Outline

1  Web Servers
   Apache and LAMP
   Python glues CGI, MySQLdb, and Sockets

2  Interactive Web Pages
   MySQLdb in client/server computing
   displaying all records in HTML table
   retrieving and packing records
   the client displays HTML table
   processing forms with CGI scripts

3  The Meaning of Python
   a dynamic language
   rapid application development

MCS 260 Lecture 40
Introduction to Computer Science
Jan Verschelde, 26 November 2008
web servers

interactive web pages

1 Web Servers

Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

2 Interactive Web Pages

MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

3 The Meaning of Python

a dynamic language
rapid application development
The combination of web server, scripting language, and database is often referred to as the LAMP stack.

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.

Python provides the glue to connect a database to a web client using sockets.
The combination of web server, scripting language, and database is often referred to as the LAMP stack.

**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.

Python provides the glue to connect a database to a web client using sockets.
The combination of web server, scripting language, and database is often referred to as the LAMP stack.

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.

Python provides the glue to connect a database to a web client using sockets.
The combination of web server, scripting language, and database is often referred to as the LAMP stack.

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.

Python provides the glue to connect a database to a web client using sockets.
the LAMP stack
Python glues

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

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.

Python provides the glue to connect a database to a web client using sockets.
The combination of web server, scripting language, and database is often referred to as the LAMP stack.

**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.

Python provides the glue to connect a database to a web client using sockets.
The combination of web server, scripting language, and database is often referred to as the LAMP stack.

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.

Python provides the glue to connect a database to a web client using sockets.
web servers
interactive web pages

1. Web Servers
   Apache and LAMP
   Python glues CGI, MySQLdb, and Sockets

2. Interactive Web Pages
   MySQLdb in client/server computing
   displaying all records in HTML table
   retrieving and packing records
   the client displays HTML table
   processing forms with CGI scripts

3. The Meaning of Python
   a dynamic language
   rapid application development
Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles with scripts table`.

Ingredients:

1. MySQL database `OurPyFiles` stores in its `scripts table` lecture number, date, and name of script.
2. The server listens to two connections of clients, providing access to the database.
3. Two client scripts run in web browser:
   1. first client allows user to select sort order,
   2. second client displays sorted records.
CGI, MySQLdb, and Sockets
glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using MySQLdb
2. client is a CGI script: web interface

Example database: OurPyFiles with scripts table.

Ingredients:

1. MySQL database OurPyFiles stores in its scripts table lecture number, date, and name of script.
2. The server listens to two connections of clients, providing access to the database.
3. Two client scripts run in web browser:
   1. first client allows user to select sort order,
   2. second client displays sorted records.
Goal: build web interface to MySQL database.

Components:

1. server is Python script using MySQLdb
2. client is a CGI script: web interface

Example database: OurPyFiles with scripts table.

Ingredients:

1. MySQL database OurPyFiles stores in its scripts table lecture number, date, and name of script.
2. The server listens to two connections of clients, providing access to the database.
3. Two client scripts run in web browser:
   1. first client allows user to select sort order,
   2. second client displays sorted records.
CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using MySQLdb
2. client is a CGI script: web interface

Example database: OurPyFiles with scripts table.

Ingredients:

1. MySQL database OurPyFiles stores in its scripts table lecture number, date, and name of script.
2. The server listens to two connections of clients, providing access to the database.
3. Two client scripts run in web browser:
   1. first client allows user to select sort order,
   2. second client displays sorted records.
CGI, MySQLdb, and Sockets

Goal: build web interface to MySQL database.

Components:

1. server is Python script using **MySQLdb**
2. client is a CGI script: web interface

Example database: OurPyFiles with scripts table.

Ingredients:

1. **MySQL database** OurPyFiles stores in its scripts table lecture number, date, and name of script.

2. The server listens to two connections of clients, providing access to the database.

3. Two client scripts run in web browser:
   1. first client allows user to select sort order,
   2. second client displays sorted records.
Goal: build web interface to MySQL database.

Components:

1. server is Python script using MySQLdb
2. client is a CGI script: web interface

Example database: OurPyFiles with scripts table.

Ingredients:

1. MySQL database OurPyFiles stores in its scripts table lecture number, date, and name of script.
2. The server listens to two connections of clients, providing access to the database.
3. Two client scripts run in web browser:
   1. first client allows user to select sort order,
   2. second client displays sorted records.
CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

① server is Python script using **MySQLdb**
② client is a CGI script: web interface

Example database: **OurPyFiles** with **scripts** table.

Ingredients:

① **MySQL database** **OurPyFiles** stores in its **scripts** table lecture number, date, and name of script.
② The server listens to two connections of clients, providing access to the database.
③ Two client scripts run in web browser:
   ① first client allows user to select sort order,
   ② second client displays sorted records.
CGI, MySQLdb, and Sockets

Glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using MySQLdb
2. client is a CGI script: web interface

Example database: OurPyFiles with scripts table.

Ingredients:

1. MySQL database OurPyFiles stores in its scripts table lecture number, date, and name of script.
2. The server listens to two connections of clients, providing access to the database.
3. Two client scripts run in web browser:
   1. first client allows user to select sort order,
   2. second client displays sorted records.
The First Client

**The Python code `scripts_sort.py`:**

1. runs in a browser
2. connects to the database
3. prints an HTML form for sort order
4. calls the second script
The First Client

The Python code `scripts_sort.py`:

1. runs in a browser
2. connects to the database
3. prints an HTML form for sort order
4. calls the second script
The Second Client

Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python
a dynamic language
rapid application development

Number of scripts: 77
0  L-1  2008-01-14  enumdivs.py
1  L-1  2008-01-14  facnum.py
2  L-2  2008-01-14  facnums.py
3  L-2  2008-01-14  setup.py
4  L-3  2008-01-18  sieve.py
5  L-3  2008-01-18  saddle.py
6  L-3  2008-01-18  findmin.py
7  L-4  2008-01-23  classhistogram.py
8  L-4  2008-01-23  usehistogram
9  L-4  2008-01-23  classmatrix.py
10 L-4  2008-01-23  dictarg.py
11 L-5  2008-01-25  billiards_basic.py
12 L-5  2008-01-25  billiard_paths.py
13 L-5  2008-01-25  billiards.py
14 L-6  2008-01-28  palindromes.py
15 L-6  2008-01-28  factorial.py
16 L-7  2008-01-30  sierpinski.py
17 L-7  2008-01-30  koch.py
18 L-7  2008-01-30  flake.py
19 L-7  2008-01-30  multigon.py
20 L-7  2008-01-30  cantor.py
21 L-8  2008-02-01  fibonacci
22 L-8  2008-02-01  Towers of Hanoi
23 L-9  2008-02-04  enumwstop.py
24 L-9  2008-02-04  knapsack.py
1. Web Servers
   Apache and LAMP
   Python glues CGI, MySQLdb, and Sockets

2. Interactive Web Pages
   MySQLdb in client/server computing
   displaying all records in HTML table
   retrieving and packing records
   the client displays HTML table
   processing forms with CGI scripts

3. The Meaning of Python
   a dynamic language
   rapid application development
Counting Number of Records
using of MySQLdb

scripts_count.py prints number of records in table scripts of MySQL database OurPyFiles.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```python
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```
Counting Number of Records
using of MySQLdb

scripts_count.py prints number of records in table
scripts of MySQL database OurPyFiles.

Requirements for a successful run:

① MySQL must be started: sudo mysqld_safe

② run as sudo python scripts_count.py

```python
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```
Counting Number of Records using of MySQLdb

scripts_count.py prints number of records in table scripts of MySQL database OurPyFiles.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```python
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```
Counting Number of Records
using of MySQLdb

scripts_count.py prints number of records in table
scripts of MySQL database OurPyFiles.

Requirements for a successful run:

1. MySQL must be started: sudo mysqld_safe
2. run as sudo python scripts_count.py

import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()

q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
Counting Number of Records
using of MySQLdb

scripts_count.py prints number of records in table scripts of MySQL database OurPyFiles.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```python
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```
Counting Number of Records
using of MySQLdb

scripts_count.py prints number of records in table
scripts of MySQL database OurPyFiles.

Requirements for a successful run:

1. **MySQL must be started:** `sudo mysqld_safe`
2. **run as** `sudo python scripts_count.py`

```python
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```
import MySQLdb
from socket import *

def connect():
    
    Returns client and server socket to communicate with one client.

def count():
    
    Returns the number of scripts.

def main():
    
    Accepts connection and sends #scripts.
import MySQLdb
from socket import *
def connect():
    """
    Returns client and server socket to communicate with one client.
    """

def count():
    """
    Returns the number of scripts.
    """

def main():
    """
    Accepts connection and sends #scripts.
    """
import MySQLdb
from socket import *
def connect():
    """
    Returns client and server socket to communicate with one client.
    """
def count():
    """
    Returns the number of scripts.
    """
def main():
    """
    Accepts connection and sends #scripts.
    """
Functions of the Server
structure of `scripts_server.py`

```python
import MySQLdb
from socket import *

def connect():
    """
    Returns client and server socket
to communicate with one client.
    """

def count():
    """
    Returns the number of scripts.
    """

def main():
    """
    Accepts connection and sends #scripts.
    """
```
Defining Network Connections

```python
hostname = ''  # use any address
number = 11267  # number for the port
buffer = 80  # size of the buffer

def connect():
    """
    Returns client and server socket
to communicate with one client.
    """
    server_address = (hostname, number)
    server = socket(AF_INET, SOCK_STREAM)
    server.bind(server_address)
    server.listen(1)
    print 'server waits for connection'
    client, client_address = server.accept()
    print 'server accepted connection from ', client_address
    return client, server
```
Defining Network Connections

```python
hostname = ''  # use any address
number = 11267  # number for the port
buffer = 80    # size of the buffer

def connect():
    """"""
    Returns client and server socket to communicate with one client.
    """
    server_address = (hostname, number)
    server = socket(AF_INET, SOCK_STREAM)
    server.bind((server_address)
    server.listen(1)
    print 'server waits for connection'
    client, client_address = server.accept()
    print 'server accepted connection from ', client_address
    return client, server
```
Defining Network Connections

hostname = ''   # use any address
number = 11267  # number for the port
buffer = 80     # size of the buffer

def connect():
    
    Returns client and server socket
to communicate with one client.
    
server_address = (hostname, number)
server = socket(AF_INET, SOCK_STREAM)
server.bind(server_address)
server.listen(1)
print 'server waits for connection'
client, client_address = server.accept()
print 'server accepted connection from ',
client_address
return client, server
Counting the Number of Records

def count():
    ""
    Returns the number of scripts.
    ""
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    q = 'select count(*) from scripts'
    cr.execute(q)
    r = cr.fetchone()
    n = int(r[0])
    return n
def main():
    """
    Accepts connection and sends #scripts.
    """
    client, server = connect()
    print 'server connects to database'
    nb = count()
    print 'server sends #scripts to client'
    data = str(nb)
    client.send(data)
    print 'count sent, closing off'
    server.close()
The Function `main()`

```python
def main():
    
    """
    Accepts connection and sends #scripts.
    """
    
    client, server = connect()
    print 'server connects to database'
    nb = count()
    print 'server sends #scripts to client'
    data = str(nb)
    client.send(data)
    print 'count sent, closing off'
    server.close()
```
def main():
    """
    Accepts connection and sends #scripts.
    """
    client, server = connect()
    print 'server connects to database'
    nb = count()
    print 'server sends #scripts to client'
    data = str(nb)
    client.send(data)
    print 'count sent, closing off'
    server.close()
First Version of the Client

in the file scripts_client.py

from socket import *

hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer

server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = client.recv(buffer)
print 'client received "' + data + '"

client.close()
The Client is Web Interface

CGI script `scripts_web.py`

```python
#!/Library/Frameworks/.../bin/python
print "Content-Type: text/plain\n\n"

from socket import *
hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer
server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)
print 'client is connected'
data = client.recv(buffer)
print 'Number of scripts : ' + data
client.close()
```
The Client is Web Interface

CGI script `scripts_web.py`

```python
#!/Library/Frameworks/.../bin/python
print "Content-Type: text/plain\n\n"

from socket import *
hostname = 'localhost' # on same host
number = 11267 # same port number
buffer = 80 # size of the buffer
server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)
print 'client is connected'
data = client.recv(buffer)
print 'Number of scripts : ' + data
client.close()
```
The Client is Web Interface

CGI script scripts_web.py

```python
#!/Library/Frameworks/.../bin/python
print "Content-Type: text/plain\n\n"
from socket import *
hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer
server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)
print 'client is connected'
data = client.recv(buffer)
print 'Number of scripts : ' + data
client.close()
```
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python
a dynamic language
rapid application development
Displaying on Web Page

extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records and makes HTML table to display

Synchronization is very important: *for every send of the server, there must be a matching recv by the client!"
Displaying on Web Page

extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records and makes HTML table to display

Synchronization is very important: for every send of the server, there must be a matching recv by the client!
Displaying on Web Page
extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records and makes HTML table to display

Synchronization is very important: \textit{for every send of the server, there must be a matching recv by the client!}
Displaying on Web Page
extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records and makes HTML table to display

Synchronization is very important: for every send of the server, there must be a matching recv by the client!
Displaying on Web Page
extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records and makes HTML table to display

Synchronization is very important: for every send of the server, there must be a matching recv by the client!
Functions of the Server

in file `scripts_servdb.py`

```python
def ConnectClient():
    
    Returns client and server socket.

def CountRecords(c):
    
    Returns the #scripts, given cursor c.

def RetrieveRecords(c):
    
    Given cursor c, returns all records.

def PackTuple(t):
    
    Packs data tuple as string.
```
The Function main()

in the file scripts_servvdb.py

def main():
    
    """
    Accepts connection and sends records.
    """

db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
nb = CountRecords(cr)
client, server = ConnectClient()
client.send(str(nb))
R = RetrieveRecords(cr)
for i in range(0,len(R)):
    client.send(PackTuple(R[i]))
server.close()
def main():
    
    """
    Accepts connection and sends records.
    """

db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()

nb = CountRecords(cr)

client, server = ConnectClient()

client.send(str(nb))

R = RetrieveRecords(cr)

for i in range(0, len(R)):
    client.send(PackTuple(R[i]))

server.close()
def main():
    """
    Accepts connection and sends records.
    """
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    client, server = ConnectClient()
    client.send(str(nb))
    R = RetrieveRecords(cr)
    for i in range(0,len(R)):
        client.send(PackTuple(R[i]))
    server.close()
web servers
interactive web pages

1. Web Servers
   Apache and LAMP
   Python glues CGI, MySQLdb, and Sockets

2. Interactive Web Pages
   MySQLdb in client/server computing
   displaying all records in HTML table
   retrieving and packing records
   the client displays HTML table
   processing forms with CGI scripts

3. The Meaning of Python
   a dynamic language
   rapid application development
Retrieving and Packing Tuples

def RetrieveRecords(c):
    ""
    Given cursor c, returns all records.
    ""
    q = 'select * from scripts'
c.execute(q)
return c.fetchall()

def PackTuple(t):
    ""
    Packs the tuple as string with items separated by colons. Notice padding!
    ""
    s = t[0] + '-' + str(int(t[1])) + ':'
s = s + str(t[2]) + ':' + t[3] + ':'
r = s + (buffer - len(s))*'
return r
Retrieving and Packing Tuples

def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
c.execute(q)
return c.fetchall()

def PackTuple(t):
    """
Packs the tuple as string with items separated by colons. Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
s = s + str(t[2]) + ':' + t[3] + ':'
r = s + (buffer - len(s))*' '
return r
Retrieving and Packing Tuples

```python
def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
c.execute(q)
return c.fetchall()

def PackTuple(t):
    """
Packs the tuple as string with items separated by colons. Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
s = s + str(t[2]) + ':' + t[3] + ':'
r = s + (buffer - len(s))*'
return r
```
Retrieving and Packing Tuples

def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
    c.execute(q)
    return c.fetchall()

def PackTuple(t):
    """
    Packs the tuple as string with items
    separated by colons. Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
    s = s + str(t[2]) + ':' + t[3] + ':'
    r = s + (buffer - len(s))*' ' 
    return r
Retrieving and Packing Tuples

def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
    c.execute(q)
    return c.fetchall()

def PackTuple(t):
    """
    Packs the tuple as string with items separated by colons. Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
    s = s + str(t[2]) + ':' + t[3] + ':'
    r = s + (buffer - len(s))*'
    return r
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python
a dynamic language
rapid application development
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive
Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python
a dynamic language
rapid application development

Code for the Client
start of the file `scripts_showall.py`

```python
#!/Library/Frameworks/.../bin/python

from socket import *
hostname = 'localhost' # on same host
number = 11267 # same port number
buffer = 80 # size of the buffer

def PrintHeader(title):
    ""
    writes title and header of page
    ""
    print """"Content-type: text/html
<html>
<head>
<title>%s</title>
</head>
<body>"" % title
```
Code for the Client

start of the file scripts_showall.py

#!/Library/Frameworks/.../bin/python

from socket import *
hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer

def PrintHeader(title):
    """
    writes title and header of page
    """
    print """"Content-type: text/html
<html>
<head>
<title>%s</title>
</head>
<body>""" % title
The Function main() in the Client
in the file scripts_showall.py

```python
def main():
    
    """
    Connects and prints data of server.
    """

    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    RetrieveTable(client,n)
    client.close()
```
The Function main() in the Client

in the file scripts_showall.py

```python
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    RetrieveTable(client,n)
    client.close()
```
def main():
    
    """
    Connects and prints data of server.
    """

    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    RetrieveTable(client,n)
    client.close()
Retrieving and Displaying
in the file scripts_showall.py

```python
def RetrieveTable(s,n):
    """
    Retrieves table of n records, using socket s to communicate.
    """
    print "<table>"
    for i in range(0,n):
        data = s.recv(buffer)
        d = data.split('::')
        print "<tr>"
        print "<td>%d</td>" % i
        print "<td>%s</td>" % d[0]
        print "<td>%s</td>" % d[1]
        print "<td>%s</td>" % d[2]
        print "</tr>"
    print "</table>"
```
def RetrieveTable(s, n):
    ""
    Retrieves table of n records, using socket s to communicate.
    ""
    print "<table>
    for i in range(0, n):
        data = s.recv(buffer)
        d = data.split(':')
        print "<tr>"
        print "<td>%d</td>" % i
        print "<td>%s</td>" % d[0]
        print "<td>%s</td>" % d[1]
        print "<td>%s</td>" % d[2]
        print "</tr>"
    print "</table>"
def RetrieveTable(s,n):
    """
    Retrieves table of n records, using socket s to communicate.
    """
    print "<table>"
    for i in range(0,n):
        data = s.recv(buffer)
        d = data.split(':')
        print "<tr>"
        print "<td>%d</td>" % i
        print "<td>%s</td>" % d[0]
        print "<td>%s</td>" % d[1]
        print "<td>%s</td>" % d[2]
        print "</tr>"
    print "</table>"
Radio Buttons for Sort Order

This HTML form is stored in
- users public_html directory on Unix
- users Sites directory on Mac OS X
- in htdocs of Apache directory on Windows
Radio Buttons for Sort Order

This HTML form is stored in

- **`users public_html`** directory on Unix
- **`users Sites`** directory on Mac OS X
- **`htdocs`** of Apache directory on Windows
CGI Script to Confirm Choice

This CGI script is stored in

- /var/www/cgi-bin on Unix
- /Library/WebServer/CGI-Executables on Mac OS X
- cgi-bin on Apache directory on Windows
CGI Script to Confirm Choice

This CGI script is stored in

- `/var/www/cgi-bin` on Unix
- `/Library/WebServer/CGI-Executables` on Mac OS X
- `cgi-bin` on Apache directory on Windows
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python
a dynamic language
rapid application development

---

**HTML Code**

body in file sort_order.html

```
<h1> determine sort order </h1>
<form action="http://localhost/cgi-bin/sort_order.py">
<p>
sort by
<input type="radio" name="sort" value = 0 checked> type
<input type="radio" name="sort" value = 1> date
<input type="radio" name="sort" value = 2> name
<br>
order is
<input type="radio" name="order" value = True checked> ascending
<input type="radio" name="order" value = False> descending
</p>
<p> <input type="submit"> </p>
```
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets
Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts
The Meaning of Python
a dynamic language
rapid application development

HTML Code
body in file sort_order.html

<h1> determine sort order </h1>
<form action="http://localhost/cgi-bin/sort_order.py">
<p>sort by</p>
<input type="radio" name="sort" value=0 checked> type
<input type="radio" name="sort" value=1> date
<input type="radio" name="sort" value=2> name
<br>
order is
<input type="radio" name="order" value=True checked> ascending
<input type="radio" name="order" value=False> descending
</p>
<p> <input type="submit"> </p>
<h1> determine sort order </h1>
<form action="http://localhost/cgi-bin/sort_order.py">
<p>
sort by
<input type="radio" name="sort" value="0" checked> type
<input type="radio" name="sort" value="1"> date
<input type="radio" name="sort" value="2"> name
<br>
order is
<input type="radio" name="order" value="True" checked> ascending
<input type="radio" name="order" value="False"> descending
</p>
<p> <input type="submit"> </p>
</form>
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python
a dynamic language
rapid application development

HTML Code

body in file sort_order.html

```html
<h1> determine sort order </h1>
<form action="http://localhost/cgi-bin/sort_order.py">
<p>
sort by <br>
<input type="radio" name="sort" value = 0 checked> type <br>
<input type="radio" name="sort" value = 1> date <br>
<input type="radio" name="sort" value = 2> name <br>
</p>
order is <br>
<input type="radio" name="order" value = True checked> ascending <br>
<input type="radio" name="order" value = False> descending <br>
</p>
<p> <input type="submit"> </p>
```
CGI Script

in file sort_order.py

#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
    s = 'sort by type and number'
elif sortby == '1':
    s = 'sort by date'
else:
    s = 'sort by name'
if eval(orderis):
    s = s + ' in ascending order'
else:
    s = s + ' in descending order'
print "Content-Type: text/plain"
print s
CGI Script
in file sort_order.py

#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
    s = 'sort by type and number'
elif sortby == '1':
    s = 'sort by date'
else:
    s = 'sort by name'
if eval(orderis):
    s = s + ' in ascending order'
else:
    s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python
a dynamic language
rapid application development

---

CGI Script
in file sort_order.py

```python
#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
    s = 'sort by type and number'
elif sortby == '1':
    s = 'sort by date'
else:
    s = 'sort by name'
if eval(orderis):
    s = s + ' in ascending order'
else:
    s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
```
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets
Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts
The Meaning of Python
a dynamic language
rapid application development

CGI Script
in file sort_order.py

#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
    s = 'sort by type and number'
elif sortby == '1':
    s = 'sort by date'
else:
    s = 'sort by name'
if eval(orderis):
    s = s + ' in ascending order'
else:
    s = s + ' in descending order'
print "Content-Type: text/plain\n"
p
print s
Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python
a dynamic language
rapid application development

CGI Script
in file sort_order.py

#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
    s = 'sort by type and number'
elif sortby == '1':
    s = 'sort by date'
else:
    s = 'sort by name'
if eval(orderis):
    s = s + ' in ascending order'
else:
    s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
Web Servers
 Apache and LAMP
 Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages
 MySQLdb in client/server computing
 displaying all records in HTML table
 retrieving and packing records
 the client displays HTML table
 processing forms with CGI scripts

The Meaning of Python
 a dynamic language
 rapid application development

CGI Script
 in file sort_order.py

#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()

sortby = form['sort'].value
orderis = form['order'].value

if sortby == '0':
    s = 'sort by type and number'
elif sortby == '1':
    s = 'sort by date'
else:
    s = 'sort by name'

if eval(orderis):
    s = s + ' in ascending order'
else:
    s = s + ' in descending order'

print "Content-Type: text/plain\n"
print s
Web Servers

Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

Interactive Web Pages

MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

The Meaning of Python

a dynamic language
rapid application development
Forms with CGI Scripts

Good for testing:

1. f.html has form, action refers to f.py
2. f.py defines CGI script, invoked by submit

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. f.html has form, action refers to f.py
2. f.py defines CGI script, invoked by submit

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. \texttt{f.html} has form, action refers to \texttt{f.py}
2. \texttt{f.py} defines CGI script, invoked by \texttt{submit}

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. First client displays number of records, prints the form for the sort order, and activates the second client
2. Second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. *f.html* has form, action refers to *f.py*
2. *f.py* defines CGI script, invoked by *submit*

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. First client displays number of records, prints the form for the sort order, and activates the second client.
2. Second client processes the form, sends sort order to server, and retrieves and displays sorted records.

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. \texttt{f.html} has form, action refers to \texttt{f.py}
2. \texttt{f.py} defines CGI script, invoked by \texttt{submit}

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. \texttt{f.html} has form, action refers to \texttt{f.py}
2. \texttt{f.py} defines CGI script, invoked by \texttt{submit}

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. \texttt{f.html} has form, action refers to \texttt{f.py}
2. \texttt{f.py} defines CGI script, invoked by \texttt{submit}

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
Forms with CGI Scripts

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. First client displays number of records, prints the form for the sort order, and activates the second client
2. Second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.
def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order \"' + sortorder + '\\"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order "' + sortorder + '"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...'
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
main() in Server

scripts_sortdb.py

def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order "' + sortorder + '"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...'
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
main() in Server

scripts_sortdb.py

def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order "' + sortorder + '"
    R = RetrieveRecords(cr, sortorder)
    print 'sending records ...'
    for i in range(0, len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order "' + sortorder + '"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('sorting all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print '<B>Number of scripts : %d</B>' % n
    PromptSortOrder()
    client.close()
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('sorting all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    PromptSortOrder()
    client.close()
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('sorting all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    PromptSortOrder()
    client.close()
main() in Second Client

in file scripts_sortall.py

def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print '<b>Number of scripts : %d</b>' % n
    SendSortOrder(client)
    RetrieveTable(client, n)
    client.close()
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<b>Number of scripts : %d</b>" % n
    SendSortOrder(client)
    RetrieveTable(client, n)
    client.close()
First Client prompts Sort Order

```python
def PromptSortOrder():
    """
    Display a form to ask user for field to sort on and the order.
    """
    print ""
    <form action="http://localhost/cgi-bin/scripts_sortall.py">
        ... rest of html code ...
    </form>
    <p>
        ... rest of html code ...
    """
```
def SendSortOrder(cs):
    ""
    Sends sort order to server using the client socket cs.
    ""
    form = cgi.FieldStorage()
    sortby = form['sort'].value
    if eval(form['order'].value):
        sortby = sortby + '+'
    else:
        sortby = sortby + '-'
    cs.send(sortby)
Second Client sends Sort Order

def SendSortOrder(cs):
    
    """
    Sends sort order to server using the client socket cs.
    """
    form = cgi.FieldStorage()
    sortby = form['sort'].value
    if eval(form['order'].value):
        sortby = sortby + '+'
    else:
        sortby = sortby + '-'
    cs.send(sortby)
Second Client sends Sort Order

def SendSortOrder(cs):
    
    """
    Sends sort order to server using the client socket cs.
    """

    form = cgi.FieldStorage()
    sortby = form['sort'].value
    if eval(form['order'].value):
        sortby = sortby + '+'
    else:
        sortby = sortby + '-'
    cs.send(sortby)
Query in Server Script

```python
def RetrieveRecords(c, sortorder):
    """
    Given cursor c, returns all records, taking sortorder into account.
    """
    q = 'select * from scripts'
    if sortorder[0] == '0':
        q = q + ' order by t,n'
    elif sortorder[0] == '1':
        q = q + ' order by d'
    else:
        q = q + ' order by f'
    if sortorder[1] == '+':
        q = q + ' asc'
    else:
        q = q + ' desc'
    c.execute(q)
    return c.fetchall()
```
def RetrieveRecords(c, sortorder):
    """
    Given cursor c, returns all records, taking sortorder into account.
    """
    q = 'select * from scripts'
    if sortorder[0] == '0':
        q = q + ' order by t,n'
    elif sortorder[0] == '1':
        q = q + ' order by d'
    else:
        q = q + ' order by f'
    if sortorder[1] == '+':
        q = q + ' asc'
    else:
        q = q + ' desc'
    c.execute(q)
    return c.fetchall()
def RetrieveRecords(c, sortorder):
    
    Given cursor c, returns all records, taking sortorder into account.

    q = 'select * from scripts'
    if sortorder[0] == '0':
        q = q + ' order by t,n'
    elif sortorder[0] == '1':
        q = q + ' order by d'
    else:
        q = q + ' order by f'
    if sortorder[1] == '+':
        q = q + ' asc'
    else:
        q = q + ' desc'
    c.execute(q)
    return c.fetchall()
web servers
interactive web pages

1 Web Servers
Apache and LAMP
Python glues CGI, MySQLdb, and Sockets

2 Interactive Web Pages
MySQLdb in client/server computing
displaying all records in HTML table
retrieving and packing records
the client displays HTML table
processing forms with CGI scripts

3 The Meaning of Python
a dynamic language
rapid application development
The Development Cycle
compilation versus interpretation

Traditional build cycle:

1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.

Python eliminates steps 4 and 5.

Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
The Development Cycle
compilation versus interpretation

Traditional build cycle:

1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.

Python eliminates steps 4 and 5.

Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
The Development Cycle
compilation versus interpretation

Traditional build cycle:
1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.
Python eliminates steps 4 and 5.
Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
The Development Cycle

compilation versus interpretation

Traditional build cycle:

1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.

Python eliminates steps 4 and 5.

Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
The Development Cycle
compilation versus interpretation

Traditional build cycle:
1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.

Python eliminates steps 4 and 5.

Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
The Development Cycle

compilation versus interpretation

Traditional build cycle:

1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.

Python eliminates steps 4 and 5.

Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
The Development Cycle
compilation versus interpretation

Traditional build cycle:

1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.

Python eliminates steps 4 and 5.

Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
The Development Cycle
compilation versus interpretation

Traditional build cycle:

1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.

Python eliminates steps 4 and 5.

Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
The Development Cycle
compilation versus interpretation

Traditional build cycle:

1. run the application
2. test the behavior of the code
3. stop the application
4. edit the program code
5. recompile the code
6. relink executable
7. goto step 1

This is the static language build cycle.

Python eliminates steps 4 and 5.

Recompilation and relinking is not a trivial task for systems with over 100,000 lines of code.
Python is a dynamic Language

some advertisements

Python may be a scripting language, but it scales well.

- “Python is executable pseudocode.”
  very readable,
  complexities involving memory addresses are hidden

- “Python is OOP done right.”
  compared to C++,
  the class mechanism in Python is much simpler

Python is a dynamic Language

some advertisements

Python may be a scripting language, but it scales well.

- “Python is executable pseudocode.”
  very readable,
  complexities involving memory addresses are hidden

- “Python is OOP done right.”
  compared to C++,
  the class mechanism in Python is much simpler

Python is a dynamic Language

some advertisements

Python may be a scripting language, but it scales well.

- “Python is executable pseudocode.” very readable, complexities involving memory addresses are hidden
- “Python is OOP done right.” compared to C++, the class mechanism in Python is much simpler

web servers

Interactive Web Pages

The Meaning of Python
Rapid Application Development
an answer to the software crisis

prototyping: all Python
hybrid: mixture of Python and C/C++
delivery: all C/C++

Consider the slider:

prototyping
hybrid
delivery

all Python
mixture
all C/C++

Identifying the bottlenecks in a working prototype leads to a gradual development of modules in C/C++. Python complements languages like C/C++ and Java. Cython (C-extensions for Python) is a language close to Python with support for C to generate efficient C code.
Rapid Application Development
an answer to the software crisis

prototyping: all Python
hybrid: mixture of Python and C/C++
delivery: all C/C++

Consider the slider:

prototyping hybrid delivery
all Python mixture all C/C++

Identifying the bottlenecks in a working prototype leads to a gradual development of modules in C/C++.

Python complements languages like C/C++ and Java. Cython (C-extensions for Python) is a language close to Python with support for C to generate efficient C code.
Rapid Application Development
an answer to the software crisis

prototyping: all Python
hybrid: mixture of Python and C/C++
delivery: all C/C++

Consider the slider:

prototyping all Python
hybrid mixture
delivery all C/C++

Identifying the bottlenecks in a working prototype leads to a gradual development of modules in C/C++.

Python complements languages like C/C++ and Java. Cython (C-extensions for Python) is a language close to Python with support for C to generate efficient C code.
Sinking the Titanic
the iceberg that sank large software projects...

The Titanic refers to the iceberg. What we see is the exposed interface (the domain of Python), what is hidden are the system internals (the domain of C/C++).

"Python provides a simple but powerful rapid development language, along with the integration tools to apply it in realistic development environments."

Mark Lutz, page 1195

The idea of integration is not new, but the innovation is in providing the tools.

Python is open source and cross platform.
Sinking the Titanic
the iceberg that sank large software projects...

The Titanic refers to the iceberg. What we see is the exposed interface (the domain of Python), what is hidden are the system internals (the domain of C/C++).

"Python provides a simple but powerful rapid development language, along with the integration tools to apply it in realistic development environments."

Mark Lutz, page 1195

The idea of integration is not new, but the innovation is in providing the tools.

Python is open source and cross platform.
Sinking the Titanic
the iceberg that sank large software projects...

The Titanic refers to the iceberg. What we see is the exposed interface (the domain of Python), what is hidden are the system internals (the domain of C/C++).

"Python provides a simple but powerful rapid development language, along with the integration tools to apply it in realistic development environments."

Mark Lutz, page 1195

The idea of integration is not new, but the innovation is in providing the tools.

Python is open source and cross platform.
Summary + Assignments

We offered a preview of MCS 275 (Spring 2009).

Assignments:

1. Verify if Apache is installed on your computer.
2. Install Apache if necessary.
3. Make a interactive web page with the yield and balance computation of lecture 4.

Final exam on Monday 8 December from 1 to 3PM in LC A4.