Outline

1 Python Scripts in Browsers
   Apache and LAMP
dynamic web pages

2 Dynamic Web Pages
   HTTP: GET and POST methods
   FROM to accept user data
   server-side scripting

3 Introduction to CGI
   hello localhost
   a login page
   Python to generate HTML

MCS 260 Lecture 36
Introduction to Computer Science
Jan Verschelde, 17 November 2008
Python Scripts in Browsers
Apache and LAMP
Dynamic Web Pages
HTTP: GET and POST methods
FROM to accept user data
server-side scripting
Introduction to CGI
hello localhost
a login page
Python to generate HTML
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.
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](http://www.apache.org).
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](http://www.apache.org).
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.
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.
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.
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 10.5 (Leopard)

Apache is platform independent. We will demonstrate on a Mac OS X 10.5 (Leopard).

1. Apache is already installed on Mac Leopard, launch Safari with http://localhost/ to verify.
2. To enable web sharing, select Sharing from the System Preferences.
3. Instead of public_html, the Sites directory is where Mac users store their web pages.
4. 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.
Apache is platform independent. We will demonstrate on a Mac OS X 10.5 (Leopard).

1. **Apache is already installed on Mac Leopard, launch Safari with** http://localhost/ **to verify.**

2. To enable web sharing, select Sharing from the System Preferences.

3. Instead of public_html, the Sites directory is where Mac users store their web pages.

4. 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.
Running Apache
on a Mac OS X 10.5 (Leopard)

Apache is platform independent.
We will demonstrate on a Mac OX 10.5 (Leopard).

1. Apache is already installed on Mac Leopard, launch Safari with http://localhost/ to verify.

2. To enable web sharing, select Sharing from the System Preferences.

3. Instead of public_html, the Sites directory is where Mac users store their web pages.

4. Instead of /var/www/cgi-bin, CGI scripts are in /Library/WebServer/CGI-Executables.

Using localhost we remain working offline.
Apache is platform independent. We will demonstrate on a Mac OS X 10.5 (Leopard).

1. Apache is already installed on Mac Leopard, launch Safari with http://localhost/ to verify.
2. To enable web sharing, select Sharing from the System Preferences.
3. Instead of public_html, the Sites directory is where Mac users store their web pages.
4. 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.
Apache is platform independent. We will demonstrate on a Mac OX 10.5 (Leopard).

1. Apache is already installed on Mac Leopard, launch Safari with http://localhost/ to verify.

2. To enable web sharing, select Sharing from the System Preferences.

3. Instead of public_html, the Sites directory is where Mac users store their web pages.

4. 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.
Running Apache
on a Mac OS X 10.5 (Leopard)

Apache is platform independent.
We will demonstrate on a Mac OX 10.5 (Leopard).

1. Apache is already installed on Mac Leopard, launch Safari with http://localhost/ to verify.
2. To enable web sharing, select Sharing from the System Preferences.
3. Instead of public_html, the Sites directory is where Mac users store their web pages.
4. 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.
Apache on MS Windows

Apache is platform independent.

On a Windows 2000 Dell Latitude Laptop:

2. Login as Administrator and install Apache binary.
3. Connect to http://127.0.0.1:8080/
4. html files are in c:/Program Files/Apache " + "Software Foundation/Apache2.2/htdocs"
5. scripts are in "c:/Program Files/Apache " + "Software Foundation/Apache2.2/cgi-bin"

Important: localhost = 127.0.0.1:8080.
To run Apache, you may have to turn off the IIS (Internet Information Server) of Microsoft.
Apache on MS Windows

Apache is platform independent.

On a Windows 2000 Dell Latitude Laptop:

1. Download binaries of Apache webserver from
   \[http://<mirror>/pub/apache/httpd/binaries/win32\]

2. Login as Administrator and install Apache binary.

3. Connect to \[http://127.0.0.1:8080/\]

4. html files are in \[c:/Program Files/Apache " + "Software Foundation/Apache2.2/htdocs"\]

5. scripts are in \"c:/Program Files/Apache " + "Software Foundation/Apache2.2/cgi-bin"\]

Important: localhost = 127.0.0.1:8080.

To run Apache, you may have to turn off the IIS (Internet Information Server) of Microsoft.
Apache on MS Windows

Apache is platform independent.

On a Windows 2000 Dell Latitude Laptop:

2. Login as Administrator and install Apache binary.
3. Connect to http://127.0.0.1:8080/
4. html files are in 
   c:/Program Files/Apache Software Foundation/Apache2.2/htdocs
5. scripts are in 
   "c:/Program Files/Apache Software Foundation/Apache2.2/cgi-bin"

Important: localhost = 127.0.0.1:8080.
To run Apache, you may have to turn off the IIS (Internet Information Server) of Microsoft.
Apache on MS Windows

Apache is platform independent.

On a Windows 2000 Dell Latitude Laptop:

1. Download binaries of Apache webserver from
   http://<mirror>/pub/apache/httpd/binaries/win32

2. Login as Administrator and install Apache binary.

3. Connect to http://127.0.0.1:8080/

4. html files are in c:/Program Files/Apache " +
   "Software Foundation/Apache2.2/htdocs"

5. scripts are in "c:/Program Files/Apache " +
   "Software Foundation/Apache2.2/cgi-bin"

Important: localhost = 127.0.0.1:8080.
To run Apache, you may have to turn off
the IIS (Internet Information Server) of Microsoft.
Apache on MS Windows

Apache is platform independent.

On a Windows 2000 Dell Latitude Laptop:

1. Download binaries of Apache webserver from
   http://<mirror>/pub/apache/httpd/binaries/win32

2. Login as Administrator and install Apache binary.

3. Connect to http://127.0.0.1:8080/

4. html files are in c:/Program Files/Apache " +
   "Software Foundation/Apache2.2/htdocs"

5. scripts are in "c:/Program Files/Apache " +
   "Software Foundation/Apache2.2/cgi-bin"

Important: localhost = 127.0.0.1:8080.
To run Apache, you may have to turn off
the IIS (Internet Information Server) of Microsoft.
Apache on MS Windows

Apache is platform independent.

On a Windows 2000 Dell Latitude Laptop:

1. Download binaries of Apache webserver from
   http://<mirror>/pub/apache/httpd/binaries/win32
2. Login as Administrator and install Apache binary.
3. Connect to http://127.0.0.1:8080/
4. html files are in c:/Program Files/Apache " + "Software Foundation/Apache2.2/htdocs"
5. scripts are in "c:/Program Files/Apache " + "Software Foundation/Apache2.2/cgi-bin"

Important: localhost = 127.0.0.1:8080.
To run Apache, you may have to turn off
the IIS (Internet Information Server) of Microsoft.
Apache on MS Windows

Apache is platform independent.

On a Windows 2000 Dell Latitude Laptop:


2. Login as Administrator and install Apache binary.

3. Connect to http://127.0.0.1:8080/

4. html files are in c:/Program Files/Apache " + "Software Foundation/Apache2.2/htdocs"

5. scripts are in "c:/Program Files/Apache " + "Software Foundation/Apache2.2/cgi-bin"

Important: localhost = 127.0.0.1:8080.
To run Apache, you may have to turn off the IIS (Internet Information Server) of Microsoft.
Python Scripts in Browsers

Apache and LAMP

dynamic web pages

Dynamic Web Pages

HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML
Python Works!

our first CGI script

The Python script `python_works.py`:

```python
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python
print "Content-Type: text/plain\n\n"
print "Python works"
```

The first line in the script is the full path name of the Python interpreter.
Python Works!

our first CGI script

The Python script `python_works.py`:

```
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python
print "Content-Type: text/plain\n\n"
print "Python works"
```

The first line in the script is the full path name of the Python interpreter.
Python Works!
our first CGI script

The Python script `python_works.py`:

```python
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python
print "Content-Type: text/plain\n\n" print "Python works"
```

The first line in the script is the full path name of the Python interpreter.
**Python Scripts in Browsers**
Apache and LAMP
dynamic web pages

**Dynamic Web Pages**
HTTP: GET and POST methods
FROM to accept user data
server-side scripting

**Introduction to CGI**
hello localhost
a login page
Python to generate HTML
Dynamic Web Pages

HTTP: GET and POST methods

HTTP (HyperText Transfer Protocol) determines request-response communication between web browser and web server.

Methods of HTTP:

GET method is a request for a static resource, such as an HTML page.
Just typing the URL of the requested web page invokes the GET method.

POST method is a request for a dynamic resource, with input parameters of the request contained within the body of the request.

The GET and POST methods are most commonly used.
Dynamic Web Pages

HTTP: GET and POST methods

HTTP (HyperText Transfer Protocol) determines request-response communication between web browser and web server.

Methods of HTTP:

**GET** method is a request for a static resource, such as an HTML page. Just typing the URL of the requested web page invokes the **GET** method.

**POST** method is a request for a dynamic resource, with input parameters of the request contained within the body of the request.

The **GET** and **POST** methods are most commonly used.
Dynamic Web Pages

HTTP: GET and POST methods

HTTP (HyperText Transfer Protocol) determines request-response communication between web browser and web server.

Methods of HTTP:

**GET** method is a request for a static resource, such as an HTML page. Just typing the URL of the requested web page invokes the **GET** method.

**POST** method is a request for a dynamic resource, with input parameters of the request contained within the body of the request.

The **GET** and **POST** methods are most commonly used.
Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

**HTML**  `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

**HEAD** specifies header information of a document.

**TITLE** specifies title of the document.

**BODY** contains the body text of the document.

**FONT** used to alter font size and color of text.

**H1** to display headings of type 1, other heading elements are **H2** and **H3**.

**P** defines a paragraph.

**OL** ordered list, unordered list is **UL**.

**LI** list element in ordered or unordered list.

Learn **HTML** by looking at source of web pages.
Commonly used elements in HTML documents:

**HTML** `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

**HEAD** specifies header information of a document.

**TITLE** specifies title of the document.

**BODY** contains the body text of the document.

**FONT** used to alter font size and color of text.

**H1** to display headings of type 1, other heading elements are **H2** and **H3**.

**P** defines a paragraph.

**OL** ordered list, unordered list is **UL**.

**LI** list element in ordered or unordered list.

Learn **HTML** by looking at source of web pages.
Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

- **HTML** `<HTML>` marks start of HTML document, and `</HTML>` marks the end.
- **HEAD** specifies header information of a document.
- **TITLE** specifies title of the document.
- **BODY** contains the body text of the document.
- **FONT** used to alter font size and color of text.
- **H1** to display headings of type 1, other heading elements are **H2** and **H3**.
- **P** defines a paragraph.
- **OL** ordered list, unordered list is **UL**.
- **LI** list element in ordered or unordered list.

Learn **HTML** by looking at source of web pages.
Elements of HTML
HyperText Markup Language

Commonly used elements in HTML documents:

- **HTML** `<HTML>` marks start of HTML document, and `</HTML>` marks the end.
- **HEAD** specifies header information of a document.
- **TITLE** specifies title of the document.
- **BODY** contains the body text of the document.
- **FONT** used to alter font size and color of text.
  - **H1** to display headings of type 1, other heading elements are **H2** and **H3**.
  - **P** defines a paragraph.
  - **OL** ordered list, unordered list is **UL**.
  - **LI** list element in ordered or unordered list.

Learn **HTML** by looking at source of web pages.
Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

- **HTML** `<HTML>` marks start of HTML document, and `</HTML>` marks the end.
- **HEAD** specifies header information of a document.
- **TITLE** specifies title of the document.
- **BODY** contains the body text of the document.
- **FONT** used to alter font size and color of text.
  - **H1** to display headings of type 1, other heading elements are **H2** and **H3**.
  - **P** defines a paragraph.
  - **OL** ordered list, unordered list is **UL**.
  - **LI** list element in ordered or unordered list.

Learn **HTML** by looking at source of web pages.
Elements of HTML
HyberText Markup Language

Commonly used elements in HTML documents:

**HTML** `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

**HEAD** specifies header information of a document.

**TITLE** specifies title of the document.

**BODY** contains the body text of the document.

**FONT** used to alter font size and color of text.

**H1** to display headings of type 1, other heading elements are **H2** and **H3**.

**P** defines a paragraph.

**OL** ordered list, unordered list is **UL**.

**LI** list element in ordered or unordered list.

Learn **HTML** by looking at source of web pages.
Elements of HTML
HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1, other heading elements are H2 and H3.

P defines a paragraph.

OL ordered list, unordered list is UL.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.
Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

**HTML** `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

**HEAD** specifies header information of a document.

**TITLE** specifies title of the document.

**BODY** contains the body text of the document.

**FONT** used to alter font size and color of text.

**H1** to display headings of type 1, other heading elements are **H2** and **H3**.

**P** defines a paragraph.

**OL** ordered list, unordered list is **UL**.

**LI** list element in ordered or unordered list.

Learn **HTML** by looking at source of web pages.
Elements of HTML
HyperText Markup Language

Commonly used elements in HTML documents:

**HTML**  
`<HTML>` marks start of HTML document, and `</HTML>` marks the end.

**HEAD** specifies header information of a document.

**TITLE** specifies title of the document.

**BODY** contains the body text of the document.

**FONT** used to alter font size and color of text.

**H1** to display headings of type 1, other heading elements are **H2** and **H3**.

**P** defines a paragraph.

**OL** ordered list, unordered list is **UL**.

**LI** list element in ordered or unordered list.

Learn **HTML** by looking at source of web pages.
dynamic web pages
introduction to CGI

1. Python Scripts in Browsers
   Apache and LAMP
dynamic web pages

2. Dynamic Web Pages
   HTTP: GET and POST methods
FROM to accept user data
server-side scripting

3. Introduction to CGI
   hello localhost
   a login page
   Python to generate HTML
Accepting Data from Users

To accept data from users, three elements are generally used:

**FORM** contains all the code related to a form. Its purpose is to accept user input in a systematic and structured manner.

**INPUT** specifies the code used to create the form controls that accept user input.

**SELECT** used to display lists in a form.

Designing and creating interactive web pages is similar to GUI design.
Accepting Data from Users

To accept data from users, three elements are generally used:

**FORM** contains all the code related to a form. Its purpose is to accept user input in a systematic and structured manner.

**INPUT** specifies the code used to create the form controls that accept user input.

**SELECT** used to display lists in a form.

Designing and creating interactive web pages is similar to GUI design.
Accepting Data from Users

To accept data from users, three elements are generally used:

**FORM** contains all the code related to a form. Its purpose is to accept user input in a systematic and structured manner.

**INPUT** specifies the code used to create the form controls that accept user input.

**SELECT** used to display lists in a form.

Designing and creating interactive web pages is similar to GUI design.
Accepting Data from Users

To accept data from users, three elements are generally used:

- **FORM** contains all the code related to a form. Its purpose is to accept user input in a systematic and structured manner.

- **INPUT** specifies the code used to create the form controls that accept user input.

- **SELECT** used to display lists in a form.

Designing and creating interactive web pages is similar to GUI design.
A form is a collection of text boxes, radio buttons, check boxes, and buttons.

Two attributes of a form:

- **METHOD** is GET or POST.
- **ACTION** is typically used to specify the code that will process the input data.

Syntax:

```html
<FORM METHOD="GET_or_POST" ACTION="file_name">
    code_of_the_form
</FORM>
```
A form is a collection of text boxes, radio buttons, check boxes, and buttons.

Two attributes of a form:

- **METHOD** is GET or POST.
- **ACTION** is typically used to specify the code that will process the input data.

Syntax:

```
<FORM METHOD="GET_or_POST" ACTION="file_name">
  code_of_the_form
</FORM>
```
A form is a collection of text boxes, radio buttons, check boxes, and buttons.

Two attributes of a form:

- **METHOD** is GET or POST.
- **ACTION** is typically used to specify the code that will process the input data.

Syntax:

```
<FORM METHOD="GET_or_POST" ACTION="file_name">
  code_of_the_form
</FORM>
```
The `INPUT` element is specified inside a `FORM` element.

The `INPUT` elements consist of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

- **TYPE** specifies type of control to accept user input.
- **NAME** specifies name of a control, for identification.
- **VALUE** holds value entered by user, or default.

Five types of control:
(1) submit button; (2) text boxes; (3) radio buttons;
(4) check boxes; (5) combo boxes.
The **INPUT** element is specified inside a **FORM** element.

The **INPUT** elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

- **TYPE** specifies type of control to accept user input.
- **NAME** specifies name of a control, for identification.
- **VALUE** holds value entered by user, or default.

Five types of control:

1. submit button;
2. text boxes;
3. radio buttons;
4. check boxes;
5. combo boxes.
The **INPUT** element is specified inside a **FORM** element.

The **INPUT** elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

- **TYPE** specifies type of control to accept user input.
- **NAME** specifies name of a control, for identification.
- **VALUE** holds value entered by user, or default.

Five types of control:

1. submit button;
2. text boxes;
3. radio buttons;
4. check boxes;
5. combo boxes.
The **INPUT** element is specified inside a **FORM** element.

The **INPUT** elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

- **TYPE** specifies type of control to accept user input.
- **NAME** specifies name of a control, for identification.
- **VALUE** holds value entered by user, or default.

Five types of control:

1. submit button
2. text boxes
3. radio buttons
4. check boxes
5. combo boxes
The **INPUT** element is specified inside a **FORM** element.

The **INPUT** elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

- **TYPE** specifies type of control to accept user input.
- **NAME** specifies name of a control, for identification.
- **VALUE** holds value entered by user, or default.

Five types of control:

1. submit button;
2. text boxes;
3. radio buttons;
4. check boxes;
5. combo boxes.
the **INPUT** element

The **INPUT** element is specified inside a **FORM** element.

The **INPUT** elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

- **TYPE** specifies type of control to accept user input.
- **NAME** specifies name of a control, for identification.
- **VALUE** holds value entered by user, or default.

Five types of control:

1. submit button;
2. text boxes;
3. radio buttons;
4. check boxes;
5. combo boxes.
Python Scripts in Browsers
Apache and LAMP
dynamic web pages

Dynamic Web Pages
HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI
hello localhost
a login page
Python to generate HTML
Client/Server Scripting

We distinguish between

• **Client-Side Scripting**: processed by browser advantage: saves time of the server

• **Server-Side Scripting**: processed by server needed for synchronization, modification of data server time is then the time

Python is a powerful server-side scripting language. The \texttt{cgi} module has to be imported, in order to communicate the data from client.
Client/Server Scripting

We distinguish between

- Client-Side Scripting: processed by browser advantage: saves time of the server
- Server-Side Scripting: processed by server needed for synchronization, modification of data server time is then *the* time

Python is a powerful server-side scripting language. The `cgi` module has to be imported, in order to communicate the data from client.
Client/Server Scripting

We distinguish between

- **Client-Side Scripting**: processed by browser advantage: saves time of the server
- **Server-Side Scripting**: processed by server needed for synchronization, modification of data server time is then *the* time

Python is a powerful server-side scripting language. The `cgi` module has to be imported, in order to communicate the data from client.
Python Scripts in Browsers
Apache and LAMP
dynamic web pages

Dynamic Web Pages
HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI
hello localhost
a login page
Python to generate HTML
The script **hello_world.py**:

```python
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
   - MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be saved in the `cgi-bin` directory, usually in `/var/www/cgi-bin`. 
our first CGI script

hello world once more

The script **hello_world.py**:
```
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
   MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the `cgi-bin` directory, usually in `/var/www/cgi-bin`. 
our first CGI script
hello world once more

The script `hello_world.py`:
```python
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
   MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the `cgi-bin` directory, usually in `/var/www/cgi-bin`. 
our first CGI script
hello world once more

The script hello_world.py:

```
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
   
   MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the *cgi-bin* directory, usually in `/var/www/cgi-bin`. 
our first CGI script
    hello world once more

The script `hello_world.py`:

```python
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
   MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the `cgi-bin` directory, usually in `/var/www/cgi-bin`. 
How do we run `hello_world.py` in a browser?

The localhost in an URL refers to the local machine.

The complete path to run `hello_world.py` is `http://localhost/cgi-bin/hello_world.py`

Leopard (Mac OS X 10.5) comes with Apache:
Go to the sharing panel in system preferences and enable "Web Sharing".
Local Web Servers

about localhost

How do we run `hello_world.py` in a browser?

The `localhost` in an URL refers to the local machine.

The complete path to run `hello_world.py` is:

```
http://localhost/cgi-bin/hello_world.py
```

Leopard (Mac OS X 10.5) comes with Apache:
Go to the sharing panel in system preferences and enable "Web Sharing".
How do we run `hello_world.py` in a browser?

The localhost in an URL refers to the local machine.

The complete path to run `hello_world.py` is `http://localhost/cgi-bin/hello_world.py`

Leopard (Mac OS X 10.5) comes with Apache: Go to the sharing panel in system preferences and enable "Web Sharing".
Local Web Servers

How do we run hello_world.py in a browser?

The localhost in an URL refers to the local machine.

The complete path to run hello_world.py is http://localhost/cgi-bin/hello_world.py

Leopard (Mac OS X 10.5) comes with Apache: Go to the sharing panel in system preferences and enable "Web Sharing".
Python Scripts in Browsers
Apache and LAMP
dynamic web pages

Dynamic Web Pages
HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI
hello localhost
a login page
Python to generate HTML
A Login Page

1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action, the action is implemented with Python script.
A Login Page

1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action, the action is implemented with Python script.
A Login Page

1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action, the action is implemented with Python script.
A Login Page

1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action, the action is implemented with Python script.
A Login Page

1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action, the action is implemented with Python script.
Python Scripts in Browsers
Apache and LAMP
dynamic web pages
Dynamic Web Pages
HTTP: GET and POST methods
FROM to accept user data
server-side scripting
Introduction to CGI
hello localhost
a login page
Python to generate HTML

### The Code

```html
<HTML>
  <HEAD>
    <TITLE>login page</TITLE>
  </HEAD>
  <HR><CENTER>
    <FORM method="POST"
        action="http://localhost/cgi-bin/validate.py">
      <P>Login name:<input type="text"
        name="login"
        value =""/></P>
      <P>Password:<input type="password"
        name="password"
        value =""/></P>
      <P><input type ="submit" value="SUBMIT"></P>
    </FORM>
  </HR>
</CENTER>
</BODY>
```
Python Scripts in Browsers

Dynamic Web Pages

HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the code

the file loginpage.html

```
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>

<HR><CENTER>
<FORM method="POST"
    action="http://localhost/cgi-bin/validate.py">
    <P>Login name:<input type="text"
        name="login"
        value =""/></P>
    <P>Password:<input type="password"
        name="password"
        value =""/></P>
    <P><input type ="submit" value="SUBMIT"></P>
</FORM>

<HR>
</CENTER>

</BODY>
```
the code

the file loginpage.html

```html
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>

<HR><CENTER>
<FORM method="POST" action="http://localhost/cgi-bin/validate.py">
  <P>Login name:<input type="text" name="login" value=""></P>
  <P>Password:<input type="password" name="password" value=""></P>
  <P><input type="submit" value="SUBMIT"></P>
</FORM>

<HR>
</CENTER>
</BODY>
```
the code

the file loginpage.html

<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST" action="http://localhost/cgi-bin/validate.py">
<P>Login name:<input type="text" name="login" value=""></P>

<P>Password:<input type="password" name="password" value=""></P>

<P><input type="submit" value="SUBMIT"></P>
</FORM>
</CENTER>
</BODY>
the code

the file loginpage.html

```html
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST"
    action="http://localhost/cgi-bin/validate.py">
    <P>Login name:<input type="text"
        name="login"
        value =""></P>
    <P>Password:<input type="password"
        name="password"
        value =""></P>
    <P><input type ="submit" value="SUBMIT"></P>
</FORM>
</CENTER>
</BODY>
```
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST"
    action="http://localhost/cgi-bin/validate.py">
    <P>Login name:<input type="text"
        name="login"
        value =""/></P>
    <P>Password:<input type="password"
        name="password"
        value =""/></P>
    <P><input type ="submit" value="SUBMIT"></P>
</FORM>
</CENTER>
</BODY>
</HTML>
Communication with Client

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```python
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```
Communication with Client

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```python
fs = cgi.FieldStorage()
print fs[‘login’].value
print fs[‘password’].value
```
Communication with Client

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```python
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```
Communication with Client

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```python
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```
Communication with Client

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```python
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```
# validate.py

```python
#!/Library/Frameworks/Python.framework/Versions/Current
import cgi
form = cgi.FieldStorage()
print "Content-Type: text/plain\n"
error = False
try:
    n = form[‘login’].value
except KeyError:
    print "please enter your name"
    error = True
try:
    p = form[‘password’].value
except KeyError:
    print "please enter a password"
    error = True
if not error:
    print ‘welcome ’ + n
```
Script validate.py

#!/Library/Frameworks/Python.framework/Versions/Current
import cgi
form = cgi.FieldStorage()
print "Content-Type: text/plain\n"
error = False
try:
    n = form[‘login’].value
except KeyError:
    print "please enter your name"
    error = True
except KeyError:
    print "please enter your password"
    error = True
if not error:
    print ‘welcome ’ + n
Script validate.py

#!/Library/Frameworks/Python.framework/Versions/Current
import cgi
form = cgi.FieldStorage()
print "Content-Type: text/plain\n"
error = False
try:
    n = form['login'].value
except KeyError:
    print "please enter your name"
    error = True
try:
    p = form['password'].value
except KeyError:
    print "please enter a password"
    error = True
if not error:
    print 'welcome ' + n
Script validate.py

#!/Library/Frameworks/Python.framework/Versions/Current
import cgi
form = cgi.FieldStorage()
print "Content-Type: text/plain"
error = False
try:
    n = form[‘login’].value
except KeyError:
    print "please enter your name"
    error = True
try:
    p = form[‘password’].value
except KeyError:
    print "please enter a password"
    error = True
if not error:
    print ‘welcome ’ + n
1. Python Scripts in Browsers
   Apache and LAMP
   dynamic web pages

2. Dynamic Web Pages
   HTTP: GET and POST methods
   FROM to accept user data
   server-side scripting

3. Introduction to CGI
   hello localhost
   a login page
   Python to generate HTML
Displaying Current Date and Time

Running the script `webtime.py` the prompt:

```
Content-type: text/html

<HTML>
<HEAD><TITLE>current date and time</TITLE></HEAD>
<BODY>
Mon Nov 17 11:27:53 2008
</BODY></HTML>
```

The Python script `webtime.py` produces HTML code, containing computed results, e.g. the current time:

```
>>> import time
>>> time.ctime()
'Mon Nov 17 11:27:53 2008'
```
Python to generate HTML

#!/Library/Frameworks/Python.framework/Versions/Current/bin/python

import cgi
import time

def printHeader(title):
    print "Content-type: text/html\n"
    print ""

<HTML>
<HEAD><TITLE>%s</TITLE></HEAD>
<BODY>"" % title

printHeader("current date and time")
print time.ctime(time.time())
print ""></BODY></HTML>"
Python to generate HTML

#!/Library/Frameworks/Python.framework/Versions/Current/bin/python

import cgi
import time

def printHeader(title):
    print "Content-type: text/html\n"
    print ""

    <HTML>
    <HEAD><TITLE>%s</TITLE></HEAD>
    <BODY>"" % title

    printHeader("current date and time")
    print time.ctime(time.time())
    print "</BODY></HTML>"
Python to generate HTML

#!/Library/Frameworks/Python.framework/Versions/Current/bin/python

import cgi
import time

def printHeader(title):
    print "Content-type: text/html\n"
    print ""

<HTML>
<HEAD><TITLE>%s</TITLE></HEAD>
<BODY>"" % title

printHeader("current date and time")
print time.ctime(time.time())
print "</BODY></HTML>"
#!/Library/Frameworks/Python.framework/Versions/Current/bin/python

import cgi
import time

def printHeader(title):
    print "Content-type: text/html\n"
    print ""

<HTML>
<HEAD><TITLE>%s</TITLE></HEAD>
<BODY>"" % title

printHeader("current date and time")
print time.ctime(time.time())
print ""</BODY></HTML>"
Summary + Assignments

We started chapter 13 in *Python Power*

Assignments:

   Is its license compatible with GPL?
2. Make a web interface for the formatting of elapsed time
   (see the first project).
3. Write a CGI script in Python for a currency converter.

The last quiz happens this week.

Last homework collection on Monday 1 December, at 1PM:

#4 of L-27; #1 of L-28; #3, 8 of L-29; #1, 2 of L-30;
#4, 5 of L-31; #1, 4 of L-32 → good for 20 points.