an application

A Virtual University

description of an application

the four pillars in the application

Uploading Files

form of the HTML code to upload

processing uploaded file with CGI script

The CGIHTTPServer Module

testing HTML forms offline

MCS 275 Lecture 33
Programming Tools and File Management
Jan Verschelde, 7 April 2008

an application

A Virtual University

- description of an application
- the four pillars in the application

Uploading Files

- form of the HTML code to upload
- processing uploaded file with CGI script

The CGIHTTPServer Module

- testing HTML forms offline

A Virtual University

description of an application

The running example in *Making Use of Python* is the administration of `Techsity University`.

The application rests on four pillars:

1. web forms with CGI scripts,
2. information management with databases,
3. multiple servers to handle the load,
4. multithreaded servers handle many clients.

→ chapters 10, 11, 12, and 13 in the book.

A Virtual University

description of an application

The running example in *Making Use of Python*
is the administration of `Techsity University`.

The application rests on four pillars:

1. web forms with CGI scripts,
2. information management with databases,
3. multiple servers to handle the load,
4. multithreaded servers handle many clients.

→ chapters 10, 11, 12, and 13 in the book.

A Virtual University

description of an application

The running example in *Making Use of Python*
is the administration of `Techsity University`.

The application rests on four pillars:

1. web forms with CGI scripts,
2. information management with databases,
3. multiple servers to handle the load,
4. multithreaded servers handle many clients.

→ chapters 10, 11, 12, and 13 in the book.

A Virtual University

description of an application

The running example in *Making Use of Python* is the administration of `Techsity University`.

The application rests on four pillars:

1. web forms with CGI scripts,
2. information management with databases,
3. multiple servers to handle the load,
4. multithreaded servers handle many clients.

→ chapters 10, 11, 12, and 13 in the book.

A Virtual University

description of an application

The running example in *Making Use of Python* is the administration of `Techsity University`.

The application rests on four pillars:

1. web forms with CGI scripts,
2. information management with databases,
3. multiple servers to handle the load,
4. multithreaded servers handle many clients.

→ chapters 10, 11, 12, and 13 in the book.

A Virtual University

description of an application

The running example in *Making Use of Python*
is the administration of `Techsity University`.

The application rests on four pillars:

1. web forms with CGI scripts,
2. information management with databases,
3. multiple servers to handle the load,
4. multithreaded servers handle many clients.

→ chapters 10, 11, 12, and 13 in the book.

an application

A Virtual University

- description of an application
- the four pillars in the application

Uploading Files

- form of the HTML code to upload
- processing uploaded file with CGI script

The CGIHTTPServer Module

- testing HTML forms offline

CGI Programming

web interfaces

Course Administration:

- browsing of the course catalog,
- answering queries about courses,
- course registration.

Online Courses:

- browsing through the course materials,
- downloading notes and slides,
- online class and lab interactions,
- uploading answers to assignments.

CGI Programming

web interfaces

Course Administration:

- browsing of the course catalog,
- answering queries about courses,
- course registration.

Online Courses:

- browsing through the course materials,
- downloading notes and slides,
- online class and lab interactions,
- uploading answers to assignments.

CGI Programming

web interfaces

Course Administration:

- browsing of the course catalog,
- answering queries about courses,
- course registration.

Online Courses:

- browsing through the course materials,
- downloading notes and slides,
- online class and lab interactions,
- uploading answers to assignments.

CGI Programming

web interfaces

Course Administration:

- browsing of the course catalog,
- answering queries about courses,
- course registration.

Online Courses:

- browsing through the course materials,
- downloading notes and slides,
- online class and lab interactions,
- uploading answers to assignments.

CGI Programming

web interfaces

Course Administration:

- browsing of the course catalog,
- answering queries about courses,
- course registration.

Online Courses:

- browsing through the course materials,
- downloading notes and slides,
- online class and lab interactions,
- uploading answers to assignments.

CGI Programming

web interfaces

Course Administration:

- browsing of the course catalog,
- answering queries about courses,
- course registration.

Online Courses:

- browsing through the course materials,
- downloading notes and slides,
- online class and lab interactions,
- uploading answers to assignments.

CGI Programming

web interfaces

Course Administration:

- browsing of the course catalog,
- answering queries about courses,
- course registration.

Online Courses:

- browsing through the course materials,
- downloading notes and slides,
- online class and lab interactions,
- uploading answers to assignments.

Information Management

with databases

Database to administer courses has three tables:

1. students: information about students,
2. courses: prerequisites, description, ...,
3. enrollment: links students with courses.

Every course has its own database, for

- detailed syllabus,
- assignments, notes and slides,
- administration of grades.

Information Management

with databases

Database to administer courses has three tables:

1. students: information about students,
2. courses: prerequisites, description, ...,
3. enrollment: links students with courses.

Every course has its own database, for

- detailed syllabus,
- assignments, notes and slides,
- administration of grades.

Information Management

with databases

Database to administer courses has three tables:

1. students: information about students,
2. courses: prerequisites, description, ...,
3. enrollment: links students with courses.

Every course has its own database, for

- detailed syllabus,
- assignments, notes and slides,
- administration of grades.

Information Management

with databases

Database to administer courses has three tables:

1. students: information about students,
2. courses: prerequisites, description, ...,
3. enrollment: links students with courses.

Every course has its own database, for

- detailed syllabus,
- assignments, notes and slides,
- administration of grades.

Information Management

with databases

Database to administer courses has three tables:

1. students: information about students,
2. courses: prerequisites, description, ...,
3. enrollment: links students with courses.

Every course has its own database, for

- detailed syllabus,
- assignments, notes and slides,
- administration of grades.

Information Management

with databases

Database to administer courses has three tables:

1. students: information about students,
2. courses: prerequisites, description, ...,
3. enrollment: links students with courses.

Every course has its own database, for

- detailed syllabus,
- assignments, notes and slides,
- administration of grades.

Multiple Servers

to handle the load

Multiple computers to

- handle online registration,
- manage running of courses,
- backup essential data.

We expect our servers to be multifunctional:

- peak periods for registration,
- prime time for online courses.

Multiple Servers

to handle the load

Multiple computers to

- handle online registration,
- manage running of courses,
- backup essential data.

We expect our servers to be multifunctional:

- peak periods for registration,
- prime time for online courses.

Multiple Servers

to handle the load

Multiple computers to

- handle online registration,
- manage running of courses,
- backup essential data.

We expect our servers to be multifunctional:

- peak periods for registration,
- prime time for online courses.

Multiple Servers

to handle the load

Multiple computers to

- handle online registration,
- manage running of courses,
- backup essential data.

We expect our servers to be multifunctional:

- peak periods for registration,
- prime time for online courses.

Multiple Servers

to handle the load

Multiple computers to

- handle online registration,
- manage running of courses,
- backup essential data.

We expect our servers to be multifunctional:

- peak periods for registration,
- prime time for online courses.

Multithreaded Servers

to handle the load

All servers are multithreaded:

1. handle indefinite number of requests,
2. for an indefinite time.

Distributed computing over multiple computers:
→ load balancing and rescheduling of requests.

Multithreaded Servers

to handle the load

All servers are multithreaded:

1. handle indefinite number of requests,
2. for an indefinite time.

Distributed computing over multiple computers:
→ load balancing and rescheduling of requests.

Multithreaded Servers

to handle the load

All servers are multithreaded:

1. handle indefinite number of requests,
2. for an indefinite time.

Distributed computing over multiple computers:
→ load balancing and rescheduling of requests.

an application

A Virtual University

- description of an application
- the four pillars in the application

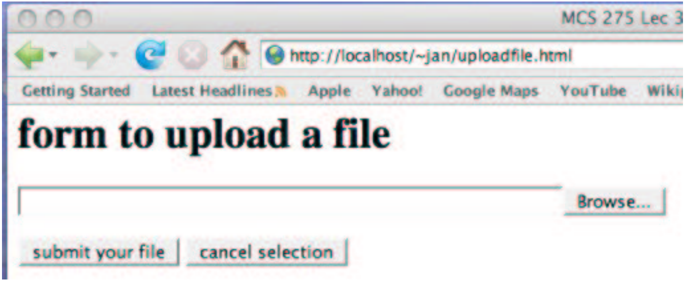
Uploading Files

- form of the HTML code to upload
- processing uploaded file with CGI script

The CGIHTTPServer Module

- testing HTML forms offline

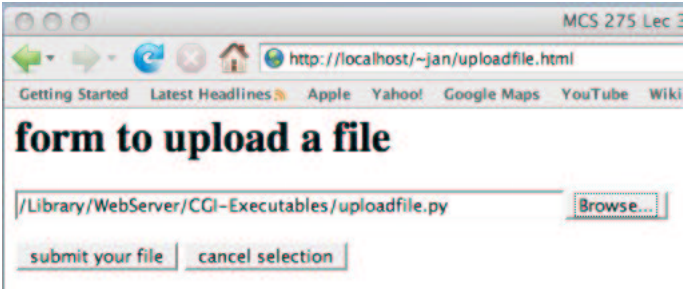
Simple Form to upload Files



http://localhost/~jan/uploadfile.html

form to upload a file

After Browsing the Disk

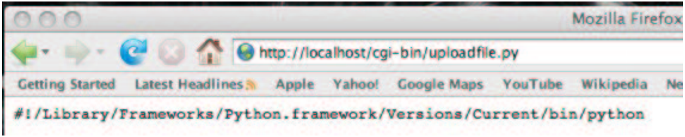


http://localhost/~jan/uploadfile.html

form to upload a file

/Library/WebServer/CGI-Executables/uploadfile.py

Printing the First Line



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

Uploading Files

via an HTML form

Answers to assignments will be uploaded.

We must specify the encoding of the form adding

```
enctype = "multitype/form-data"
```

as one of the tags.

We then use an input element of type `file`.
For example:

```
<input type="file" name="upfile" size="50">
```

Uploading Files

via an HTML form

Answers to assignments will be uploaded.

We must specify the encoding of the form adding

```
enctype = "multitype/form-data"
```

as one of the tags.

We then use an input element of type `file`.
For example:

```
<input type="file" name="upfile" size="50">
```

Uploading Files

via an HTML form

Answers to assignments will be uploaded.

We must specify the encoding of the form adding

```
enctype = "multitype/form-data"
```

as one of the tags.

We then use an input element of type `file`.
For example:

```
<input type="file" name="upfile" size="50">
```

HTML code

in the file `uploadfile.html`

```
<html>
<head>
<title> MCS 275 Lec 33: uploading a file </title>
</head>

<body>

<h1> form to upload a file </h1>

<form method="post"
      action="http://localhost/cgi-bin/uploadfile.py"
      enctype="multipart/form-data">

<input type="file" name="upfile" size = "50">

<p> <input type="submit" value="submit your file">
    <input type="reset" value="cancel selection"> </p>
</form>
</body>
</html>
```

A Virtual University

description of an application

the four pillars in the application

Uploading Files

form of the HTML code to upload

processing uploaded file with CGI script

The CGIHTTPServer Module

testing HTML forms offline

HTML code

in the file `uploadfile.html`

```
<html>
<head>
<title> MCS 275 Lec 33: uploading a file </title>
</head>

<body>

<h1> form to upload a file </h1>

<form method="post"
      action="http://localhost/cgi-bin/uploadfile.py"
      enctype="multipart/form-data">

<input type="file" name="upfile" size = "50">

<p> <input type="submit" value="submit your file">
    <input type="reset" value="cancel selection"> </p>
</form>
</body>
</html>
```

HTML code

in the file `uploadfile.html`

```
<html>
<head>
<title> MCS 275 Lec 33: uploading a file </title>
</head>

<body>

<h1> form to upload a file </h1>

<form method="post"
      action="http://localhost/cgi-bin/uploadfile.py"
      enctype="multipart/form-data">

<input type="file" name="upfile" size = "50">

<p> <input type="submit" value="submit your file">
    <input type="reset" value="cancel selection"> </p>
</form>
</body>
</html>
```

HTML code

in the file `uploadfile.html`

```
<html>
<head>
<title> MCS 275 Lec 33: uploading a file </title>
</head>

<body>

<h1> form to upload a file </h1>

<form method="post"
      action="http://localhost/cgi-bin/uploadfile.py"
      enctype="multipart/form-data">

<input type="file" name="upfile" size = "50">

<p> <input type="submit" value="submit your file">
    <input type="reset" value="cancel selection"> </p>
</form>
</body>
</html>
```

an application

A Virtual University

- description of an application
- the four pillars in the application

Uploading Files

- form of the HTML code to upload
- processing uploaded file with CGI script

The CGIHTTPServer Module

- testing HTML forms offline

Processing the Uploaded File

name field of the input element has value `'upfile'`.

In CGI script:

```
form = cgi.FieldStorage
```

Use the key name through its value as defined in the form to access the file:

```
uploaded = form['upfile']
```

Use the file attribute of `uploaded` for reading:

```
line = uploaded.file.readline()
```

Processing the Uploaded File

name field of the input element has value ’upfile’.

In CGI script:

```
form = cgi.FieldStorage
```

Use the key name through its value as defined in the form to access the file:

```
uploaded = form[’upfile’]
```

Use the file attribute of uploaded for reading:

```
line = uploaded.file.readline()
```

Processing the Uploaded File

name field of the input element has value `'upfile'`.

In CGI script:

```
form = cgi.FieldStorage
```

Use the key name through its value as defined in the form to access the file:

```
uploaded = form['upfile']
```

Use the file attribute of `uploaded` for reading:

```
line = uploaded.file.readline()
```

Processing the Uploaded File

name field of the input element has value `'upfile'`.

In CGI script:

```
form = cgi.FieldStorage
```

Use the key name through its value as defined in the form to access the file:

```
uploaded = form['upfile']
```

Use the file attribute of `uploaded` for reading:

```
line = uploaded.file.readline()
```

Script to see Uploaded File

in the file uploadfile.py

```
#!/Library/Frameworks/../bin/python
# L-33 MCS 275 Mon 7 Apr 2008 : uploadfile.py

# This CGI script takes the input of the form
# uploadfile.html and writes the first line of
# the file in plaintext on the web page.

import cgi
form = cgi.FieldStorage()

print "Content-Type: text/plain\n"

uploaded = form[’upfile’]

line = uploaded.file.readline()
print line
```

Script to see Uploaded File

in the file `uploadfile.py`

```
#!/Library/Frameworks/../bin/python
# L-33 MCS 275 Mon 7 Apr 2008 : uploadfile.py

# This CGI script takes the input of the form
# uploadfile.html and writes the first line of
# the file in plaintext on the web page.

import cgi
form = cgi.FieldStorage()

print "Content-Type: text/plain\n"

uploaded = form[’upfile’]

line = uploaded.file.readline()
print line
```

Script to see Uploaded File

in the file `uploadfile.py`

```
#!/Library/Frameworks/../bin/python
# L-33 MCS 275 Mon 7 Apr 2008 : uploadfile.py

# This CGI script takes the input of the form
# uploadfile.html and writes the first line of
# the file in plaintext on the web page.

import cgi
form = cgi.FieldStorage()

print "Content-Type: text/plain\n"

uploaded = form[’upfile’]

line = uploaded.file.readline()
print line
```

Script to see Uploaded File

in the file `uploadfile.py`

```
#!/Library/Frameworks/../bin/python
# L-33 MCS 275 Mon 7 Apr 2008 : uploadfile.py

# This CGI script takes the input of the form
# uploadfile.html and writes the first line of
# the file in plaintext on the web page.

import cgi
form = cgi.FieldStorage()

print "Content-Type: text/plain\n"

uploaded = form[’upfile’]

line = uploaded.file.readline()
print line
```

an application

A Virtual University

description of an application

the four pillars in the application

Uploading Files

form of the HTML code to upload

processing uploaded file with CGI script

The CGIHTTPServer Module

testing HTML forms offline

The CGIHTTPServer Module

```
>>> import CGIHTTPServer
>>> help(CGIHTTPServer)
```

security warning:

don’t use this code unless you are inside a firewall!

Good for testing forms offline:

- no need to install Apache,
- no root administrator permissions required.

The CGIHTTPServer Module

```
>>> import CGIHTTPServer
>>> help(CGIHTTPServer)
```

security warning:
don’t use this code unless you are inside a firewall!

Good for testing forms offline:

- no need to install Apache,
- no root administrator permissions required.

The CGIHTTPServer Module

```
>>> import CGIHTTPServer
>>> help(CGIHTTPServer)
```

security warning:
don’t use this code unless you are inside a firewall!

Good for testing forms offline:

- no need to install Apache,
- no root administrator permissions required.

A Virtual University
description of an application
the four pillars in the application
Uploading Files
form of the HTML code to upload
processing uploaded file with CGI script
The CGIHTTPServer Module
testing HTML forms offline

The CGIHTTPServer Module

```
>>> import CGIHTTPServer
>>> help(CGIHTTPServer)
```

security warning:
don’t use this code unless you are inside a firewall!

Good for testing forms offline:

- no need to install Apache,
- no root administrator permissions required.

The CGIHTTPServer Module

```
>>> import CGIHTTPServer
>>> help(CGIHTTPServer)
```

security warning:
don’t use this code unless you are inside a firewall!

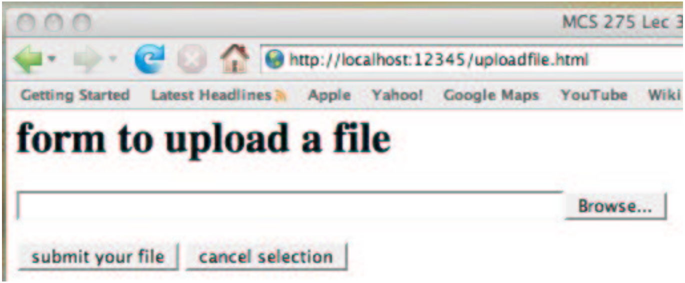
Good for testing forms offline:

- no need to install Apache,
- no root administrator permissions required.

Web Client connects to Port 12345

Starting `simplecgiserver.py` at command prompt:

```
$ python simplecgiserver.py
press ctrl c to stop server
localhost - - [07/Apr/2008 08:55:37] "GET / HTTP/1.1" 200
localhost - - [07/Apr/2008 08:56:49] "GET /uploadfile.html
```



http://localhost:12345/uploadfile.html

form to upload a file

A Simple CGI Server

test your forms off line

```
# L-33 MCS 275 Mon 7 Apr 2008 : simplecgiserver.py

# After
# $ python simplecgiserver.py
# launch a browser at
# http://localhost:12345/uploadfile.html

from BaseHTTPServer import HTTPServer
from CGIHTTPServer import CGIHTTPRequestHandler

s = HTTPServer((’’,12345),CGIHTTPRequestHandler)

try:
   print ’press ctrl c to stop server’
   s.serve_forever()
except KeyboardInterrupt:
   print ’ctrl c pressed, stopping server’
   s.socket.close()
```

A Simple CGI Server

test your forms off line

```
# L-33 MCS 275 Mon 7 Apr 2008 : simplecgiserver.py

# After
# $ python simplecgiserver.py
# launch a browser at
# http://localhost:12345/uploadfile.html

from BaseHTTPServer import HTTPServer
from CGIHTTPServer import CGIHTTPRequestHandler

s = HTTPServer((’’,12345),CGIHTTPRequestHandler)

try:
   print ’press ctrl c to stop server’
   s.serve_forever()
except KeyboardInterrupt:
   print ’ctrl c pressed, stopping server’
   s.socket.close()
```

A Simple CGI Server

test your forms off line

```
# L-33 MCS 275 Mon 7 Apr 2008 : simplecgiserver.py

# After
# $ python simplecgiserver.py
# launch a browser at
# http://localhost:12345/uploadfile.html

from BaseHTTPServer import HTTPServer
from CGIHTTPServer import CGIHTTPRequestHandler

s = HTTPServer((’’,12345),CGIHTTPRequestHandler)

try:
   print ’press ctrl c to stop server’
   s.serve_forever()
except KeyboardInterrupt:
   print ’ctrl c pressed, stopping server’
   s.socket.close()
```

A Simple CGI Server

test your forms off line

```
# L-33 MCS 275 Mon 7 Apr 2008 : simplecgiserver.py

# After
# $ python simplecgiserver.py
# launch a browser at
# http://localhost:12345/uploadfile.html

from BaseHTTPServer import HTTPServer
from CGIHTTPServer import CGIHTTPRequestHandler

s = HTTPServer(('',12345),CGIHTTPRequestHandler)

try:
   print 'press ctrl c to stop server'
   s.serve_forever()
except KeyboardInterrupt:
   print 'ctrl c pressed, stopping server'
   s.socket.close()
```

A Simple CGI Server

test your forms off line

```
# L-33 MCS 275 Mon 7 Apr 2008 : simplecgiserver.py

# After
# $ python simplecgiserver.py
# launch a browser at
# http://localhost:12345/uploadfile.html

from BaseHTTPServer import HTTPServer
from CGIHTTPServer import CGIHTTPRequestHandler

s = HTTPServer((’’,12345),CGIHTTPRequestHandler)

try:
   print ’press ctrl c to stop server’
   s.serve_forever()
except KeyboardInterrupt:
   print ’ctrl c pressed, stopping server’
   s.socket.close()
```

Summary + Assignments

We ended more of chapter 14 in *Making Use of Python*.

Assignments:

1. Use one script to upload files instead of using separate HTML code. Combine the HTML code and the code to process the uploaded file into functions `PrintForm()` and `ProcessFile()`.
2. Extend the `uploadfile.py` script as follows. In case the file uploaded is a Python script, it counts the number of functions (just scan for `def`) and prints this count to the web page.