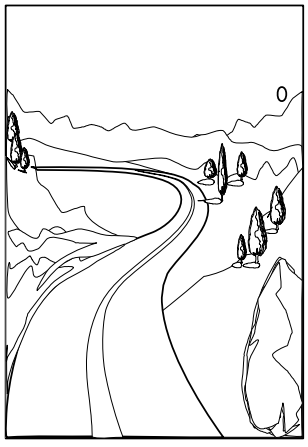
Internet Programming with Python

Wesley J. Chun
wescpy@gmail.com
CyberWeb Consulting
http://cyberwebconsulting.com

 1

The Big Picture



- Introduction
- Network Programming with sockets
- Internet Client Programming
- CGI Programming
- Conclusion

 2

Administrivia

- **Focus**
  - **Introduction to 3 or 4 distinct areas of Internet Programming**
  - **Process: lowest-level moving up to higher-level programming**
  - **Enough knowledge transfer to get you started right away**
- **Target Audience**
  - **Software Engineers, System and Network Administrators**
  - **Basic knowledge of Python or other high-level language**
  - **Other technical professionals w/programming background**
- **Instructor Background**
  - **Primarily a C/Unix background when I discovered Python**
  - **Engineer for Yahoo!Mail (address book and spellchecker)**
  - **Engineer for Yahoo! People Search (formerly `Four11.com`)**
  - **Volunteer for local user groups and Python Tutor mailing list**
  - **Wrote *Core Python Programming* (2009, 2007), *Python Fundamentals* (2009), and co-author of *Python Web Development with Django* (2009)**

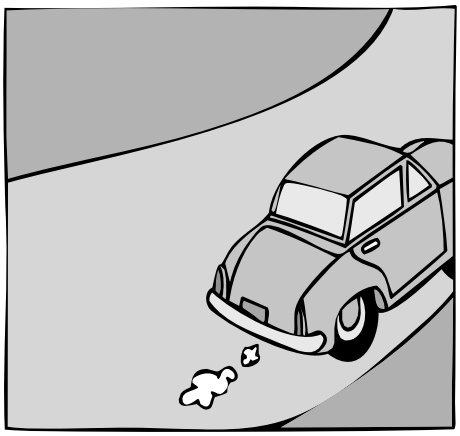
 3

Network Programming with Sockets

Segment 1

 4

Roadmap



- Introduction
- Client-Server Architecture
- Network sockets
- `socket` Module
- socket Object
- `SocketServer` Module
- Conclusion

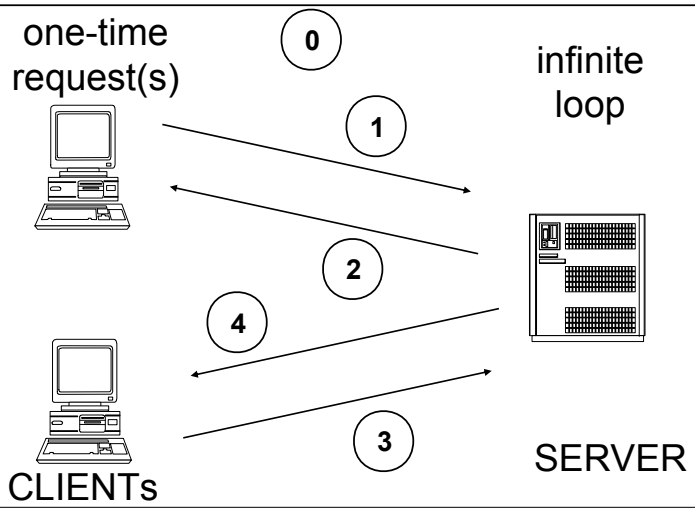
 5

Introduction to Networking

- What is networking?
  - Simply put, connecting 2 or more computers together
  - Communication via agreed-upon "protocol"
- Networking more than just wires between machines
  - Data sharing
  - Problem solving via collaboration
  - Human communication
  - Conducting of business or personal transactions
  - Provision or requisition of services
- Some network protocol suites
  - TCP/IP
  - IPX/SPX (Novell)
  - NetBEUI (Microsoft)

 6

Client/Server Architecture



 7

Client/Server Pseudocode

- **Servers run in an infinite loop**
  - **Wait for client connection**
  - **Serve the request**

```
while True:
   receive_request_from_client()
   service_request()
   send_response_to_client()
```

- **Clients make one connection for service and quit**

```
send_request_to_server()
receive_response_from_server()
```

 8

Background & Introduction to Sockets

- def: Static executable files are programs.
- def: Programs in execution are processes.
- def: Two or more processes are engaged/participating in _____ _____ (IPC) if they are passing data to and/or from each other.
- def: _____ are data structures representing the communication mechanism between processes.
- Sockets can be setup between processes...
  - On same host (File-based [AF_UNIX/AF_LOCAL])
  - On different hosts (Internet-based [AF_INET])

 9

Socket Characteristics

- Connection-oriented
  - Stream-based (SOCK_STREAM)
  - Reliable and Ordered Messages
  - Transmission Control Protocol (TCP)
  - Analogous to telephone conversation protocol
- Connectionless
  - Message/Datagram-based (SOCK_DGRAM)
  - Unreliable and Not-necessarily-ordered Messages
  - User Datagram Protocol (UDP)
  - Analogous to postal service delivery protocol
- Underlying Infrastructure IPC Mechanism Combinations
  - SOCK_STREAM + AF_INET (TCP/IP)
  - SOCK_DGRAM + AF_INET (UDP/IP)
  - Can also use both with AF_UNIX / AF_LOCAL

 10

Connection-Oriented Call Sequence

Server	Client
`ss = socket()`	`cs = socket()`
`ss.bind()`	
`ss.listen()`	`cs.connect()`
`clnt_loop:`	
`cs = ss.accept()`	
`comm_loop:`	*`comm_loop:`*
`recv()/send()`	`send()/recv()`
`send()/recv()`	`recv()/send()`
`cs.close()`	`cs.close()`
`ss.close()`	

- **Something to think about…**
  Receiving other calls while you are on the phone

 11

Connectionless Call Sequence

Server	Client
`ss = socket()`	`cs = socket()`
`ss.bind()`	
`loop:`	*`loop:`*
`recvfrom()/sendto()`	`sendto()/recvfrom()`
`ss.close()`	`cs.close()`

- **Something to think about…**
  Receiving letters from different people in the mail

 12

`socket` Module

Name	Description
`socket()`	Creates socket object
`SOCK_STREAM`	Flag to set up a TCP socket
`SOCK_DGRAM`	Flag to set up a UDP socket
`AF_INET`	Flag to set up an Internet/IP socket
`AF_UNIX`	Flag to set up a Unix socket
`gethostname()`	Returns local host machine name
`gethostbyaddr()`	Given IP address, returns hostname
`gethostbyname()`	Given hostname, returns IP address

 13

socket Object Methods

Name		Description
`accept()`	S	Accept a TCP connection
`bind()`	S	Bind socket to a port
`close()`		Close socket
`connect()`	C	Attempt to make a TCP connection
`listen()`	S	Start listening for TCP connections
`recv/from()`		Receive incoming message
`send/to()`		Send outgoing message

- **Methods for both unless marked S or C only**
- **DEMOs (TCP and UDP clients and servers)**

 14

SocketServer Module

- Simplifies all we have just seen
  - Provides socket server boilerplate code
  - Types provided: TCP & UDP for Unix & Inet families
  - Request handlers: Stream (TCP) & Datagram (UDP)
- How to use `SocketServer`
  - Much simpler than our first examples
  - Create a request handling class with method
  - Create a socket server given the address (host and port combination) and pass it your handler class
  - Enter server's infinite loop
- Renamed to `socketserver` in 3.x

 15

Creating a TCP server w/SocketServer

BaseRequestHandler

```
class MyReqHdlr(BaseRH):
   def handle():
      recv()/send()

ss = TCPServer()
ss.serve_forever()
```

StreamRequestHandler

```
class MyReqHdlr(StreamRH):
   def handle():
      read()/write()

ss = TCPServer()
ss.serve_forever()
```

- Base request handlers require socket-like access
- Stream and Datagram RHs provide more file-like access
- Setting up a UDP server is similar

 16

Asynchronous Service

- TCP: we have just seen are synchronous
  - This means only one client
  - Types provided: TCP & UDP for Unix & Inet families
  - Request handlers: Stream (TCP) & Datagram (UDP)
- 3 ways of handling asynchronous service
  - UDP: "poor man's asynchronicity"
  - `asyncore` provides asynchronous service by using `select` and managing clients via an event loop
  - `SocketServer`... features asynchronous handlers
    - multiple threads (Threading{TCP,UDP}Server)
    - multiple processes (Forking{TCP,UDP}Server)
    - same applies to Unix family sockets

 17

Conclusion

- Networking
  - Enables problem-solving on a larger scale
  - Gives computers more ability than if standalone
  - With Python, it's simplified and relatively painless
- Where can we go from here?
  - Create higher-level communication protocols
  - Use higher-level protocols with more insight
  - See `Demos/sockets` for more working examples
  - Also see the Twisted framework (twistedmatrix.com)
  - Add a graphical user interface (GUI): chat/IM app!

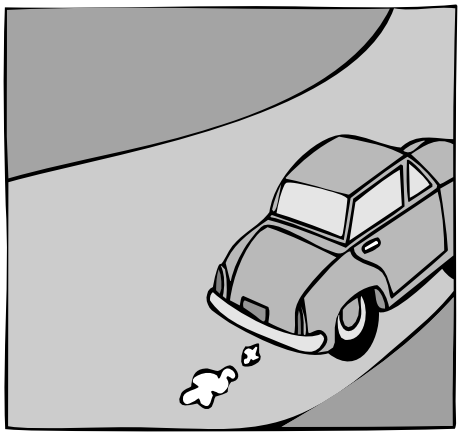
 18

Internet Client Programming

Segment 2

 19

Roadmap



- Introduction
- FTP
- NNTP
- POP3
- SMTP
- Web Clients
- Conclusion

 20

What is an Internet Client?

- Simply put:
  - Any application which uses an Internet "service"
  - Communication via agreed-upon "protocol"
- Some Internet protocols to look at:
  - File Transfer Protocol (FTP)
  - News-to-News Protocol (NNTP)
  - Post Office Protocol version 3 (POP3)
  - Hypertext Transfer Protocol (HTTP)
- Applications which use those protocols to connect to a server for "service" are *clients* of that server
  - Client-Server architecture? You bet.

 21

File Transferring Protocols

- Internet file transferring protocols:
  - File Transfer Protocol (FTP)
  - Unix-to-Unix Copy Protocol (UUCP)
  - Hypertext Transfer Protocol (HTTP)
  - Remote (Unix) file copy:
    - `rcp`, `scp` and `rsync` based on Unix `cp` command
- Today, HTTP, FTP, and scp/rsync remain popular
  - HTTP for web-based file (primarily download)
  - scp/rsync for secure file copying (upload or download)
  - FTP for web-based and text-based file transfers (up/down)

 22

File Transfer Protocol (FTP)

- File Transfer Protocol
  - Jon Postel and Joyce Reynolds
  - Request For Comment (RFC) 959 (Oct 1985)
  - Client-Server Architecture
  - Also see RFCs 2228, 2389, 2428, 2577, 2640, and 4217
- Unix multi-user concepts of username and passwords
  - FTP clients must use login/password of existing user
  - "Anonymous" logins for guest downloads
  - Clients generally time out in 15 minutes (900 seconds)

 23

Python FTP Interface: `ftplib`

- `ftplib` module... only need to import:
- `ftplib.FTP` class; some of its methods:

Name	Description
`login()`	FTP login
`quit()`	Close connection and quit
`retrlines/binary()`	Get text or binary file
`storlines/binary()`	Put text or binary file
`dir()`	Request directory listing
`cwd()`	Change working directory
`delete()`	Delete remote file

 24

Creating FTP Clients

- Connect to server
- Login
- Make service request (and hopefully get reply)
- Quit

- Python pseudocode?!?

```
from ftplib import FTP
f = FTP(your_FTP_server)
f.login('anonymous', 'guess@who.org')
...
f.quit()
```

 25

Interactive FTP Session

```
>>> from ftplib import FTP
>>> f=FTP('ftp.mozilla.org')
>>> f.login('anonymous', 'guess@who.org')
'230 Login successful.'
>>> f.pwd()
'/'
>>> f.dir()
drwxr-xr-x   20 ftp      ftp          4096 Feb 01 07:15 pub
>>> f.cwd('pub/mozilla.org')
'250 Directory successfully changed.'
>>> f.pwd()
'/pub/mozilla.org'
>>> data = []
>>> rv = f.retrlines('RETR README', data.append)
>>> rv
'226 File send OK.'
>>> len(data)
26
>>> for eachLine in data[:5]:
...  print eachLine
...
Welcome to ftp.mozilla.org!

This is the main distribution point of software and developer tools
related to the Mozilla project.  For more information, see our home
page (http://www.mozilla.org/) Go here to download Netscape Communicator:
>>> f.quit()
'221 Goodbye.'
>>>
```

 26

Network News Transfer Protocol (NNTP)

- Network News Transfer Protocol
  - Brian Kantor (UCSD) and Phil Lapsley (Cal)
  - Request For Comment (RFC) 977 (Feb 1986)
  - Utilizes the USENET News System
  - Also see RFC 2980 (update, Oct 2000)
- News archived for a certain period of time
- Login/password not necessarily required
- Server may or may not allow "posting" of messages
- Not all newsgroups may be archived on server

 27

Python NNTP Interface: `nntplib`

- `nntplib` module... only need to import:
- `nntplib.NNTP` class; some of its methods:

Name	Description
`group()`	Choose newsgroup
`quit()`	Close connection and quit
`article/head/body()`	Get entire article or just head or body
`stat/next/last()`	Set article "pointer," move to next/last
`post()`	Post article
`list()`	Request list of valid newsgroups
`xhdr()`	Retrieve specific headers from articles

 28

Creating NNTP Clients

- Connect to server
- Choose newsgroup
  - `group()` returns reply, count, first, last, group #
- Perform action:
  - Scroll through (and read) articles
  - `article()` returns reply, article #, entire message
  - Get or post article
- Quit

```
from nntplib import NNTP
n = NNTP(your_NNTP_server)
r,c,f,l,g = n.group('comp.lang.python')
…
n.quit()
```

 29

Interactive NNTP Session

```
>>> from nntplib import NNTP
>>> n = NNTP(your_NNTP_server)
>>> r,c,f,l,g = n.group('comp.lang.python') # reply, count, 1st, last, groupname
>>> r,a,i,b = n.article('110457')           # reply, artnum, artID, message body
>>> for eachLine in b:
      print eachLine

From: "Alex Martelli" <al…oo.com>
Subject: Re: Rounding Question
Date: Wed, 21 Feb 2001 17:05:36 +0100

"Remco Gerlich" <sc…d.nl> wrote in message news:slrn997ktk.h0.sc...d.nl...
> Jacob Kaplan-Moss <ja…csc.edu> wrote in comp.lang.python:
> > So I've got a number between 40 and 130 that I want to round up to the nearest 10.  That is:
> >
> >    40 --> 40, 41 --> 50, …, 49 --> 50, 50 --> 50, 51 --> 60
>
> Rounding like this is the same as adding 5 to the number and then rounding
> down. Rounding down is substracting the remainder if you were to divide by
> 10, for which we use the % operator in Python.

This will work if you use +9 in each case rather than +5 (note that he
doesn't really want rounding -- he wants 41 to 'round' to 50, for ex),

Alex

>>> n.quit()
'205 closing connection - goodbye!'
>>>
```

 30

Electronic Mail Transferring Protocols

- Internet electronic mail (e-mail) transferring protocols:
  - Message Transport Agent (MTA)
    - Responsible for routing, queuing, sending of e-mail
    - i.e., Sendmail & QMail (Unix), Microsoft Exchange (win32)
  - Message Transport System (MTS)
    - Protocol used by MTAs to transfer e-mail (host-to-host)
    - Simple Mail Transfer Protocol (SMTP) [RFCs 821 & 2821]
  - (Message) User Agent ([M]UA)
    - Protocols used to get e-mail from servers (client-to-host)
    - Post Office Protocols (POP2) [RFC937] & (POP3) [RFC1939]
    - Internet Message Access Protocol (IMAP) [RFC2060]
    - Eudora, Outlook, Thunderbird, pine/elm, mutt, MH, mail

 31

Post Office Protocol version 3 (POP3)

- Post Office Protocol version 3
  - John Myers (CMU) and Marshall Rose (Dover Beach)
  - Request For Comment (RFC) 1939 (May 1996)
  - Also see RFCs 1957 (Jun 1996) and 2449 (Nov 1998)
- E-Mail used to be delivered to your system (via SMTP)
- Resources/complexity made running SMTP inefficient
  - Lack of resources (cycles, disk space, superuser access)
  - Expensive to keep/maintain 24x7x365 Internet connectivity
- Users should be given "local control" of their mail
  - Such access is possible with UA mail clients

 32

Python POP3 Interface: `poplib`

- **`poplib` module… only need to import:**
- **`poplib.POP3{,SSL}` classes… some methods:**

Name	Description
`user()`	Login to mail server with user name
`pass_()`	Send user password to server
`list()`	List messages and message sizes
`retr()`	Retrieve an e-mail message
`dele()`	Delete an e-mail message
`quit()`	Close connection and quit
`stat()`	Get number of messages & mbox size

 33

Creating POP3 Clients

- **Connect to server**
- **Login**
- **Make service requests**
- **Quit**

```
from poplib import POP3
p = POP3(your_POP_server)
p.user('wesley')
…
p.pass_('secret')
…
p.quit()
```

 34

Interactive POP3 Session

```
>>> from poplib import POP3
>>> p = POP3(your_POP_server)
>>> p.user('wesley')
'+OK '
>>> p.pass_('secret')
'+OK '
>>> p.list()
('+OK ', ['1 3209', '2 20762', '3 15409', '4 1480', '5 251', '6 2065', '7 3681', '8 2129', '9
4941'], 73)
>>> h, m, o = p.retr(5)     # reply headers, message, octets (message size)
>>> h, o
('+OK ', 251)
>>> for e in m:
      print e

Date: Mon, 19 Mar 2001 16:31:26 -0800 (PST)
From: cixzkeblmv@chinahot.net
To: pixeajuocz@msn.com
Subject: You Can Do This Too!

Learn How To Make $1,875 Or MORE Every Week, Week After Week While Staying At Home.
No MLM No Selling No Junk

>>> p.dele(5)
'+OK '
>>> p.stat()
(8, 53676)
>>> p.quit()
'+OK '
>>>
```

 35

Email Download Miscellania

- IMAP module `imaplib`
  - Similarly-named classes, i.e., `IMAP4` and `IMAP4_SSL`
  - Protocol somewhat more complex than POP3
  - Likely will use the `login()`, `select()`, `search`, `fetch()`, `close()`, `logout()` methods
- Special notes for Gmail users:
  - Requires SSL (either `POP3_SSL` or `IMAP4_SSL`)
  - Connect via IMAP4 to port 993
  - Connect via POP3 to port 995
  - NEXT: Sending email via SMTP at ports 465 or 587
    - Requires EHLO, STARTTLS, EHLO before login

 36

Simple Mail Transfer Protocol (SMTP)

- Simple Mail Transfer Protocol (plus Extended SMTP)
  - Jonathan B. Postel
  - Request For Comment (RFC) 821 (Aug 1982)
  - Updated to RFC 2821 (Apr 2001) by J. Klensin
  - Related RFCs: 876, 1123, 1652, 1869, 2505, 3207, 3974
- E-Mail "hops" from MTA-to-MTA via SMTP
- Continues until e-mail reaches final destination
- Well-known SMTP servers include:
  - Open source: sendmail, exim, postfix, qmail
  - Commercial: Microsoft, IBM/Lotus, Novell

 37

Python SMTP Interface: `smtplib`

- `smtplib` module… only need to import:
- `smtplib.SMTP` class; some of its methods:

Name	Description
`helo(), ehlo()`	SMTP & ESMTP server greeting
`starttls()`	Start *Transport Layer Security* mode
`sendmail()`	Sends e-mail message
`login()`	Login to SMTP-AUTH server
`set_debuglevel()`	Sets debug level
`quit()`	Close connection and quit

 38

Creating SMTP Clients

- Connect to server
- Login (if applicable)
- Make service requests
- Quit

```
from smtplib import SMTP
s = SMTP(your_SMTP_server)
…
s.sendmail(sender, recips, msg)
…
s.quit()
```

 39

Interactive SMTP Session

```
>>> from smtplib import SMTP
>>> s = SMTP(your_SMTP_server)
>>> s.sendmail('you@your_email_server',
('guido@python.org', 'wescpy@gmail.com'),
'\r\n'.join(
   'From: you@your_email_server',
   'To: wescpy@gmail.com, guido@python.org',
   'Subject: test msg',
   '',
   'test',
   '.'
))
>>> s.quit()
'+OK '
```

 40

Other Internet Protocol Clients

- Other Internet application protocols similar
  - `telnetlib` Remote host login (see below)
  - `imaplib` Email download via IMAP4
  - `xmlrpclib` create XML-RPC clients
    - Renamed to `xmlrpc.client` in Python 3.x

```
# telnetClient.py
import telnetlib
import getpass

HOST = "localhost"
telnet = telnetlib.Telnet(HOST)
telnet.read_until("login: ")
login = raw_input("login: ")
telnet.write(login + '\n')
telnet.read_until("Password:")
passwd = getpass.getpass()
telnet.write(passwd + '\n')
telnet.write("ls\n")
telnet.write("exit\n")
print telnet.read_all()
telnet.close()
```

```
% telnetClient.py
login: wesley
Password:

Last login: Mon Jun 10 23:03:24 from solo
FreeBSD 4-2 (SNPP) #1: Mon Apr 22 14:09:03 PDT 2002

Welcome to FreeBSD!

% code              index.html      public_html
dead.letter     mail            tmp
% logout
```


Building Web Clients with Python

- `urllib` and `urlparse` modules
- Popular module functions are:

	Name	Description
urllib	`urlopen()`	Open URL like a file
	`urlretrieve()`	Download web document to disk
	`quote/_plus()`	URL-quote string/use "+" for space
	`unquote/_plus()`	URL-unquote string/use "+" for space
	`urlencode()`	Encode dictionary to key-value string
urlparse	`url{,un}parse()`	(Un)Parse URL into list of components
	`urljoin()`	Merge header and relative URL

Creating Web Clients

- Connect to server
- Send URL (static or CGI) [web client request]
- Retrieve result
- Quit

```
from urllib import urlopen, urlretrieve
f = urlopen('http://python.org')
data = f.readlines()
f.close()
```

```
html, hdr = urlretrieve('http://python.org')
f = open(html, 'r')
data = f.readlines()
f.close()
```

 43

Other Web Programming Miscellania

- `urllib` module can "speak" both HTTP and FTP
- `httplib` module used to create raw HTTP clients
  - (not commonly used -- `urllib` generally sufficient)
- `urllib2` extensible library for opening URLs
  - Classes/functions for proxies, digests, cookies, etc.
- `urllib` and `httplib` speak SSL
  - Secure Socket Layer version 3 via OpenSSL library
- Other packages and modules:
  - cgi, htmllib, Cookie, mailcap, robotparser, mimetools, mimetypes, *HTTPServer, webbrowser, cgitb, HTMLParser, cookielib, wsgiref, htmlentitydefs
  - 3rd party: BeautifulSoup, lxml, html5lib, mechanize
  - Testing: Windmill, Selenium, twill

 44

Conclusion

- Internet (Client) Programming
  - Internet protocols are application-oriented
  - Provides higher-level interface over sockets
  - Python makes it even easier and more painless
- Where can we go from here?
  - Clients of well-established servers
  - Multiple clients of differing protocols
  - Multithreaded/multiprocessed servers
  - Asynchronous client-server systems
  - Graphical user interface (GUI) applications
  - Server-side programming

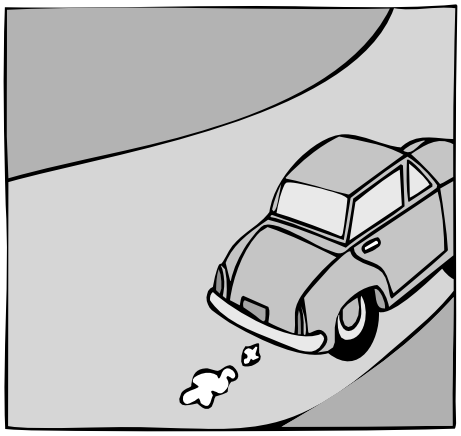
 45

CGI Programming

Segment 3

 46

Roadmap



- Introduction
- CGI Setup
- CGI
- Additional Resources
- Conclusion

 47

Introduction to CGI

- When the Web was young...
  - Web documents were static (.html files)
  - No applications on the Web
- User input desired
  - Specialized/custom/unique user input (forms)
  - Online shopping, banking, etc.
  - Server only returns static data
    - Need application to process user input
  - Side effect: Dynamically-generated HTML needed
- Access to handling application through Web server
  - Common Gateway Interface (CGI)

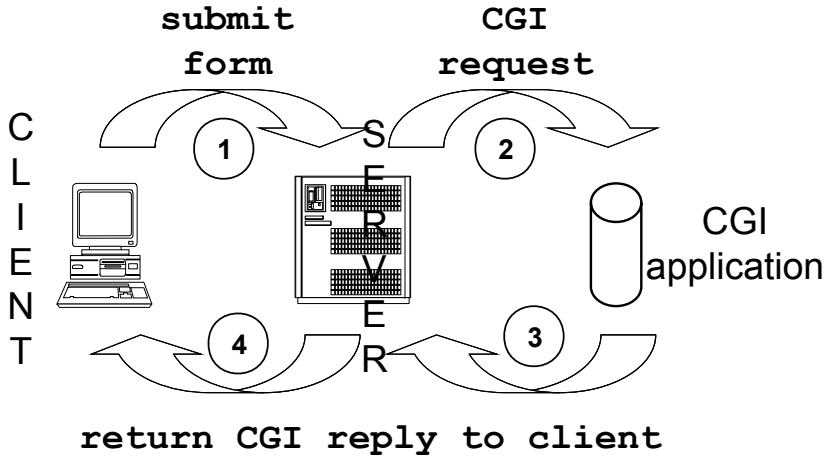
 48

Making CGI Happen

- **Preliminary work... CGI Setup**
  - **Configure your Web server for CGI (and Python)**
  - **Design/create Web pages with forms (HTML)**
- **What is CGI?**
  - **Take input from user** (forwarded through server)
  - **Process data and obtain results**
  - **Generate HTML to return** (including HTTP headers)
  - **Send output to user** (via stdout then through server)
- **Keep these in mind...**
  - **Errors are valid Web pages**
  - **"Internal Server Error"s are your mortal enemy**

 49

CGI: 2-Tier Client-Server Architecture



 50

Configure Server for CGI (& Python)

- Edit Web server configuration files (`/conf` directory)
  - Reset/restart server with each config file update
- Test with simple (bundled) CGI sample scripts first
- Then configure Python as a CGI handler
  - Server must recognize `.py` requests
  - Set location of Python CGI scripts
- Production: Integrate Python into Web server
  - I.e., Apache modules `mod_python` or `PyApache`
  - Performance hindered by interpreter launch

Create Web Pages with Forms

- Use FORM directive and INPUT mechanisms

```
<FORM ACTION="your_Python_script.py">
<INPUT TYPE=... NAME=...>
       :
<INPUT TYPE=submit></FORM>
```

- HTML provides a variety of input "widgets"

```
checkbox, file, hidden, image, password,
radio, reset, submit, text, textarea
```

- Each input type must have a CGI variable name

Web Pages with Forms (foo.html)

- Create form for user-filled data:

```
<!-- This page asks a user for name and phone# -->
<HTML><BODY>
<FORM ACTION="/cgi-bin/foo.py">

Enter Name:
<INPUT TYPE=text NAME=name SIZE=30>

<P>

Enter Telephone Number:
<INPUT TYPE=text NAME=phone SIZE=30>

<INPUT TYPE=submit>
</FORM></BODY></HTML>
```

 53

Taking Input from the User (foo.py)

- `cgi` module
- `cgi.FieldStorage()` dictionary-like class

```
"This script saves user input from form"

import cgi

form = cgi.FieldStorage()

# person = form['name'].value    # different names
# number = form['phone'].value   # ... are OK

name = form['name'].value        # same names
phone = form['phone'].value      # ... are better
```

 54

Confirming Input from User

- Blank/unchecked field means variable NOT passed
- Must be checked manually: Use `in` operator for dictionaries:

```
import cgi
form = cgi.FieldStorage()

if 'name' in form
    name = form['name'].value
else:
    name = 'NEW USER'

# (similar for 'phone')
```

 55

Process Data and Generate Output

- After extracting from CGI form...
  - You now have the data... do something with it!
  - I.e., access database, process CC transaction, etc.
- Generate HTML output (including HTTP headers)

```
out = '''Content-type: text/html


<HTML><BODY>
    :
</BODY></HTML>'''
```

  - Use `HTMLgen` or similar tools for complex HTML
    - (not part of Python standard library)
- DEMO

 56

Returning Data to the User

- Data returned to the user (through server)
  - Send results to standard output

```
print out
```

- Single string better than multiple calls to `print`

```
print 'Content-type: text/html\n\n'
print '<HTML><BODY>'
            :
print '</BODY></HTML>'
```

- Why?

 57

Error Pages: Valid CGI Transactions

- Don't forget about errors... they are valid Web pages!
- Must also return valid HTTP headers and HTML

```
out = '''Content-type: text/html


<H1>ERROR</H1>
Invalid input received... try again!
<FORM><INPUT TYPE=button VALUE=Back
ONCLICK="window.history.back()"></FORM>'''
```

- (`ONCLICK` directive above is JavaScript)

 58

"Internal Server Error"s

- ISEs (HTTPD server 500-errors)
  - These are your mortal enemies
  - Means CGI application failure
- Potential Causes
  - Bad HTTP headers and/or bad HTML
  - Python failure (most likely)
    - CGI script crapped out, resulting in...
    - Python exception output which means... (see above)
- Debugging technique: "print statements"
  - Send output to `sys.stderr` and check error log
  - Can replace `sys.stdout` or use new `print` syntax
  - Always use the `cgitb` (CGI traceback) module

 59

Scalability and Adding Complexity

- CGI can generate both form & results pages
- Create error screens (valid HTML for user)
- Make interactive pages (add state to surfing)
- Interface with network or operating system
- Connect to database or other transactional API
- Can use tools output complex HTML
  - i.e., `HTMLgen` and its descendants
- To support more features and for better URL usage, try advanced servers like CherryPy

 60

Additional Online Resources

- Python.org Web Programming Topic Guide
  - http://www.python.org/topics/web
- Linux.com: An introduction to CGI scripting with Python by Robert Currier
  - http://www.linux.com/feature/136602
- HTMLgen module
  - http://packages.debian.org/etch/python-htmlgen
- CGI Web Applications with Python by Michael Foord
  - http://pyzine.com/Issue008/Section_Articles/article_CGIOne.html
- Five Minutes to a Python CGI by David Mertz
  - http://www.ddj.com/184412536
- Writing CGI Programs in Python by Preston Landers
  - http://www.devshed.com/c/a/Python/Writing-CGI-Programs-in-Python
- Tutorials Point tutorial
  - http://www.tutorialspoint.com/python/python_cgi_programming.htm
- University of Virginia interactive tutorial
  - http://www.cs.virginia.edu/~lab2q
- About.com documents
  - http://python.about.com/od/cgiformswithpython/ss/pycgitut1.htm
  - http://python.about.com/od/cgiformswithpython/ss/test_cgi.htm

 61

Conclusion

- CGI lets web sites be interactive/dynamic
- But CGI is obsolete due to lack of scalability
  - For now, it is a great learning tool
- Where can we go from here?
  - Web development frameworks
  - Server-side middleware & backend systems
  - Creating Web Clients (other than browsers)
  - Web Servers (HTTPD)
  - Other web components:
    - Servers (CherryPy), Templates, JavaScript, etc.

 62

Web Systems Online Resources

- Zope (web application server platform)
  - http://zope.org
- Plone (content management system)
  - http://plone.org
- Web development frameworks
  - TurboGears
    - http://turbogears.org
  - Django
    - http://djangoproject.com
  - Pylons
    - http://pylonshq.com
  - web2py
    - http://web2py.com

 63

Tutorial Conclusion

- Network, Internet, and web programming open more doors
  - All make Python a powerful Internet development tool
  - Modular plug-n-play encourages code reuse and stability
  - Rapid and collaborative group development environment
- Suggested Reading:
  - *Foundations of Network Programming with Python* (Goerzen)
  - *Core Python Programming* (Chun)
    - http://corepython.com
  - *Python Web Programming* (Holden)
  - *Python in a Nutshell* (Martelli)
  - *Python Essential Reference* (Beazley)
  - Python Quick Reference Guide (Gruet)
    - http://rgruet.free.fr/#QuickRef
- Contact: Wesley J. Chun, wescpy@gmail.com
  - http://cyberwebconsulting.com

 64

Mar 18, 09 23:59 **tsTclnt.py** Page 1/1

```
#!/usr/bin/env python

from socket import *

HOST = 'localhost'
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

tcpCliSock = socket(AF_INET, SOCK_STREAM)
tcpCliSock.connect(ADDR)

while True:
    data = raw_input('> ')
    if not data:
        break
    tcpCliSock.send(data)
    data = tcpCliSock.recv(BUFSIZ)
    if not data:
        break
    print data

tcpCliSock.close()
```

Mar 18, 09 23:59 **tsTserv.py** Page 1/1

```
#!/usr/bin/env python

from socket import *
from time import ctime

HOST = ''
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

tcpSerSock = socket(AF_INET, SOCK_STREAM)
tcpSerSock.bind(ADDR)
tcpSerSock.listen(5)

while True:
    print 'waiting for connection...'
    tcpCliSock, addr = tcpSerSock.accept()
    print '...connected from:', addr

    while True:
        data = tcpCliSock.recv(BUFSIZ)
        if not data:
            break
        tcpCliSock.send('[%s] %s' % (ctime(), data))

    tcpCliSock.close()
tcpSerSock.close()
```

Mar 18, 09 23:59 **tsUclnt.py** Page 1/1

```
#!/usr/bin/env python

from socket import *

HOST = 'localhost'
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

udpCliSock = socket(AF_INET, SOCK_DGRAM)

while True:
    data = raw_input('> ')
    if not data:
        break
    udpCliSock.sendto(data, ADDR)
    data, ADDR = udpCliSock.recvfrom(BUFSIZ)
    if not data:
        break
    print data

udpCliSock.close()
```

Mar 18, 09 23:59 **tsUserv.py** Page 1/1

```
#!/usr/bin/env python

from socket import *
from time import ctime

HOST = ''
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

udpSerSock = socket(AF_INET, SOCK_DGRAM)
udpSerSock.bind(ADDR)

while True:
    print 'waiting for message...'
    data, addr = udpSerSock.recvfrom(BUFSIZ)
    udpSerSock.sendto('[%s] %s' % (ctime(), data), addr)
    print '...received from and returned to:', addr

udpSerSock.close()
```

Mar 18, 09 23:59 **tsTclntNew.py** Page 1/1

```
#!/usr/bin/env python

from socket import *

HOST = 'localhost'
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

tcpCliSock = socket(AF_INET, SOCK_STREAM)
tcpCliSock.connect(ADDR)

while True:
    data = raw_input('> ')
    if not data:
        break
    tcpCliSock.send(data)
    print "   ... waiting for reply ..."
    data = tcpCliSock.recv(BUFSIZ)
    if not data:
        break
    print data

tcpCliSock.close()
```

Mar 18, 09 23:59 **tsTservNew.py** Page 1/1

```
#!/usr/bin/env python

from socket import *

HOST = ''
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

tcpSerSock = socket(AF_INET, SOCK_STREAM)
tcpSerSock.bind(ADDR)
tcpSerSock.listen(5)

done = False
while not done:
    print 'waiting for connection...'
    tcpCliSock, addr = tcpSerSock.accept()
    print '...connected from:', addr

    while True:
        data = tcpCliSock.recv(BUFSIZ)
        if not data:
            break
        print data
        data = raw_input('> ')
        if not data:
            done = True
            break
        tcpCliSock.send(data)
        print "   ... waiting for reply ..."

    tcpCliSock.close()
tcpSerSock.close()
```

Mar 18, 09 23:59 **tsTclntSSBRH.py** Page 1/1

```
#!/usr/bin/env python

from socket import *

HOST = 'localhost'
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

while True:
    tcpCliSock = socket(AF_INET, SOCK_STREAM)
    tcpCliSock.connect(ADDR)
    data = raw_input('> ')
    if not data:
        break
    tcpCliSock.send(data)
    data = tcpCliSock.recv(BUFSIZ)
    if not data:
        break
    print data
    tcpCliSock.close()
```

Mar 18, 09 23:59 **tsTservSSBRH.py** Page 1/1

```
#!/usr/bin/env python

import SocketServer
from time import ctime

HOST = ''
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

class MyRequestHandler(SocketServer.BaseRequestHandler):

    def handle(self):
        print '...connected from:', self.client_address
        self.request.send(
            '[%s] %s' % (ctime(), self.request.recv(BUFSIZ))
        )

tcpSerSock = SocketServer.TCPServer(ADDR, MyRequestHandler)
print 'waiting for connection...'
tcpSerSock.serve_forever()
```

Mar 18, 09 23:59 **tsTclntSSSRH.py** Page 1/1

```
#!/usr/bin/env python

from socket import *

HOST = 'localhost'
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)

while True:
    tcpCliSock = socket(AF_INET, SOCK_STREAM)
    tcpCliSock.connect(ADDR)
    data = raw_input('> ')
    if not data:
        break
    tcpCliSock.send(data+'\n')
    data = tcpCliSock.recv(BUFSIZ)
    if not data:
        break
    print data
    tcpCliSock.close()
```

Mar 18, 09 23:59 **tsTservSSSRH.py** Page 1/1

```
#!/usr/bin/env python

import SocketServer
from time import ctime

HOST = ''
PORT = 21567
ADDR = (HOST, PORT)

class MyRequestHandler(SocketServer.StreamRequestHandler):
    def handle(self):
        print '...connected from:', self.client_address
        self.wfile.write('[%s] %s\n' % (
            ctime(), self.rfile.readline().strip())
        )

tcpSerSock = SocketServer.TCPServer(ADDR, MyRequestHandler)
print 'waiting for connection...'
tcpSerSock.serve_forever()
```

Mar 18, 09 23:59 **tsTclntTW.py** Page 1/1

```
#!/usr/bin/env python

from twisted.internet import protocol, reactor

HOST = 'localhost'
PORT = 21567

class TSClntProtocol(protocol.Protocol):
    def sendData(self):
        data = raw_input('> ')
        if data:
            self.transport.write(data)
        else:
            self.transport.loseConnection()

    def connectionMade(self):
        self.sendData()

    def dataReceived(self, data):
        print data
        self.sendData()

class TSClntFactory(protocol.ClientFactory):
    protocol = TSClntProtocol
    clientConnectionLost = clientConnectionFailed = \
        lambda self, connector, reason: reactor.stop()

reactor.connectTCP(HOST, PORT, TSClntFactory())
reactor.run()
```

Mar 18, 09 23:59 **tsTservTW.py** Page 1/1

```
#!/usr/bin/env python

from twisted.internet import protocol, reactor
from time import ctime

PORT = 21567

class TSServProtocol(protocol.Protocol):

    def connectionMade(self):
        clnt = self.clnt = self.transport.getPeer().host
        print '...connected from:', clnt

    def dataReceived(self, data):
        self.transport.write(
            '[%s] %s' % (ctime(), data)
        )

factory = protocol.Factory()
factory.protocol = TSServProtocol
print 'waiting for connection...'
reactor.listenTCP(PORT, factory)
reactor.run()
```

```
<HTML><HEAD><TITLE>
Friends CGI Demo (static screen)
</TITLE></HEAD>
<BODY><H3>Friends list for: <I>NEW USER</I></H3>
<FORM ACTION="/cgi-bin/friends1.py">
<B>Enter your Name:</B>
<INPUT TYPE=text NAME=person VALUE="NEW USER" SIZE=15>
<P><B>How many friends do you have?</B>
<INPUT TYPE=radio NAME=howmany VALUE="0" CHECKED> 0
<INPUT TYPE=radio NAME=howmany VALUE="10"> 10
<INPUT TYPE=radio NAME=howmany VALUE="25"> 25
<INPUT TYPE=radio NAME=howmany VALUE="50"> 50
<INPUT TYPE=radio NAME=howmany VALUE="100"> 100
<P><INPUT TYPE=submit></FORM></BODY></HTML>
```

```
::::::::::::::
cgihttpd.py
::::::::::::::
#!/usr/bin/env python

from CGIHTTPServer import test

if __name__ == '__main__':
    try:
        print 'Welcome to the machine...\nPress ^C once or twice to quit'
        test()
    except KeyboardInterrupt:
        print 'exiting server...'


::::::::::::::
friends1.py
::::::::::::::
#!/usr/bin/env python

import cgi

reshtml = '''Content-Type: text/html\n
<HTML><HEAD><TITLE>
Friends CGI Demo (dynamic screen)
</TITLE></HEAD>
<BODY><H3>Friends list for: <I>%s</I></H3>
Your name is: <B>%s</B><P>
You have <B>%s</B> friends.
</BODY></HTML>'''

form = cgi.FieldStorage()
who = form['person'].value
howmany = form['howmany'].value
print reshtml % (who, who, howmany)
```

```
#!/usr/bin/env python

import cgi

header = 'Content-Type: text/html\n\n'

formhtml = '''<HTML><HEAD><TITLE>
Friends CGI Demo</TITLE></HEAD>
<BODY><H3>Friends list for: <I>NEW USER</I></H3>
<FORM ACTION="/cgi-bin/friends2.py">
<B>Enter your Name:</B>
<INPUT TYPE=hidden NAME=action VALUE=edit>
<INPUT TYPE=text NAME=person VALUE="" SIZE=15>
<P><B>How many friends do you have?</B>
%s
<P><INPUT TYPE=submit></FORM></BODY></HTML>'''

fradio = '<INPUT TYPE=radio NAME=howmany VALUE="%s" %s> %s\n'

def showForm():
    friends = ''
    for i in [0, 10, 25, 50, 100]:
        checked = ''
        if i == 0:
            checked = 'CHECKED'
        friends = friends + fradio % \
            (str(i), checked, str(i))

    print header + formhtml % (friends)

reshtml = '''<HTML><HEAD><TITLE>
Friends CGI Demo</TITLE></HEAD>
<BODY><H3>Friends list for: <I>%s</I></H3>
Your name is: <B>%s</B><P>
You have <B>%s</B> friends.
</BODY></HTML>'''

def doResults(who, howmany):
    print header + reshtml % (who, who, howmany)

def process():
    form = cgi.FieldStorage()
    if form.has_key('person'):
        who = form['person'].value
    else:
        who = 'NEW USER'

    if form.has_key('howmany'):
        howmany = form['howmany'].value
    else:
        howmany = 0

    if form.has_key('action'):
        doResults(who, howmany)
    else:
        showForm()

if __name__ == '__main__':
    process()
```

```
#!/usr/bin/env python

import cgi
from urllib import quote_plus
from string import capwords

header = 'Content-Type: text/html\n\n'
url = '/cgi-bin/friends3.py'

errhtml = '''<HTML><HEAD><TITLE>
Friends CGI Demo</TITLE></HEAD>
<BODY><H3>ERROR</H3>
<B>%s</B><P>
<FORM><INPUT TYPE=button VALUE=Back
ONCLICK="window.history.back()"></FORM>
</BODY></HTML>'''

def showError(error_str):
    print header + errhtml % (error_str)

formhtml = '''<HTML><HEAD><TITLE>
Friends CGI Demo</TITLE></HEAD>
<BODY><H3>Friends list for: <I>%s</I></H3>
<FORM ACTION="%s">
<B>Your Name:</B>
<INPUT TYPE=hidden NAME=action VALUE=edit>
<INPUT TYPE=text NAME=person VALUE="%s" SIZE=15>
<P><B>How many friends do you have?</B>
%s
<P><INPUT TYPE=submit></FORM></BODY></HTML>'''

fradio = '<INPUT TYPE=radio NAME=howmany VALUE="%s" %s> %s\n'

def showForm(who, howmany):
    friends = ''
    for i in [0, 10, 25, 50, 100]:
        checked = ''
        if str(i) == howmany:
            checked = 'CHECKED'
        friends = friends + fradio % \
            (str(i), checked, str(i))
    print header + formhtml % (who, url, who, friends)

reshtml = '''<HTML><HEAD><TITLE>
Friends CGI Demo</TITLE></HEAD>
<BODY><H3>Friends list for: <I>%s</I></H3>
Your name is: <B>%s</B><P>
You have <B>%s</B> friends.
<P>Click <A HREF="%s">here</A> to edit your data again.
</BODY></HTML>'''

def doResults(who, howmany):
    newurl = url + '?action=reedit&person=%s&howmany=%s' % \
        (quote_plus(who), howmany)
    print header + reshtml % (who, who, howmany, newurl)

def process():
    error = ''
    form = cgi.FieldStorage()

    if form.has_key('person'):
        who = capwords(form['person'].value)
    else:
        who = 'NEW USER'

    if form.has_key('howmany'):
        howmany = form['howmany'].value
    else:
        if form.has_key('action') and \
                form['action'].value == 'edit':
            error = 'Please select number of friends.'
        else:
            howmany = 0

    if not error:
        if form.has_key('action') and \
                form['action'].value != 'reedit':
            doResults(who, howmany)
        else:
            showForm(who, howmany)
    else:
        showError(error)

if __name__ == '__main__':
        process()
```

INTRO TO DJANGO

Wesley J. Chun
Principal
CyberWeb Consulting
wescpy@gmail.com
Spring 2009

HTML/CGI Inadequate

- CGI inherently not scalable
- Tools to create web pages and respond
- Not nearly enough for web applications/services
- Database infrastructure not available
- No ability to support templates
- No real webserver support

MVC Frameworks: 1-stop shop(s)

- Full-stack (templating, DB, server) web framework
  - JavaScript library
  - Page templating system
  - Webserver
  - ORM/Database access
- Ruby has Rails, but Python has...
  - Django - all-in-one
  - TurboGears - best of breed
  - Pylons - light, flexible

Django Overview

- Developed at the Lawrence Journal-World in Kansas
- Created by experienced web developers...
- For constant journalistic requirements/deadlines
- Pythonic: follows the DRY principle
- Clean URL management
- Customimzable caching mechanism
- Internationalized support

*Supported Software

- Webservers
  - Django
  - Apache
  - ligHTTPD
  - CherryPy+WSGI
- Databases
  - MySQL
  - PostgreSQL
  - SQLite
  - Oracle

*Do some reading...

- Installation Instructions
  - http://www.djangoproject.com/documentation/install
- Documentation Central
  - http://www.djangoproject.com/documentation
- Technical Overview
  - http://www.djangoproject.com/documentation/overview
- First Tutorial
  - http://www.djangoproject.com/documentation/tutorial1

Requirements and Download

- Requires Python 2.3+
- Use its webserver or install your own
- Get a database
- Download Django
  - http://www.djangoproject.com/download

Installation and Setup

- Install it
  - Execute "[`python`] `setup.py install`" (site-packages)
- Setup PATH
  - `/usr/bin` or `C:\Python26\Scripts`
  - Make `python`(.exe) and django-admin.py path-available
- Create work area and add to PYTHONPATH
  - `/home/you/xxx` or `C:\xxx`

Building a Blog

- Example from "Python Web Development with Django"
  - by Forcier, Bissex, Chun; (c)2009 Addison Wesley
- Create project
  - `django-admin.py startproject mysite`
  - `cd mysite`
- Start webserver
  - `(./)manage.py runserver`
  - `http://localhost:8000`

Create Application

- `manage.py startapp blog`
- `cd blog`
- Edit `../settings.py`
  - Add 'mysite.blog' to INSTALLED_APPS
- Add your model to models.py

```
class BlogPost(models.Model):
    title = models.CharField(max_length=150)
    body = models.TextField(max_length=150)
    timestamp = models.DateTimeField()
```

Setup Database

- Edit `../settings.py`
- Add database
  - `DATABASE_ENGINE = 'sqlite3'`
  - `DATABASE_NAME = 'c:/xxx/django.db'`
- SyncDB
  - `../manage.py syncdb`
- Create superuser

Automatic Administration

- Edit `../settings.py`
  - Add 'django.contrib.admin' to INSTALLED_APPS
  - `../manage.py syncdb`
- Edit `../urls.py`
  - Uncomment several lines to enable admin
- Enable administration for your class
  - Edit `models.py`
  - Import the admin
  - Register your model with the admin

Interaction

- Add Content
  - `http://localhost:8000/admin`
  - Login and go to Blog posts
  - Create new blog entry
  - Create another one
- Note output Usefulness (or lack thereof)
  - Need to improve quality/relevance

Tweaking

- Changing default display
- Edit `models.py`
  - Add BlogPostAdmin class
  - `list_display = ('title', 'timestamp')`
- Note change from webserver
- Refresh page
  - http://localhost:8000/admin/blog/blogpost

*Public-facing Template

- Create archive template
  - Filename `./templates/archive.html`

```
{% for post in posts %}
<h2>{{ post.title }}</h2>
<p>{{ post.timestamp }}</p>
<p>{{ post.body }}</p>
{% endfor %}
```

*Rendering Template via View

- Create view
  - Edit file `./views.py`

```
from django.template import loader, Context
from django.http import HttpResponse
from mysite.blog.models import BlogPost
def archive(request):
    posts = BlogPost.objects.all()
    t = loader.get_template('archive.html')
    c = Context({'posts': posts})
    return HttpResponse(t.render(c))
```

*Add View Access via URLconfs

- Add URL for blog
  - Add pointer to app URLconf
    - Edit `../urls.py`
  - `(r'^blog/', include('mysite.blog.urls')),`
  - Add view to app URLconf
    - Create `./urls.py`

```
from django.conf.urls.defaults import *
from mysite.blog.views import archive
urlpatterns = patterns('',
   url(r'^$', archive),
)
```

*View Blog as a User

- Restart webserver if necessary
- View the blog entries thus far
  - `http://localhost:8000/blog`

*Template Inheritance

- Why?
- May create more than one page
- But desire consistent look-n-feel
- Create a base template
  - Add file `./templates/base.html`

*base.html

```
<html>
   <style type="text/css">
      body { color: #efd; background: #453; padding: 0
        5em; margin: 0 }
      h1 { padding: 2em 1em; background: #675 }
      h2 { color: #bf8; border-top: 1px dotted #fff;
        margin-top: 2em }
      p { margin: 1em 0 }
   </style>
   <body>
      <h1>mysite.example.com</h1>
      {% block content %}
      {% endblock %}
   </body>
</html>
```

*Extending the Base Template

- Use the archive template
  - Edit `templates/archive.html`

```
{% extends "base.html" %}
{% block content %}
    :
{% endblock %}
```

`http://localhost:8000/blog`

*Change default ordering

- Blog entries typically in reverse chrono order
- Rather than programming this via the view...
- Make change in model
  - Edit `models.py`
  - Add Meta inner class to BlogPost
    - Add ordering attribute to Meta class

```
class BlogPost(models.Model):
    :
    class Meta(object):
        ordering = ('-timestamp',)
```

*Template filters

- Filters: Django convenience utilities
- Can use to generate more user-friendly output
- ISO8601 date format "nerdy"... fix this by filtering date
  - Edit `templates/archive.html`
  - Add filter to timestamp output
    - `<p>{{ post.timestamp|date }}</p>`
  - Further enhance by using PHP date() formatting
    - `<p>{{ post.timestamp|date:"l, F jS" }}</p>`

Conclusion

- Fast to get something done in Django
- Yes, initial setup may not be trivial
- Not too much effort to create an application
- Once you have something, updates are FAST
- Now ready to do the official tutorials!