Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

MCS 507 Lecture 20
Mathematical, Statistical and Scientific Software
Jan Verschelde, 6 October 2023

Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

client/server networks

A *client* is a computer in the network that *requests* for access to data and services from another computer.

A *server* is a computer in the network *receives and processes* requests from clients.
Access permissions are determined by the server.

A *client/server network* consists of several computers connected in a network, acting as clients and/or servers.

Client/server computing emerged in the nineties to distribute applications (such as database administration) over a network.

Hypertext Transfer Protocol (HTTP)

exchange data between the client and the server

HTTP is based on request-response between
a web browser (the client) and a web server.
A typical transaction between browser and server:

1. A TCP/IP connection is established
   between browser and server.
2. The browser sends a request for a web page.
3. The server locates the file and responds,
   sending the content of the requested web page.
4. The TCP/IP connection is closed.

the Web Server Apache

the A in LAMP

The combination of web server, scripting language, and database is often referred to as the LAMP system.

LAMP stands for

- L is Linux, the operating system;
- A is Apache, the web server;
- M is MySQL, the database;
- P is Python, the scripting language.

Observe that all four are open source software.

Apache makes a cute pun on “a patchy web server”,
but its name is in honor of the Native American Apache tribe.
Its web site is at `http://www.apache.org`.

running Apache

on a Mac OS X

Apache is platform independent.
We will demonstrate on a Mac OS X.

1. Apache is already installed on Mac OS X,
   launch Safari with `http://localhost/` to verify.
2. Instead of `public_html`, the `Sites` directory is where Mac users store their web pages.
3. Instead of `/var/www/cgi-bin`, CGI scripts are in `/Library/WebServer/CGI-Executables`.
   CGI = Common Gateway Interface, is set of protocols through which applications interact with web servers.

Using `localhost` we remain working offline.

more instructions are needed

To check the version of Apache:

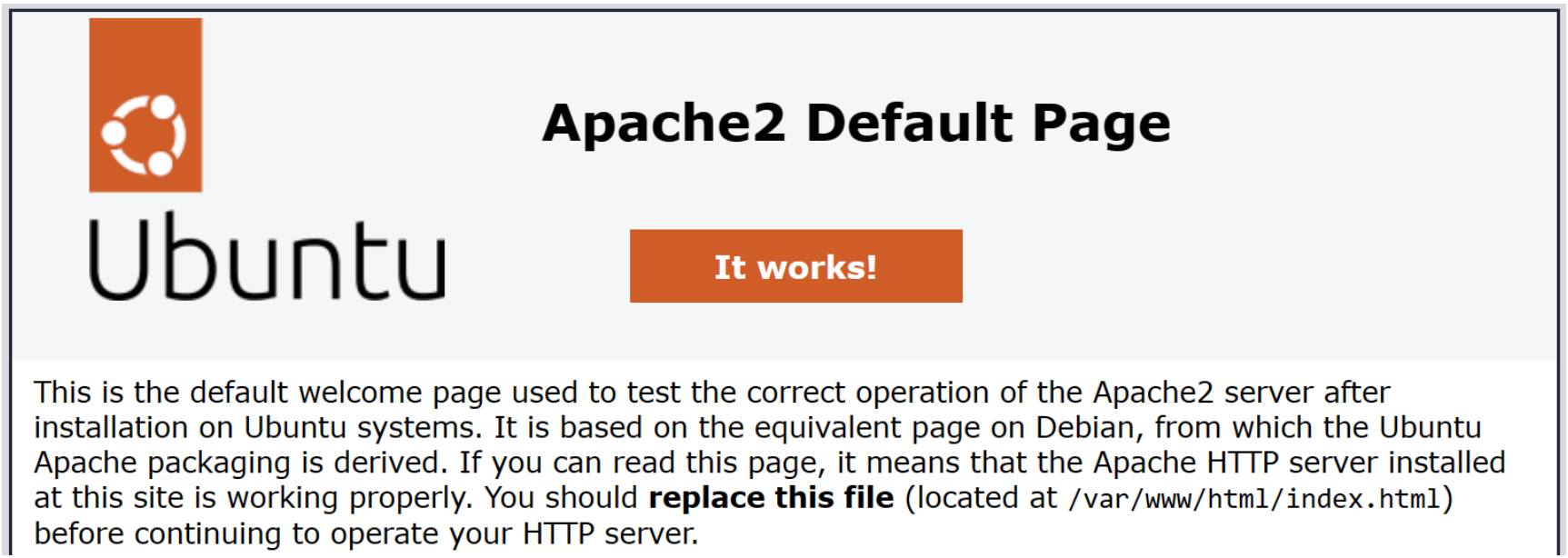
```
$ httpd -v
Server version: Apache/2.4.34 (Unix)
Server built:   Feb 22 2019 20:20:11
$
```

To launch Apache, type `sudo /usr/sbin/apachectl start`
in a Terminal window.
Pointing the browser to `localhost` (or 127.0.0.1) shows `It works!`.

`apachectl`: Apache HTTP Server Control Interface
Other useful `apachectl` arguments are `restart` and `stop`.

on Ubuntu, via Window Subsystem for Linux

(0) sudo apt-get upgrade
(1) sudo apt-get install apache2
(2) sudo service web server apache2
(3) point the browser to localhost:80



serving web pages from user directories

on Mac OS X

To serve web pages from user directories, in the file

```
/etc/apache2/extra/httpd-userdir.conf
```

uncomment the line that contains

```
Include /private/etc/apache2/users/*.conf
```

In `/etc/apache2/httpd.conf` uncomment the lines

```
LoadModule userdir_module libexec/apache2/mod_userdir.so
# User home directories
Include /private/etc/apache2/extra/httpd-userdir.conf
```

and also the line

```
LoadModule cgi_module libexec/apache2/mod_cgi.so
```

configuration of users

on Mac OS X

In the folder `/etc/apache2/users`,
if `<user>` is your user name, in the file `<user>.conf`,
then add the line

```
    Require all granted
```

inside the block

```
<Directory "/Users/<user>/Sites/">
  Options Indexes MultiViews
  AllowOverride None
  Require all granted
</Directory>
```

two more steps

on Mac OS X

1. `sudo apachectl restart`
2. In the `System Preferences`, select `Sharing`, and turn the `Internet Sharing` on.

steps on Ubuntu

to run scripts

1. enable module `cgid` via `sudo a2enmod cgid`
2. `sudo vi /etc/apache2/conf-available/serve-cgi-bin.conf`

   Make the following (or similar) edits:

```
<IfDefine ENABLE_USR_LIB_CGI_BIN>
    ScriptAlias /cgi-bin/ /var/www/cgi-bin/
    <Directory "/var/www/cgi-bin">
        AllowOverride None
        Options +ExecCGI
        AddHandler cgi-script .py
    </Directory>
</IfDefine>
```

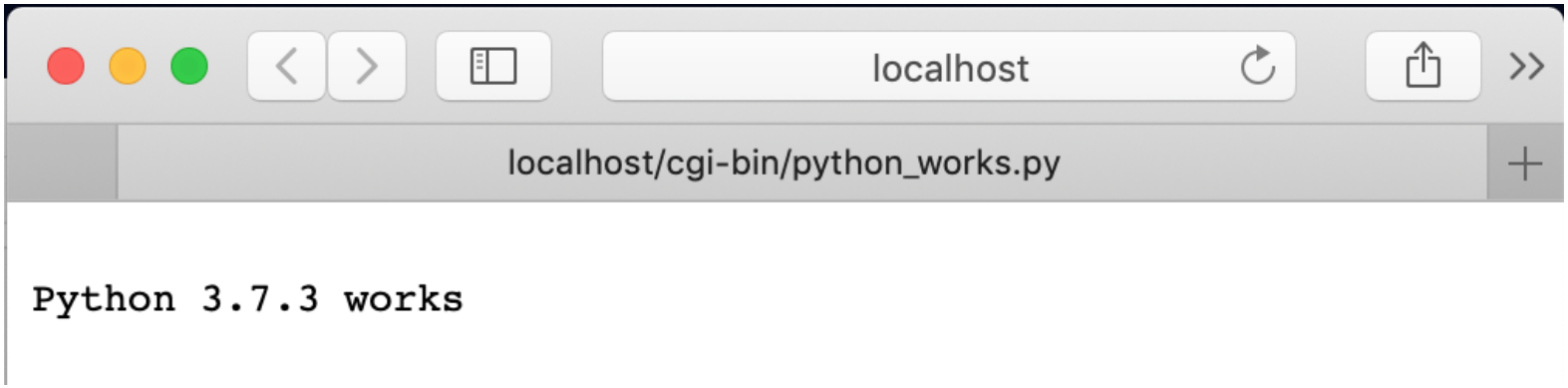
3. restart with `sudo service apache2 restart`

Python works

The script `python_works.py` contains 3 lines:

```
#!< path to python interpreter >
print("Content-Type: text/plain\n\n")
print("Python 3.7.3 works")
```

is made `executable` (after `chmod +x python_works.py`), and is in `/Library/WebServer/CGI-Executables`.



Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

running a simple CGI server

The documentation string of `myserver.py`:

```
"""
Launch this code in the directory where
the CGI script is that you want to run.
Point the browser to localhost:8000
(or whatever the value of PORTNUMBER is),
followed by / and the name of the script.
"""
```

To run the CGI scripts in this lecture,
one does not really need Apache.

The portnumber defined in the script must be open.

the script myserver.py

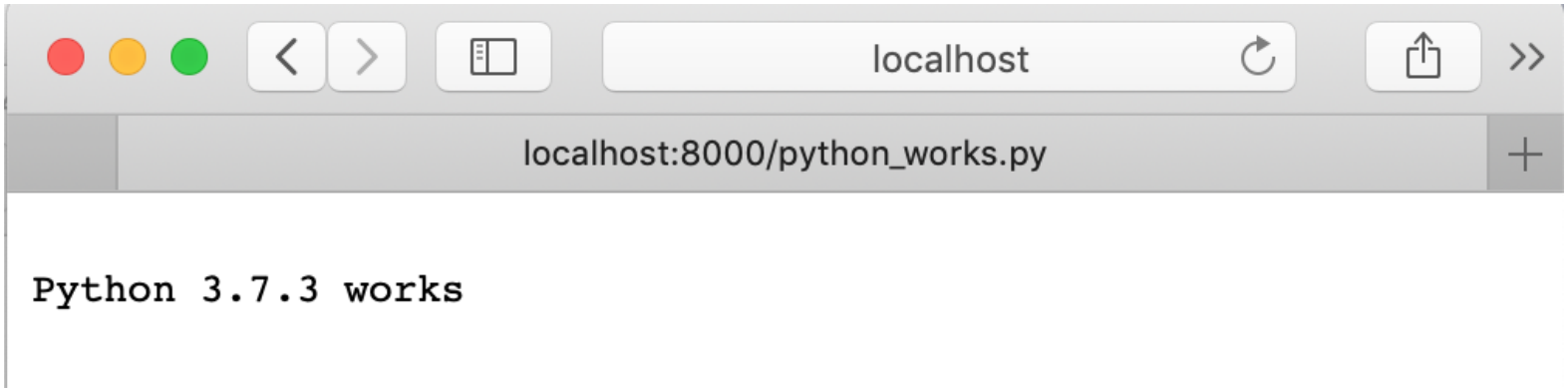
```
PORTNUMBER = 8000
import http.server
import cgitb; cgitb.enable() # CGI error report

server = http.server.HTTPServer
handler = http.server.CGIHTTPRequestHandler
server_address = ("", PORTNUMBER)
handler.cgi_directories = ["/"]

httpd = server(server_address, handler)
try:
    print('welcome to our cgi server')
    print('press ctrl c to shut down the server')
    httpd.serve_forever()
except:
    print('ctrl c pressed, shutting down server')
    httpd.socket.close()
```

running `myserver.py`

The script `python_works.py` is in the directory from which the script `myserver.py` was launched.

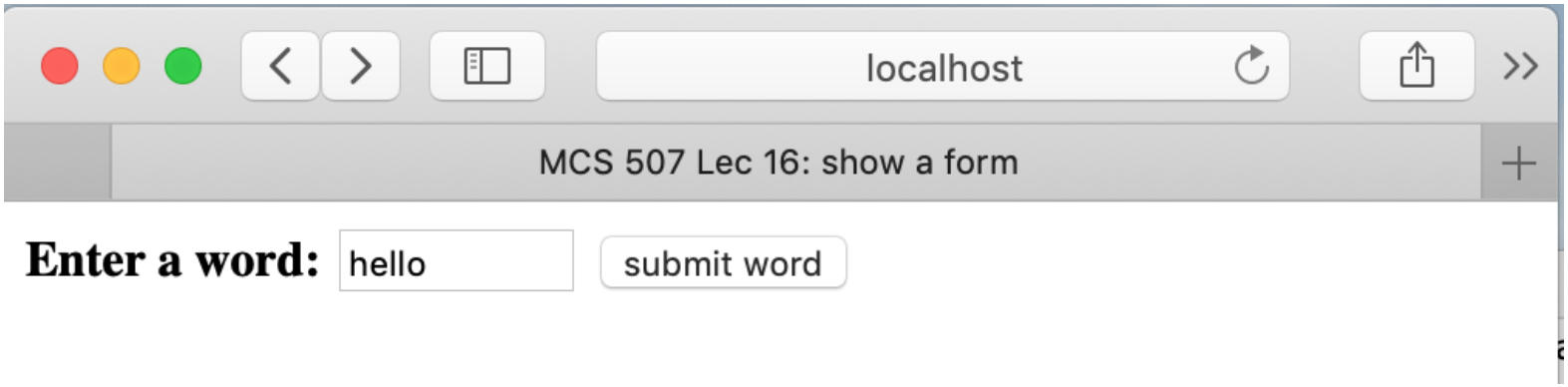


Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

showing a form

The HTML in `show_form.html` brings

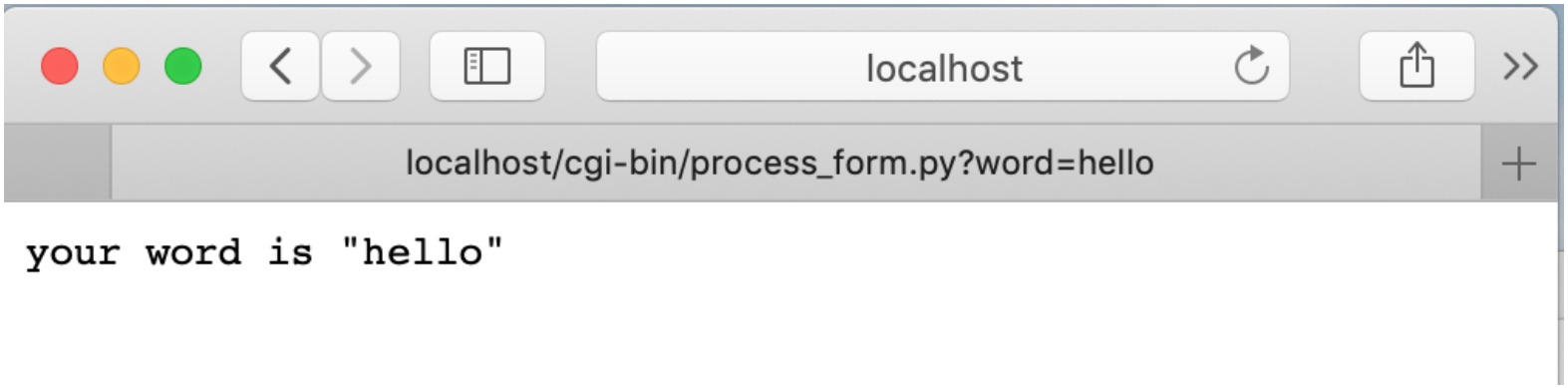


- The word `hello` is entered by the user.
- Pressing submit activates a Python script.

processing a form

Pressing submit brings up a new web page.

The script `process_form.py` gets the word and prints one line into the browser window.



code in show_form.html

```
<html>
<head>
<title> MCS 507 Lec 19: show a form </title>
</head>
<body>
<form action=
 "http://localhost/cgi-bin/process_form.py">
<b>Enter a word:</b>
<input type="text" name="word" size ="10">
<input type="submit" value="submit word">
</form>
</body>
</html>
```

code in process_form.py

```
#!< path to python interpreter >
"""

This script is called when the user presses the
submit button in the form show_form.html.
"""
import cgi
FORM = cgi.FieldStorage()
print("Content-Type: text/plain\n")
try:
    WORD = FORM['word'].value
    SHOW = 'your word is '
    SHOW = SHOW + '\"' + WORD + '\"'
    print(SHOW)
except KeyError:
    print("please enter a word")
```

Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

forms in Python scripts

writing HTML

What we did before:

1. Form element in HTML page triggers action.
2. Action triggered by submit button
   is defined by a Python script.

Advantage/disadvantage:

+ Separate HTML layout and actions.
− Submit brings up a new page each time.

The script `show_process_form.py` has the same functionality as the `show_form.html` and the `process_form.py`.

printing a header

```
#!< path to python interpreter >

def print_header(title):
    """
    Writes title and header of page.
    """
    print("""Content-type: text/html

<html>
<head>
<title>%s</title>
</head>
<body>""" % title)
```

the main function

The function `main()` of `show_process_form.py`:

```
def main():
    """
    Show and process a form.
    """
    print_header("MCS 507 Lec 16: forms")
    show_form_and_get_word()

main()
```

The function `show_form_and_get_word()`

- first checks if the form has data;
- if the form has data, then the data will be printed;
- otherwise, the form to enter a word is printed.

Notice: the function calls the script when the data is submitted.

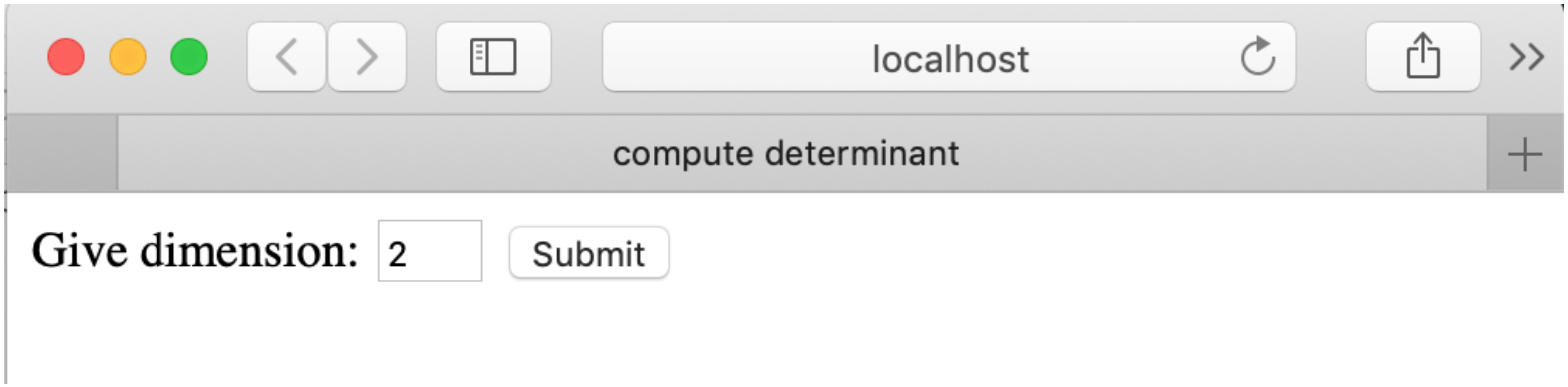
the function show_process_form

```
def show_form_and_get_word():
    """
    Prints the form to get a word.
    """
    import cgi
    form = cgi.FieldStorage()
    if "word" in form:
        print("""<p>Your word is <em>%s</em></p>
        """ % form["word"].value)
    else:
        print("""
<form method = "post"
 method = "show_process_form.py">
  <b>Enter a word:</b>
  <input type = "text" name = "word">
  <input type = "submit" value = "submit word">
</form>""")
    print("</body></html>\n")
```

Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

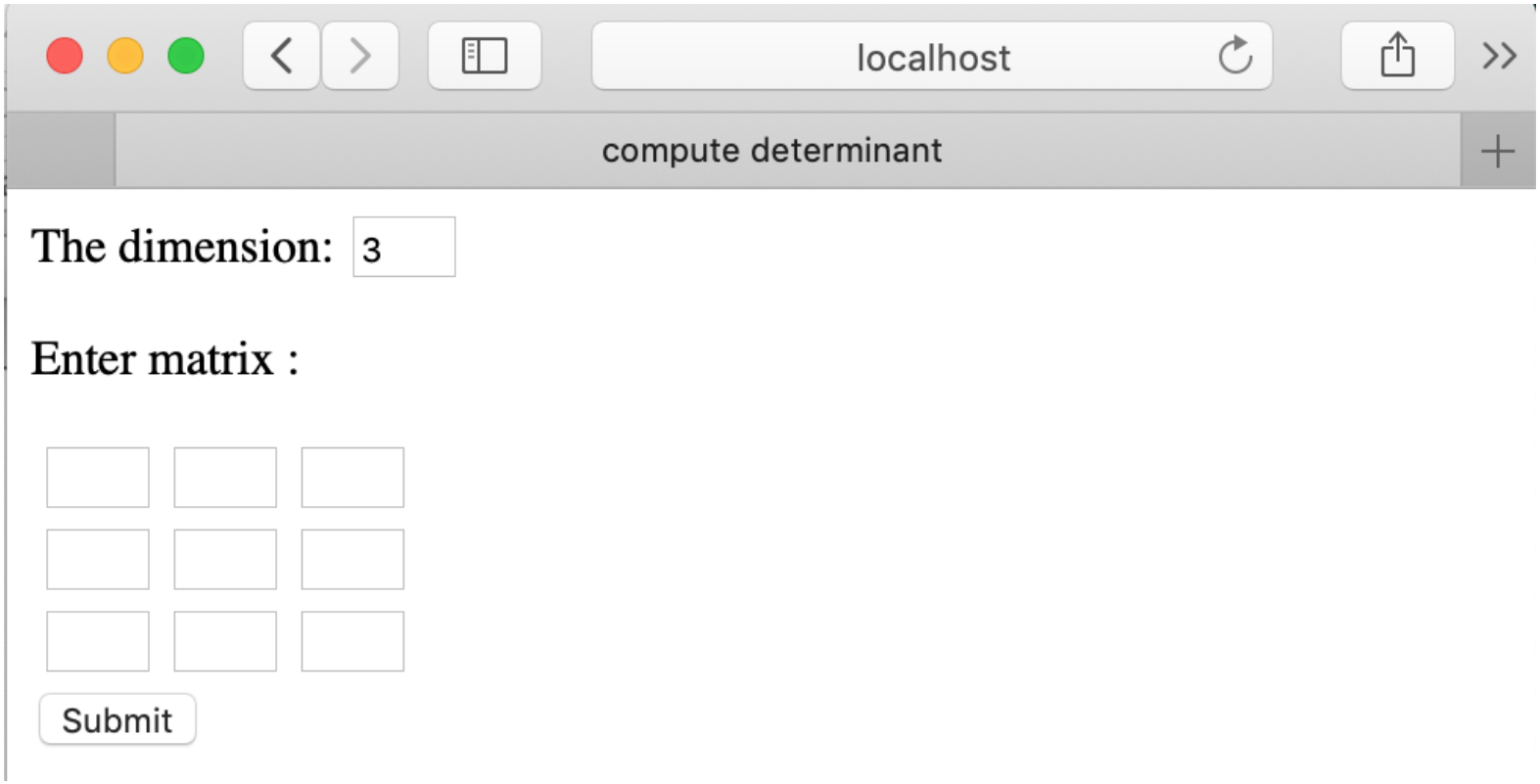
prompting for a dimension



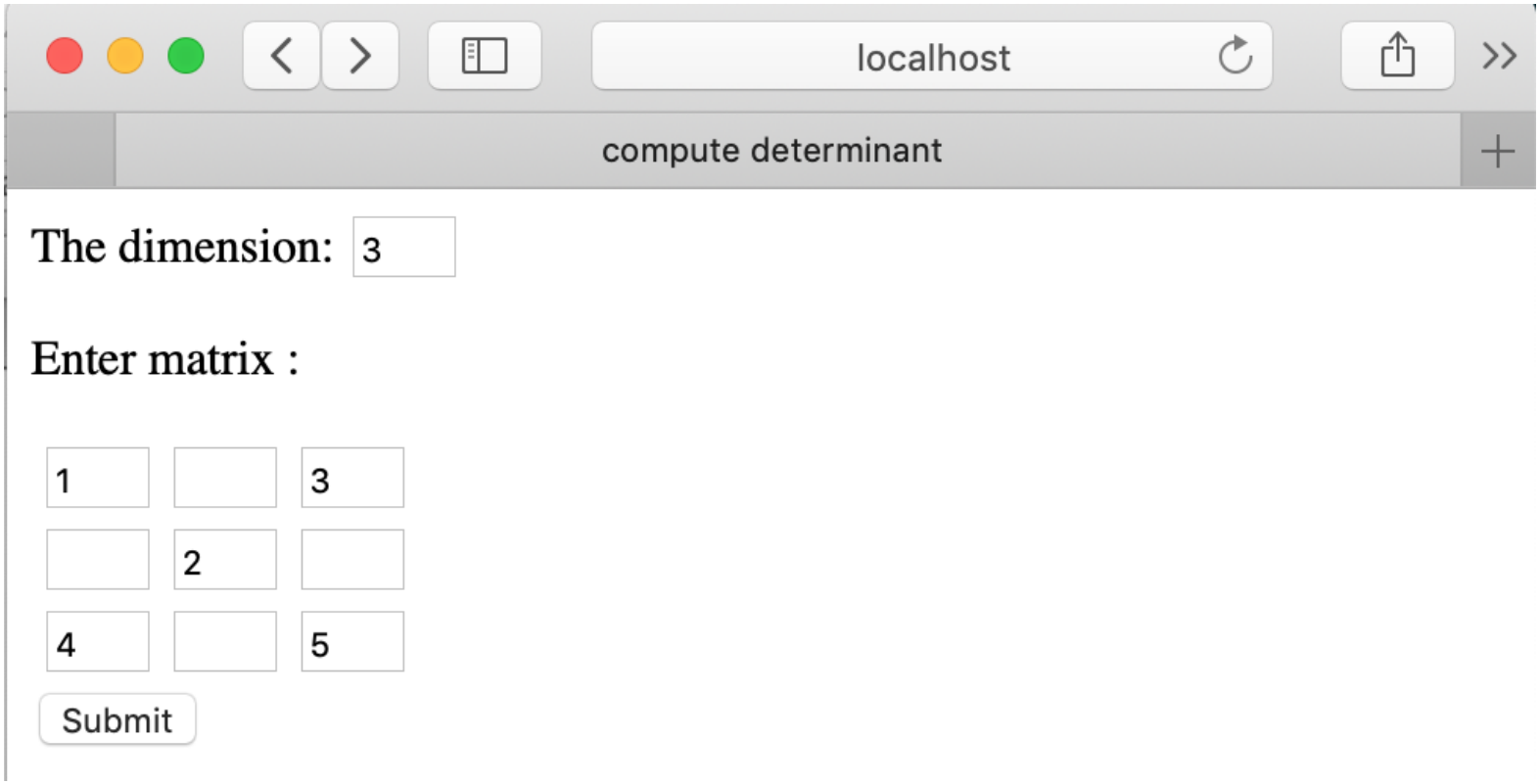
The default value is set at 2.

After the user changes it to 3, and hits submit...

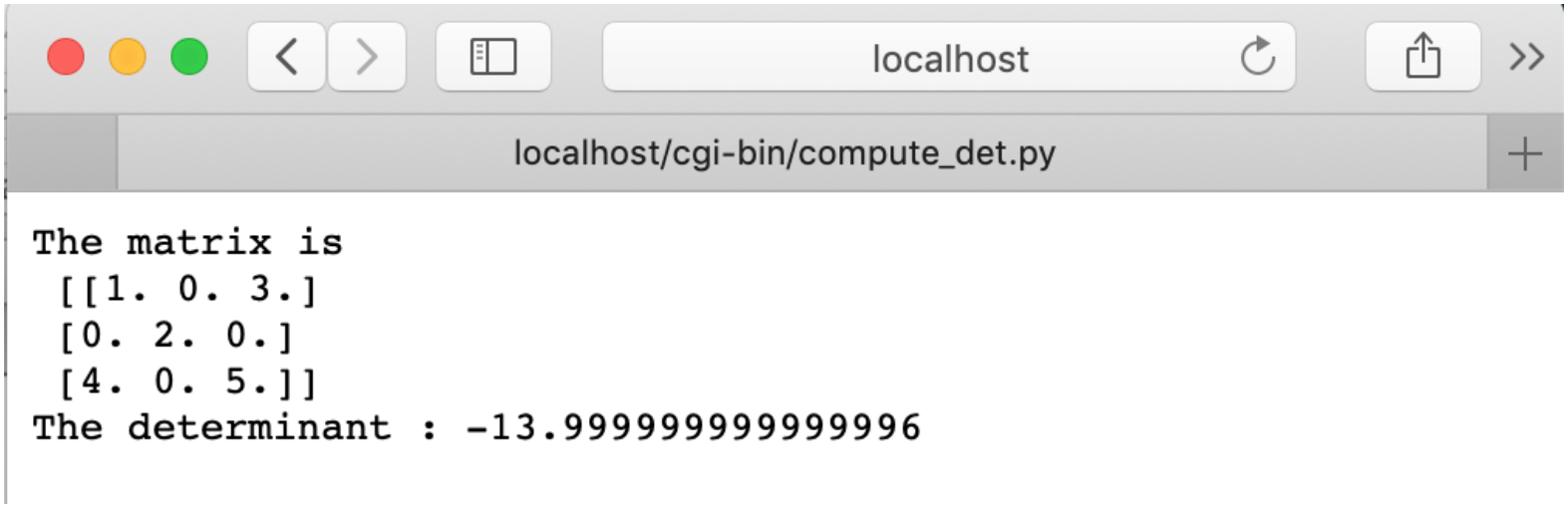
prompting for a matrix



filling in values



computing the determinant



Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

script for web determinant

The script is a sequence of two forms:

1. we ask for *the dimension* of the matrix
   The dimension is then used to generate a table of input elements to give the entries of the matrix.

2. we ask for *the entries* of the matrix
   The second form is processed by the same script.
   The action of this form brings up a new page with the matrix and the determinant.

The dimension is passed from one form to the other
through an input element.

main() in web_det.py

```
def main():
    """
    Web interface to compute a determinant.
    """
    print_header("compute determinant")
    form = cgi.FieldStorage()
    if "dim" not in form:
        ask_dimension()
    else:
        dim = int(form["dim"].value)
        ask_matrix(dim)
    print("</body></html>\n")
```

default values

```
def ask_dimension():
    """
    Form to enter the dimension.
    """
    print("""<form method = "post"
                    action = "web_det.py">
    <p>
      Give dimension:
      <input type = "text" name = "dim"
             size = 4 value = 2>
      <input type = "submit">
    </p>
    </form>""")
```

asking for a matrix

```
def ask_matrix(dim):
    """
    Form to enter a dim-by-dim matrix.
    """
    print("""<form method = "post"
                    action = "compute_det.py">""")
    print("""The dimension:
       <input type = "text" name = "dim"
              size = 4 value = %d>""" % dim)
    print("<p>Enter matrix :</p>")
```

a table in a form

```
print("<table>")
for i in range(0, dim):
    print("<tr>")
    for j in range(0, dim):
        print("""<td><input type = "text"
        name = %d,%d size = 4></td>""" % (i, j))
print("</table>")
print("""<input type = "submit">""")
print("</form>")
```

The name of each input element is `(i,j)`.

Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

the script `compute_det.py`

```
def determinant(form, dim):
    """
    Returns the determinant of the matrix
    available in the form.
    """
    from numpy import zeros
    from numpy.linalg import det
    mat = zeros((dim, dim), float)
    for i in range(0, dim):
        for j in range(0, dim):
            info = "%d,%d" % (i, j)
            if info in form:
                mat[i, j] = float(form[info].value)
    print("The matrix is\n", mat)
    return det(mat)
```

the main function

```
def main():
    """
    Computing the determinant of a matrix
    available via a CGI form.
    """
    import cgi
    print("Content-type: text/plain\n")
    form = cgi.FieldStorage()
    if "dim" in form:
        dim = int(form["dim"].value)
        detval = determinant(form, dim)
        print("The determinant :", detval)

main()
```

Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

Personalized Web Browsing

more clever cgi scripts

Storing information about a client:

1. previous selections made
   $\rightarrow$ passing data from one script to another
2. personal information
   $\rightarrow$ identification and passwords
3. information from previous visits
   $\rightarrow$ counting number of visits

Potential applications:

1. customize displayed content
2. store encrypted password for fast access
3. personalized pricing ...

cookies store information about client

Cookies are data

- stored by web server on client computer,
- managed by the browser.

Using the `cookies` module:

```
>>> from http import cookies
>>> c = cookies.SimpleCookie()
>>> c['L'] = 16
>>> c['date'] = 'Wed 2 Oct 2019'
>>> print(c)
Set-Cookie: L=16
Set-Cookie: date="Wed 2 Oct 2019"
```

Cookies are objects like dictionaries.
Reserved keys: `expires` and `path`.

computing with cookies

To compute with values stored in cookies:

- The value of cookie is the attribute `value`.
- The type of the `value` is a string.

To update the number of visits stored in the cookie `cnt`:

```
>>> from http import cookies
>>> cnt = cookies.SimpleCookie()
>>> cnt['visits'] = 0
>>> cnt['visits']
<Morsel: visits=0>
>>> cnt['visits'].value
'0'
>>> cnt['visits'] = 1 + int(cnt['visits'].value)
>>> cnt['visits'].value
'1'
```

The return of `value` is a string.

the environment variable HTTP_COOKIE

We can initialize a cookie with keys and values:

```
>>> from http.cookies import SimpleCookie
>>> c = SimpleCookie('x=1; y=2')
>>> c['x'].value
'1'
>>> c['y'].value
'2'
```

The environment variable HTTP_COOKIE is set by the web server.
When we rerun a web page the cookies are retrieved as follows:

```
>>> from os import environ
>>> environ['HTTP_COOKIE'] = 'z = 3'
>>> d = SimpleCookie(environ['HTTP_COOKIE'])
>>> d['z'].value
'3'
```

Web Interfaces

1. Running a Python Script through Browsers
   - client/server, HTTP, and Apache
   - running a simple CGI server
   - processing forms with Python scripts
   - Python code to write HTML
2. a Web Interface for the Determinant
   - dynamic interactive forms
   - passing data between forms
   - computing the determinant
3. Cookies
   - personalized web browsing
   - using cookies: count number of visits

using cookies

We use a cookie to count the number of visits.

Write a script to

1. retrieve cookie, initialize counter to zero,
2. or increment the value of counter by one,
3. and display counter value on the page.

The name of the script is `cookie_counter.py`.

1. The browser settings must accept cookies.
2. Each browser has its own cookies.

Via the `Preferences` in the browser, we can manage the cookies.

code for cookie_counter.py

```
#!< path to python interpreter >

import os
from http.cookies import SimpleCookie

def increment():
    """
    Retrieves cookie, either initializes counter,
    or increments the counter by one.
    """
    if 'HTTP_COOKIE' in os.environ:
        cnt = SimpleCookie(os.environ['HTTP_COOKIE'])
    else:
        cnt = SimpleCookie()
    if 'visits' not in cnt:
        cnt['visits'] = 0
    else:
        cnt['visits'] = 1 + int(cnt['visits'].value)
    return cnt
```

the `main()` in `cookie_counter.py`

```
def main():
    """
    Retrieves a cookie and writes
    the value of counter to the page,
    after incrementing the counter.
    """
    counter = increment()
    print(counter)
    print("Content-Type: text/plain\n")
    print("counter: %s" % counter['visits'].value)

main()
```

Summary + Exercises

Apache and Python are the A and P in LAMP.

0. Verify that Apache runs on your computer.
1. Write a web interface to compute the greatest common divisor of two user given numbers.
2. Make an HTML form that prompts the user to enter a polynomial with integer coefficients in the variable x. The action in the form calls a script that uses SymPy to extract the vector of coefficients and the degree.
3. Extend the previous exercise to make a web interface for MPSolve (see Lecture 6).