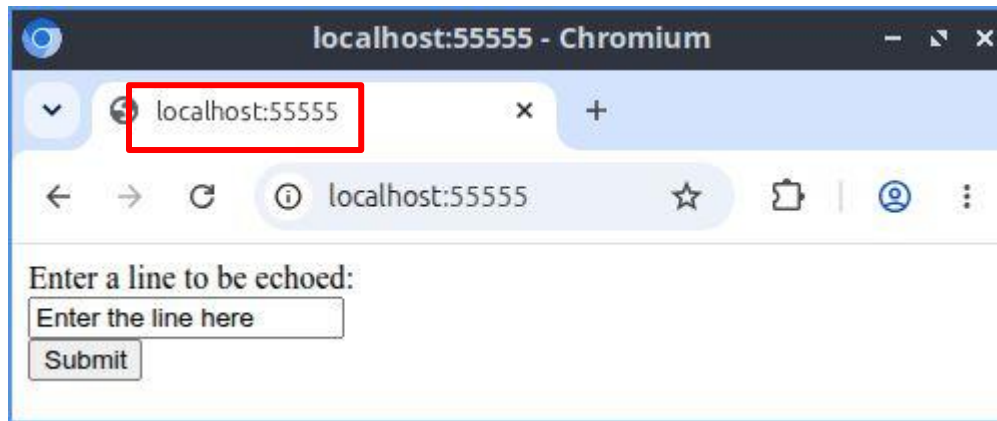
HTTP POST Method

- See **EchoWsgiPost** app

```
$ python runserver.py 55555  
Listening on port 55555
```

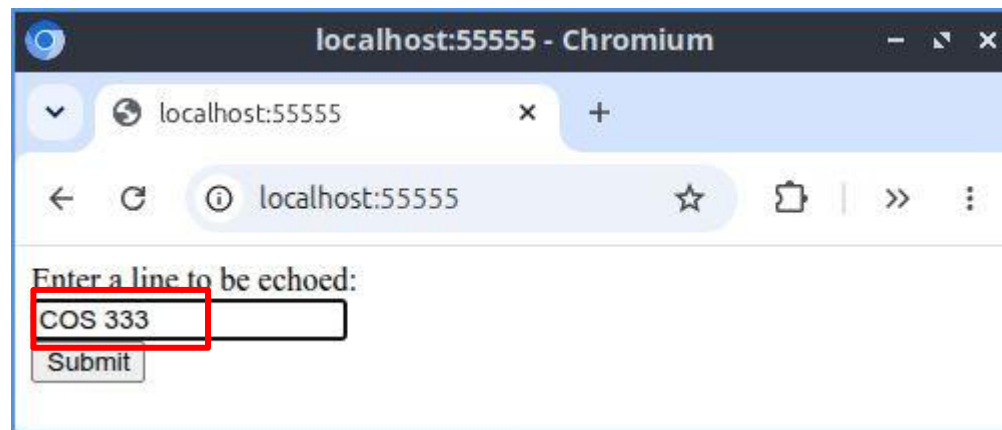
HTTP POST Method

- See EchoWsgiPost app (cont.)



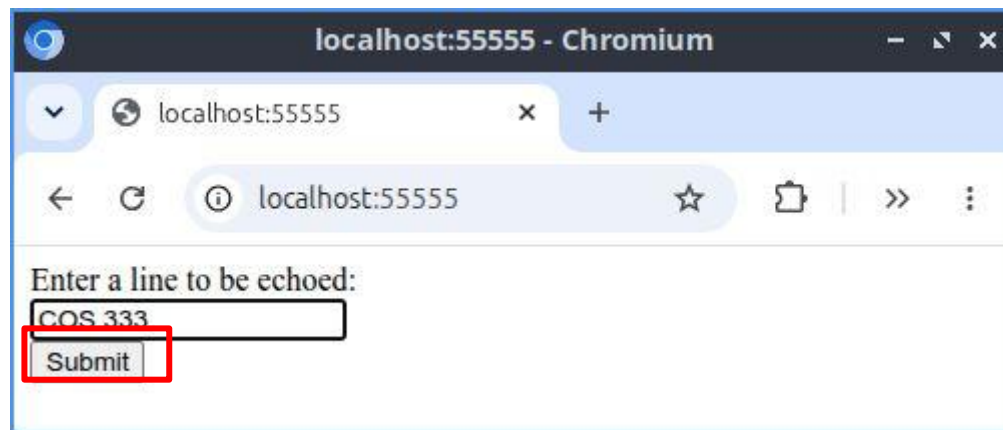
HTTP POST Method

- See EchoWsgiPost app (cont.)



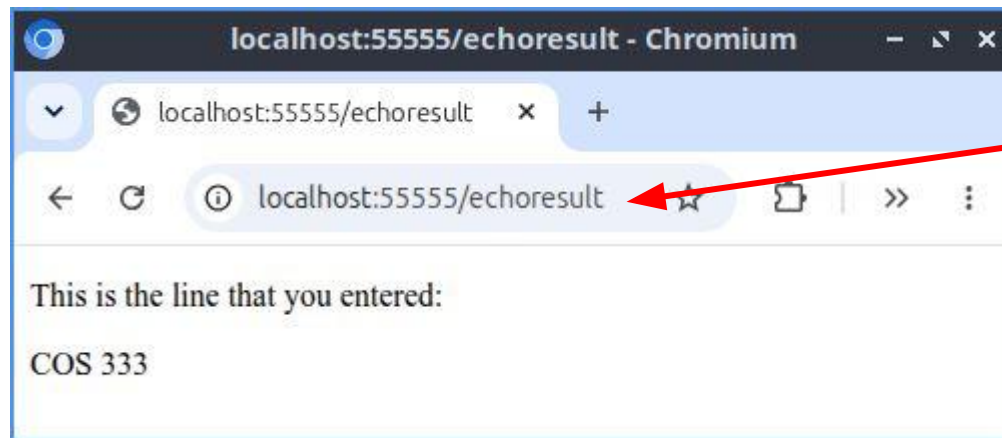
HTTP POST Method

- See EchoWsgiPost app (cont.)



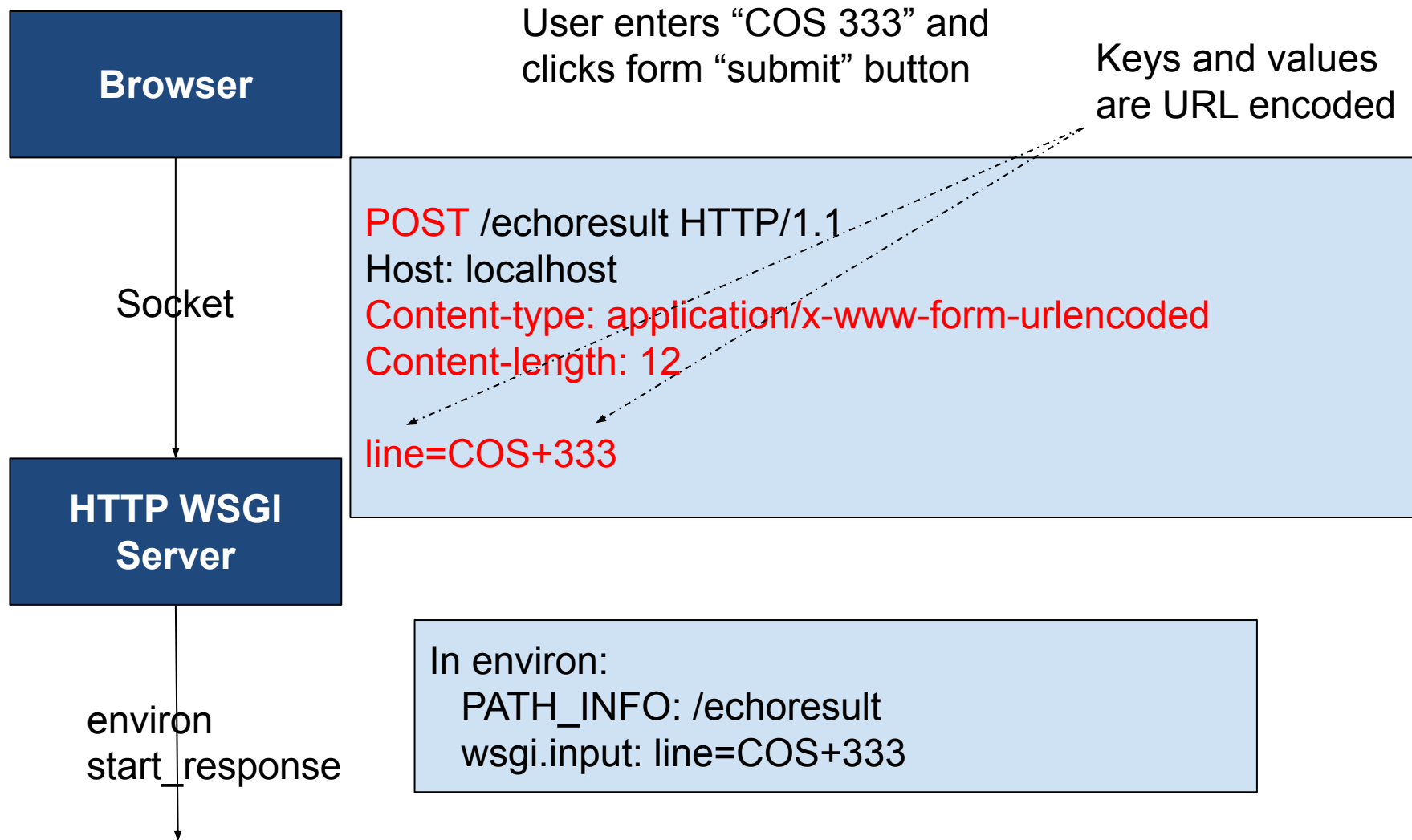
HTTP POST Method

- See **EchoWsgiPost** app (cont.)

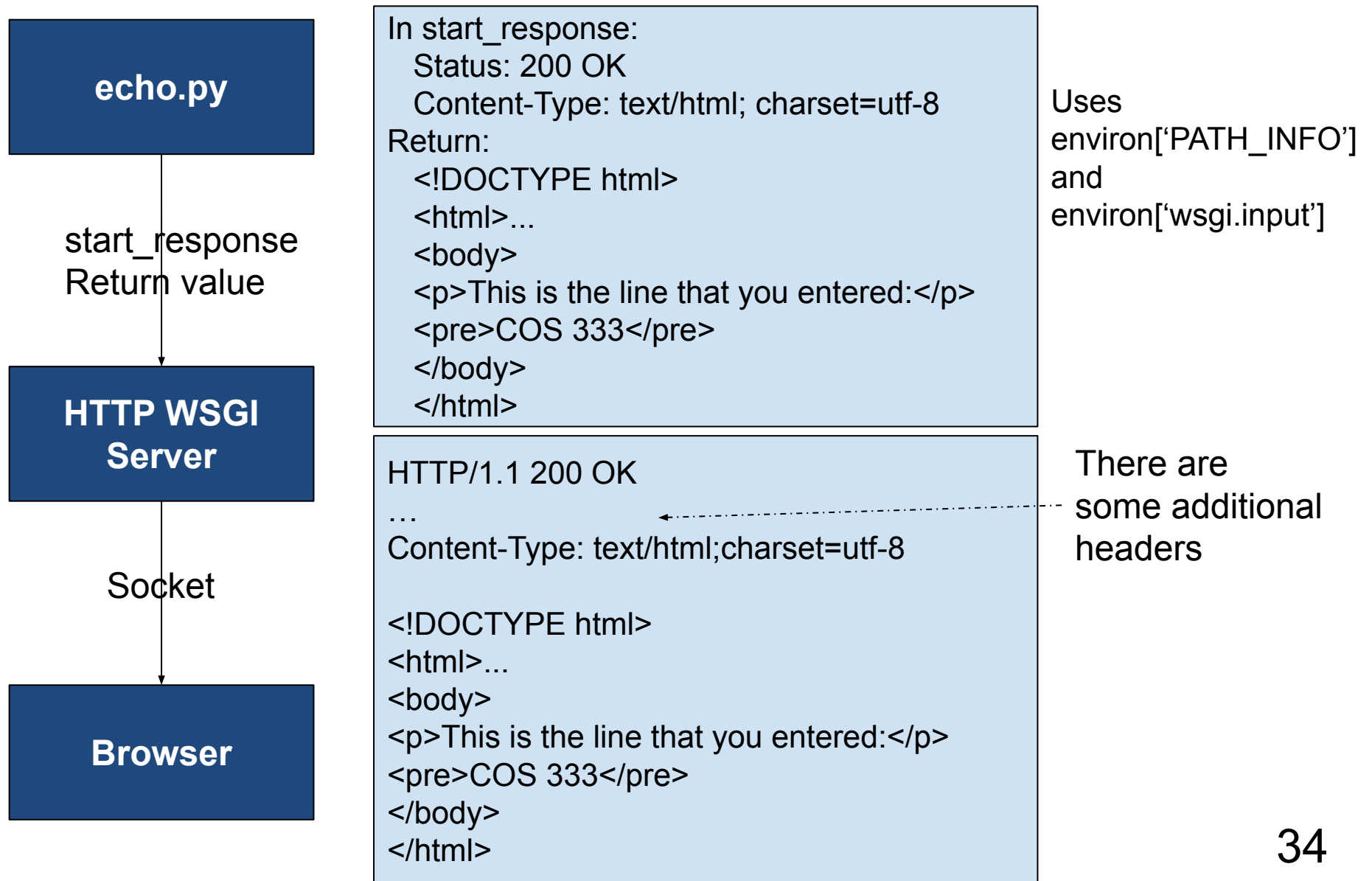


Note small
difference here

HTTP POST Method



HTTP POST Method



HTTP POST Method

- See **EchoWsgiPost** app
 - runserver.py
 - **echo.py**

Agenda

- HTTP query strings
- HTTP POST method
- **GET vs. POST**
- Fundamental example

GET vs. POST

- When to use **GET**?
- When to use **POST**?

GET vs. POST

- Technical considerations
 - Some HTTP servers reject long headers
 - Use **POST** when:
 - The query string is long, that is...
 - There are very many `key=value` pairs
 - Some keys or values are long

GET vs. POST

- Security considerations
 - Some HTTP servers store request URLs in log files
 - Use **POST** when:
 - Some `key=value` pairs in the query string must be kept secure
 - E.g., passwords

GET vs. POST

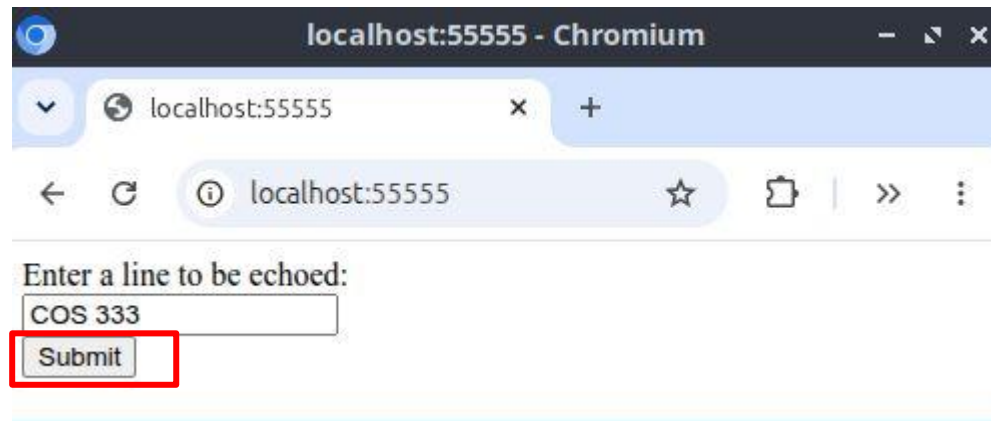
- Convention (informally):
 - Use **GET** if the app's processing **will not** change persistent data
 - Use **POST** if the app's processing **will** change persistent data

GET vs. POST

- Browser has sent **GET** request =>
 - Browser assumes request **will not** change persistent data
 - Browser **does not warn** about page reload
- Browser has sent **POST** request =>
 - Browser assumes request **will** change persistent data
 - Browser **warns** about page reload

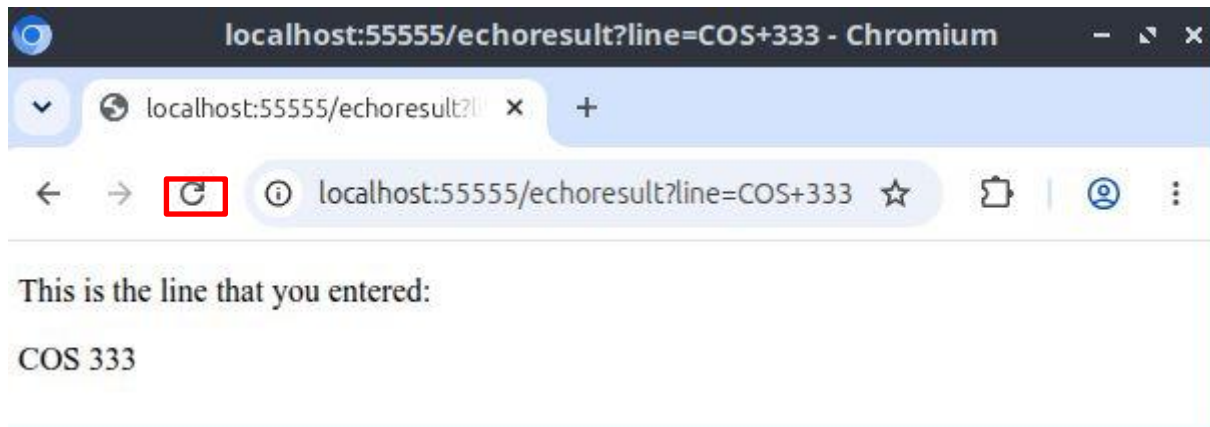
GET vs. POST

- Recall **EchoWsgiGet** app



GET vs. POST

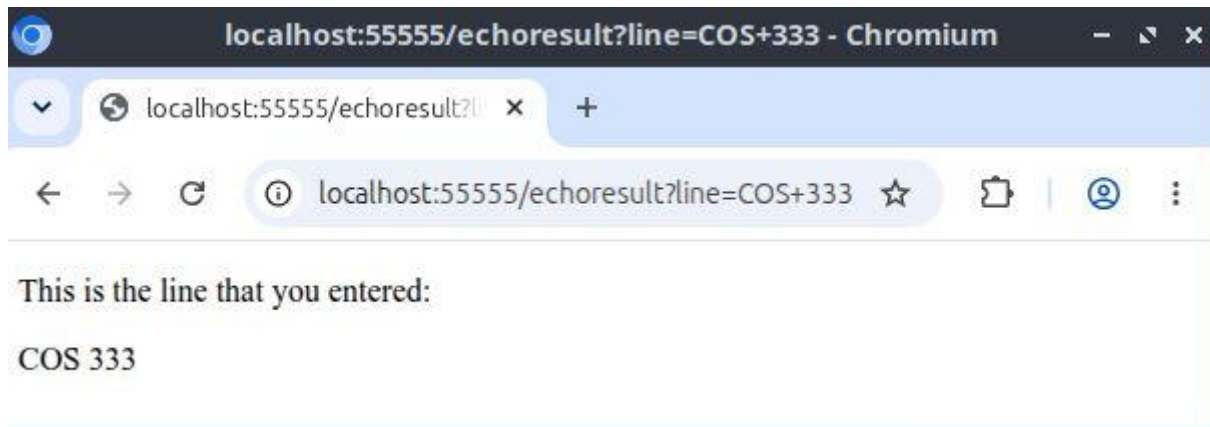
- Recall **EchoWsgiGet** app (cont.)



Reload this page

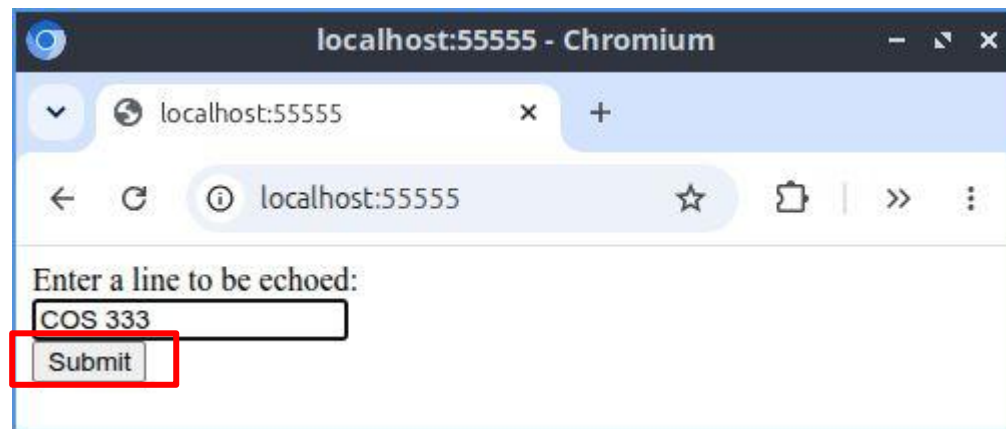
GET vs. POST

- Recall **EchoWsgiGet** app (cont.)



GET vs. POST

- Recall **EchoWsgiPost** app (revisited)



GET vs. POST

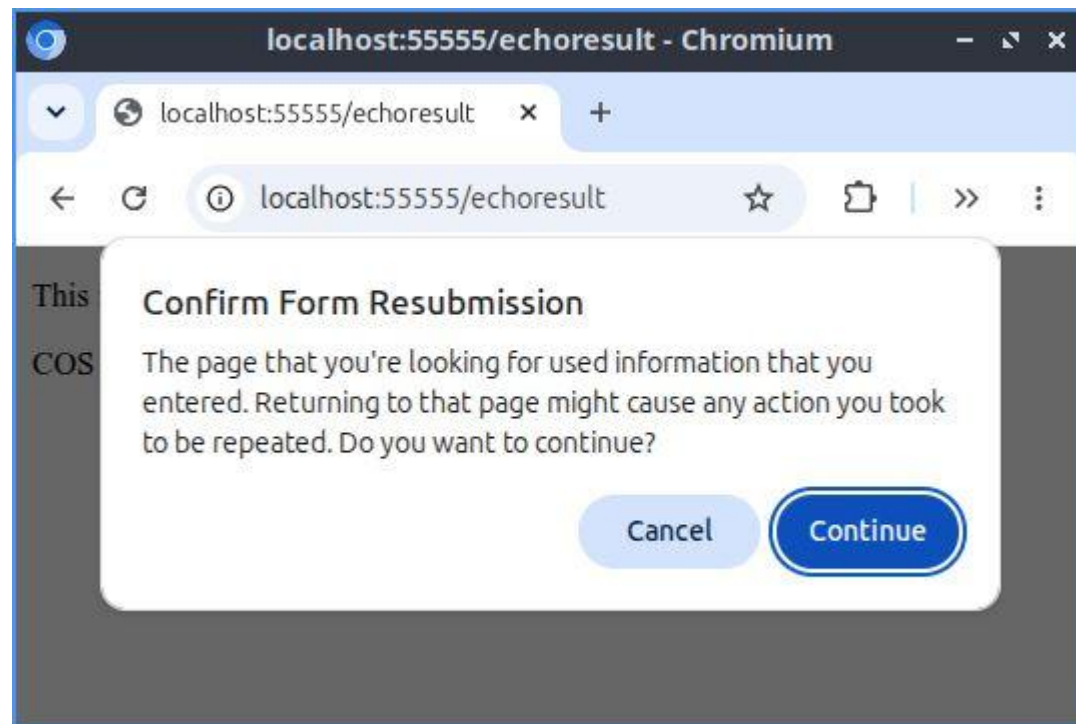
- Recall **EchoWsgiPost** app (cont.)



Reload this page

GET vs. POST

- Recall **EchoWsgiPost** app (cont.)



Question (09webpython2a)

- COS 333 Assignment 3 will be a web version of Assignment 2. The browser will send “overviews” and “details” requests to a HTTP WSGI server. Which HTTP method should the browser use for “overviews” requests? Which HTTP method should the browser use for “details” requests?
 - (a) GET, GET
 - (b) GET, POST
 - (c) POST, GET
 - (d) POST, POST
- Browse to <https://cos333attend.cs.princeton.edu> to answer

[2 points]

Aside: HTTP Methods

- **HTML** uses:
 - GET, POST
- **HTTP** defines:
 - GET, POST
 - PUT, PATCH, DELETE, CONNECT, HEAD, OPTIONS, TRACE
 - Sometimes used with AJAX requests (described later)

Aside: HTTP Methods

Method	Conventional Purpose	Idempotent?
GET	Retrieve resource	Yes
POST	Create resource	No
PUT	Replace resource	Yes
PATCH	Update resource	No
DELETE	Delete resource	Yes

Others: **CONNECT, HEAD, OPTIONS, TRACE**

See Chapter 14 of *Flask Web Development*
by Miguel Grinberg

Agenda

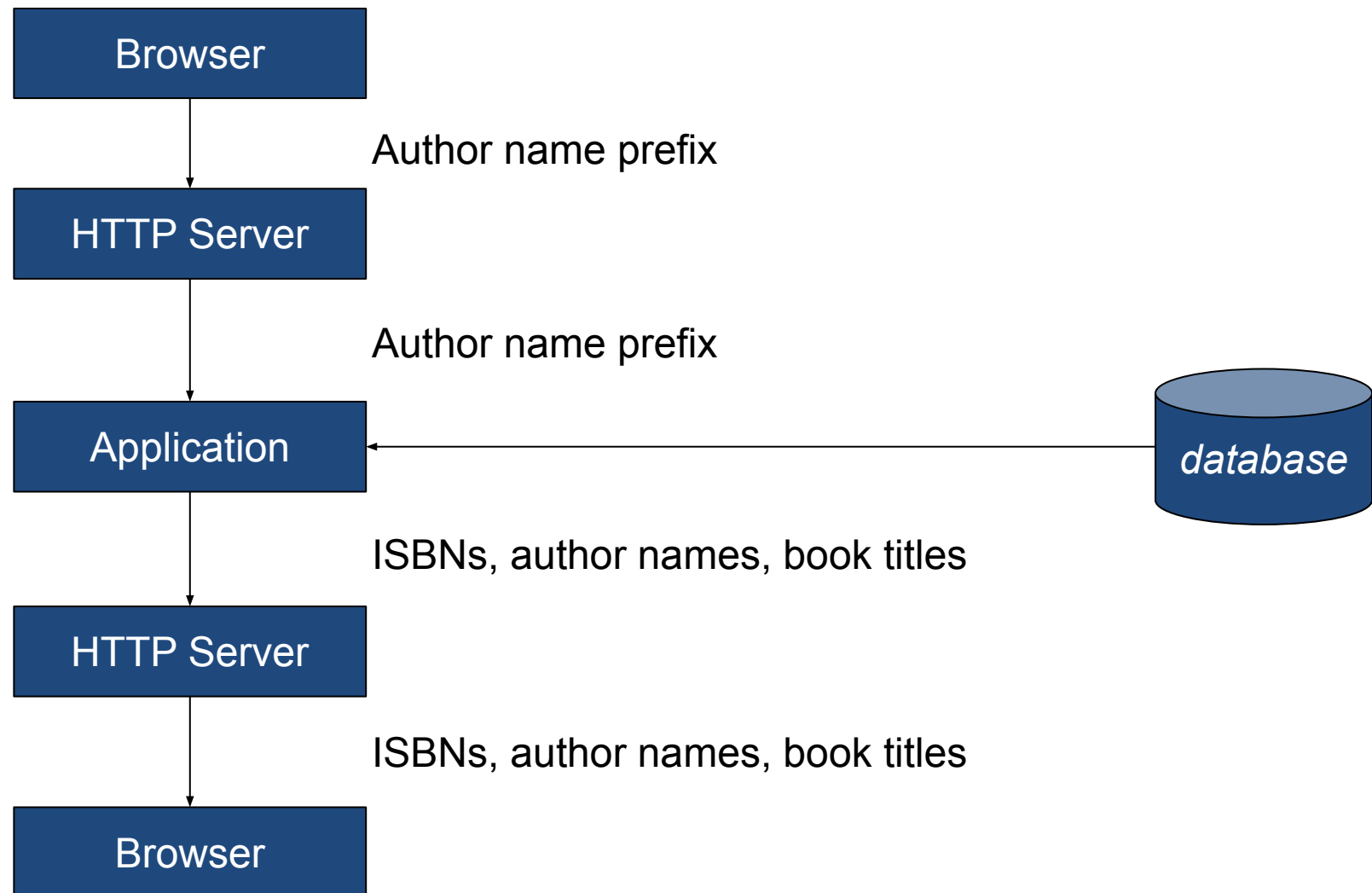
- HTTP query strings
- HTTP POST method
- GET vs. POST
- **Fundamental example**

Fundamental Example

- **Penny app**
 - Website for a very small bookstore
 - We'll see *many* versions

Fundamental Example

Penny app



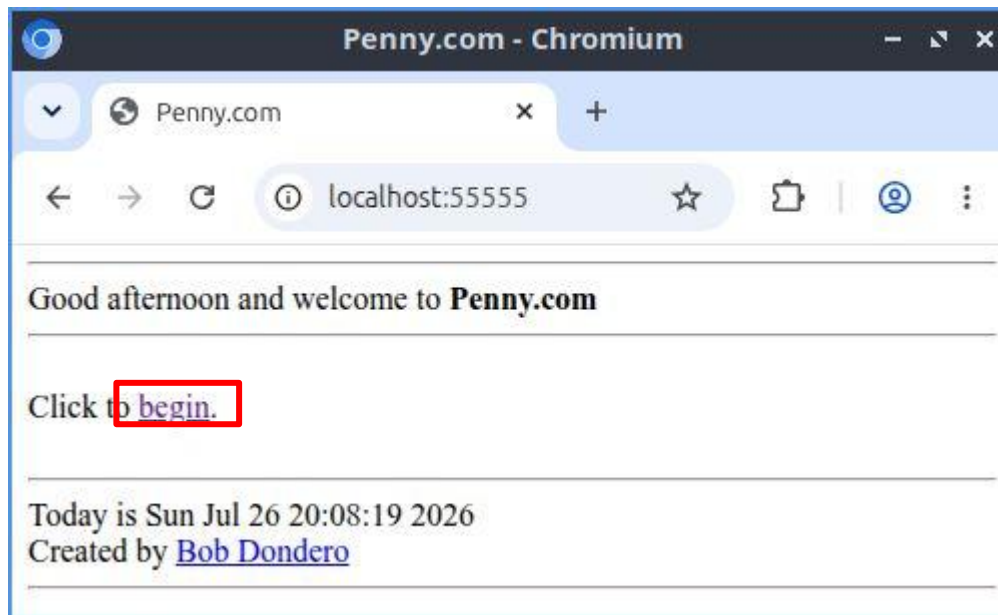
Fundamental Example

- See **PennyWsgi** app

```
$ python runserver.py 55555  
Serving HTTP on 0.0.0.0 port 55555  
(http://0.0.0.0:55555/) ...
```

Fundamental Example

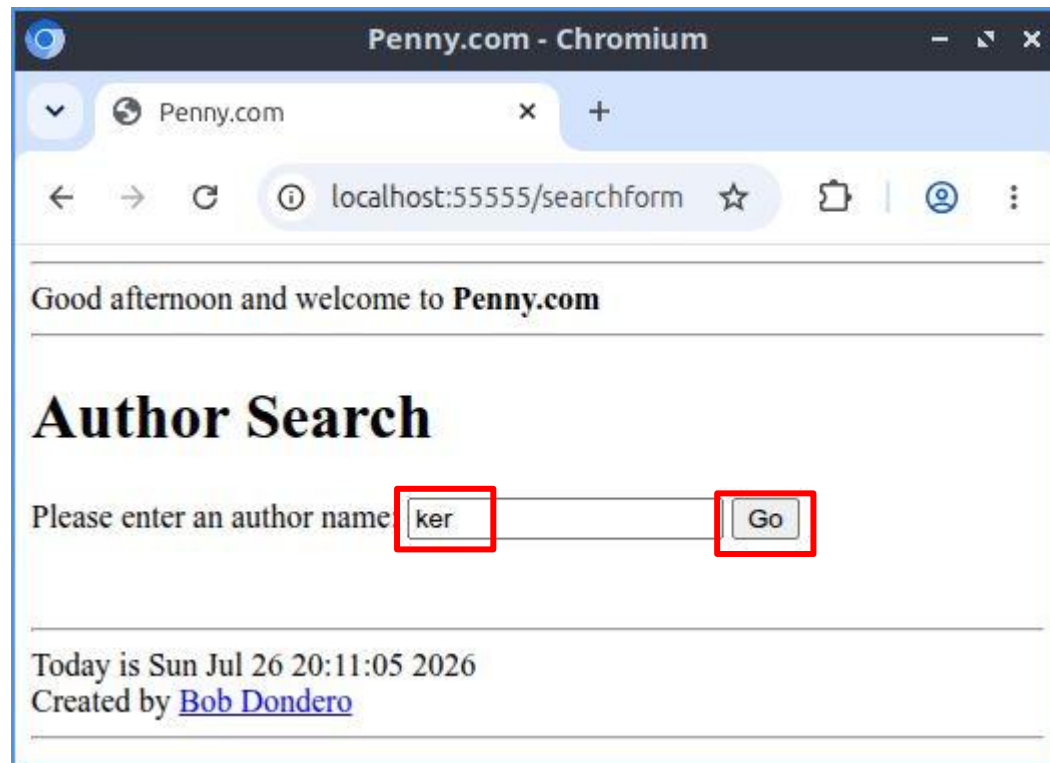
- See **PennyWsgi** app (cont.)



The
index
page

Fundamental Example

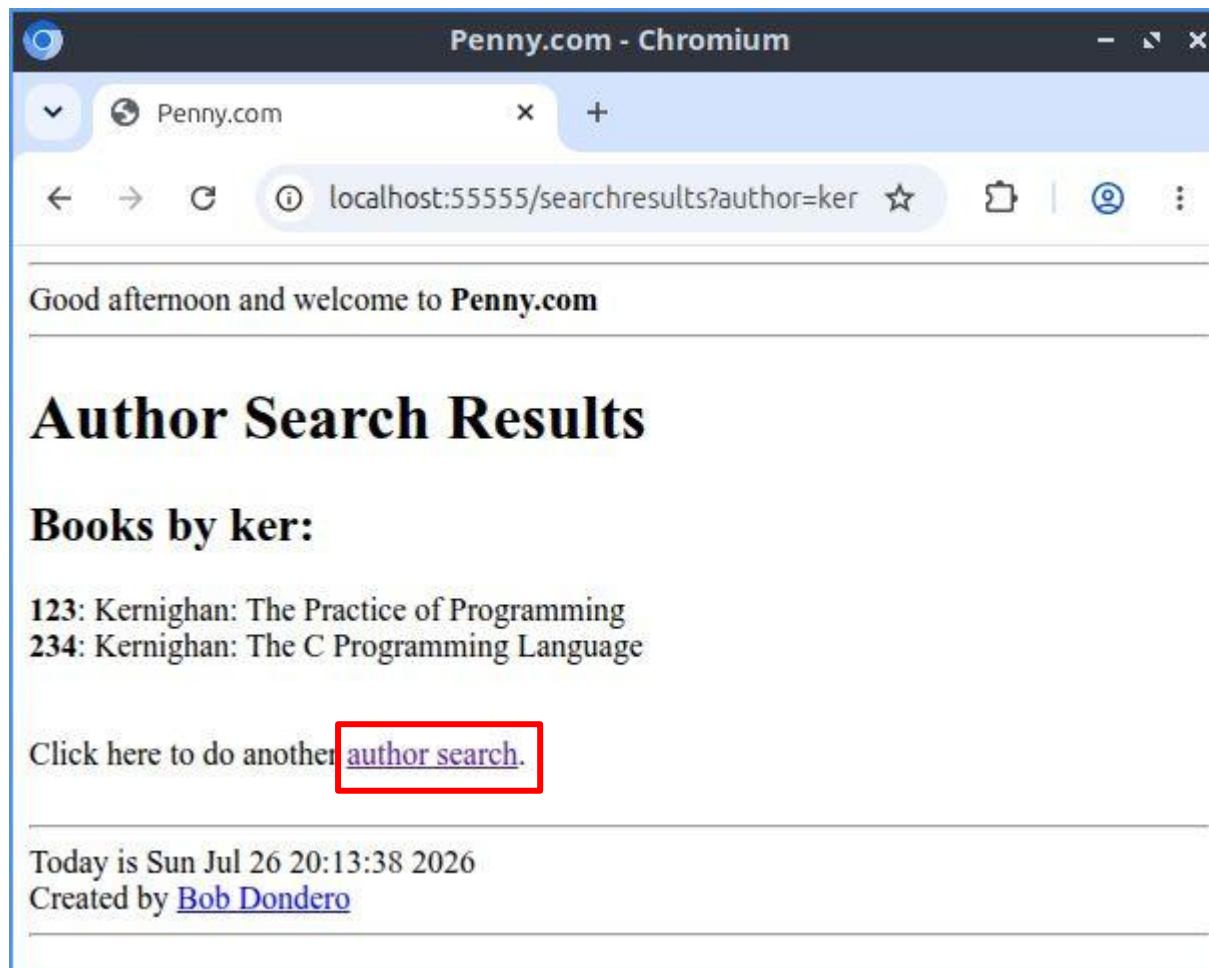
- See PennyWsgi app (cont.)



The
searchform
page

Fundamental Example

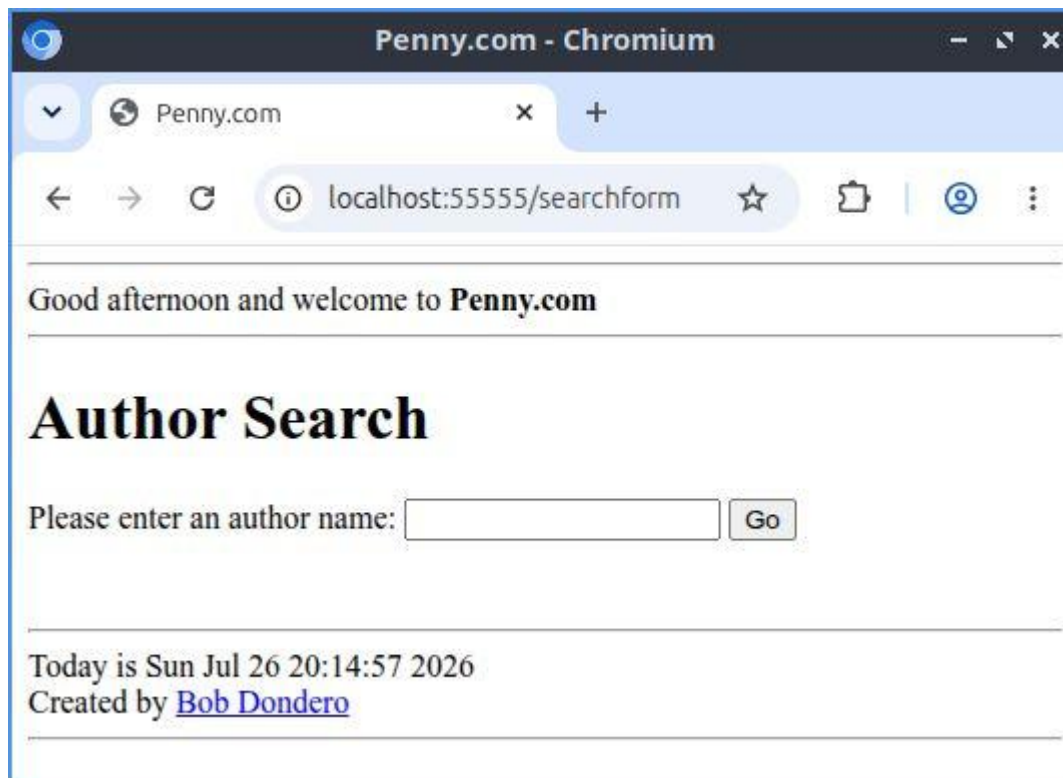
- See PennyWsgi app (cont.)



The
searchresults
page

Fundamental Example

- See **PennyWsgi** app (cont.)



The
searchform
page

Fundamental Example

- See PennyWsgi app (cont.)
 - **runserver.py**
 - **penny.sql**
 - penny.sqlite
 - **database.py**
 - **penny.py**

Summary

- We have covered:
 - HTTP query strings
 - The HTTP POST method
 - GET vs. POST
 - A fundamental example