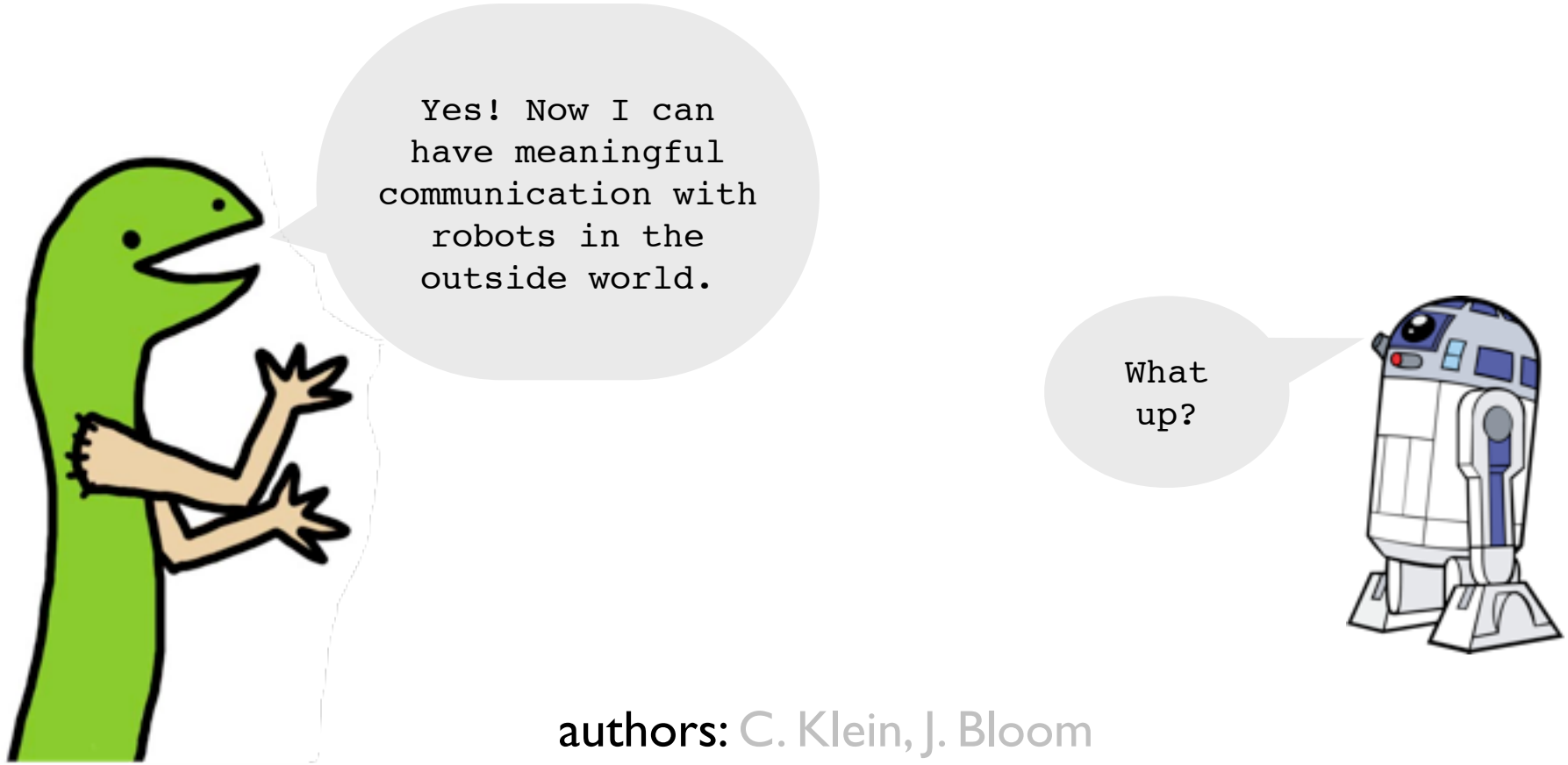
Interacting with the Outside World through Python

Part 1: Talking to Computers



authors: C. Klein, J. Bloom

Outline

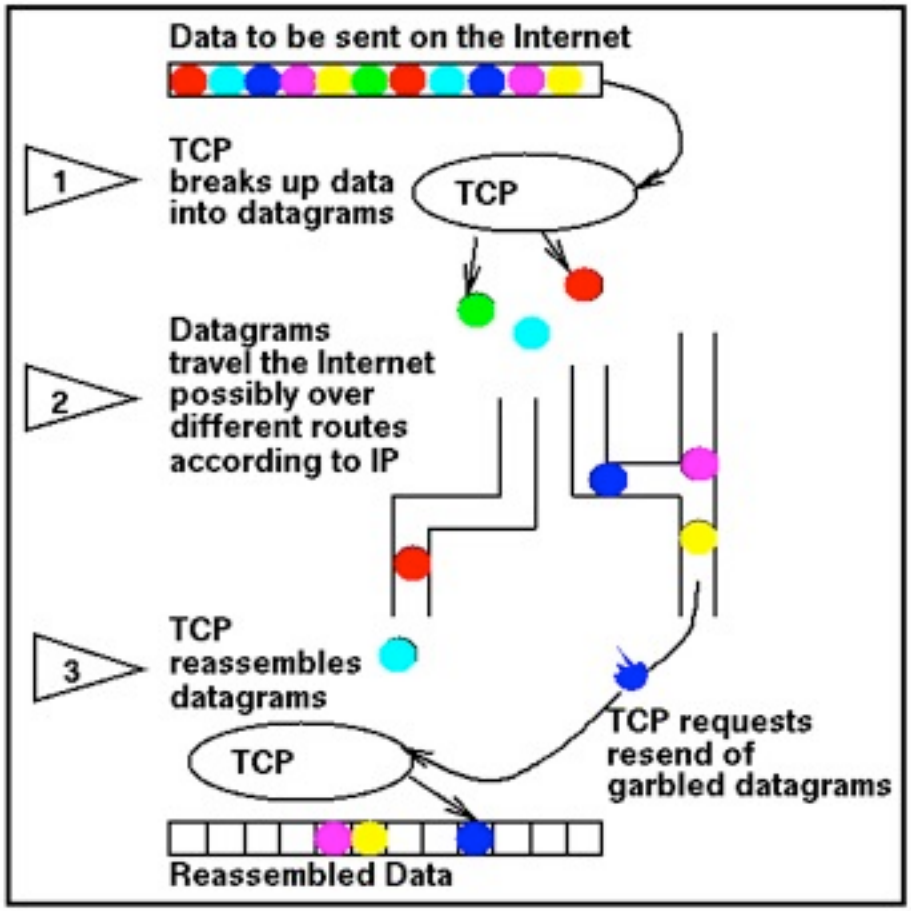
1) Websites and webservers
   - urllib, urllib2, ftplib, httplib, httplib2
   - Parsing with html5lib, BeautifulSoup

```
pip install beautifulsoup4
```

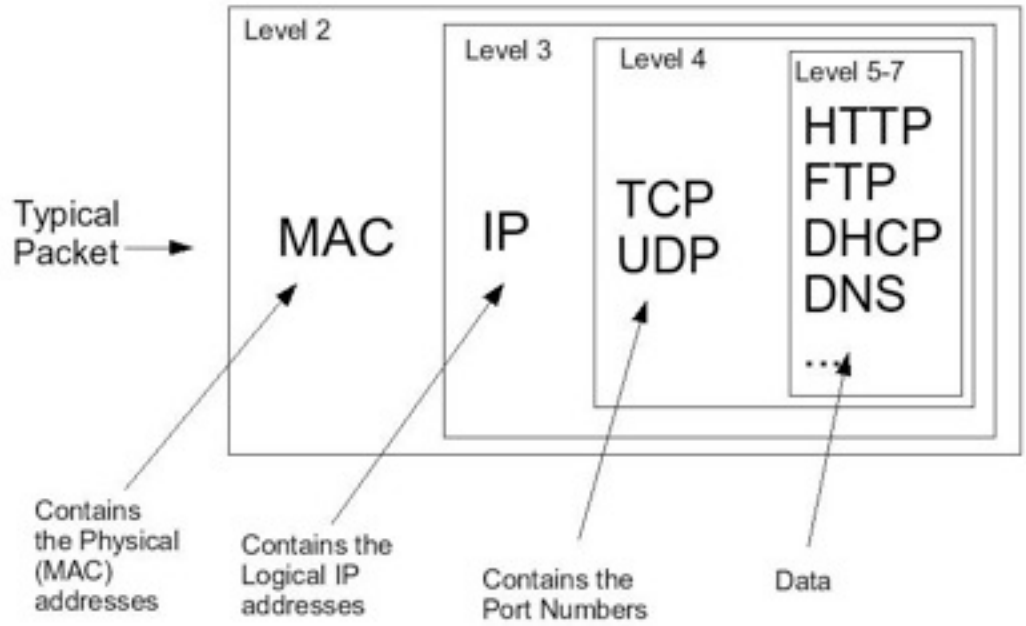
2) Transmission Control Protocol (TCP)
   - socket
3) Breakout Exercise
   - Focus on automating website access
4) Remote Procedure Call
   - SimpleXMLRPCServer, xmlrpclib

Network Communication Overview

- **TCP/IP sockets**: Most all network communication, also UDP
- TCP (Transmission Control Protocol): exchange data reliably between two network hosts
- IP (Internet Protocol): handles addressing & routing messages across one or more networks



http://jcmc.indiana.edu/vol1/issue4/tcpip.gif



Accessing a Web address (URL)

Why? Who would ever want to easily automate URL (Uniform Resource Locator) retrieval and form submission in a scripting language?

- Data mining (we'll do this in the breakout)
- Submitting information to another system
- Accessing remote compute resources ("webservices")

urllib & urllib2

The urllib provides tools & functions for high-level, but less modern, interactions.

The urllib2, more suited for complex interactions, supporting basic and digest authentication, redirections, cookies, and more.

Underlying libraries are httplib and httplib2.

FYI:
`urllib.openurl()` is deprecated in favor of `urllib2.openurl()`

urllib & urllib2

urllib docstring

Open an arbitrary URL.

See the following document for more info on URLs: "Names and Addresses, URIs, URLs, URNs, URCs", at http://www.w3.org/pub/WWW/Addressing/Overview.html

See also the HTTP spec (from which the error codes are derived):
"HTTP - Hypertext Transfer Protocol", at http://www.w3.org/pub/WWW/Protocols/

Related standards and specs:
- RFC1808: the "relative URL" spec. (authoritative status)
- RFC1738 - the "URL standard". (authoritative status)
- RFC1630 - the "URI spec". (informational status)

The object returned by URLopener().open(file) will differ per protocol. All you know is that is has methods read(), readline(), readlines(), fileno(), close() and info(). The read*(), fileno() and close() methods work like those of open files. The info() method returns a mimetools.Message object which can be used to query various info about the object, if available. (mimetools.Message objects are queried with the getheader() method.)

urllib & urllib2

urllib2 docstring

An extensible library for opening URLs using a variety of protocols

The simplest way to use this module is to call the urlopen function, which accepts a string containing a URL or a Request object (described below). It opens the URL and returns the results as file-like object; the returned object has some extra methods described below.

The OpenerDirector manages a collection of Handler objects that do all the actual work. Each Handler implements a particular protocol or option. The OpenerDirector is a composite object that invokes the Handlers needed to open the requested URL. For example, the HTTPHandler performs HTTP GET and POST requests and deals with non-error returns. The HTTPRedirectHandler automatically deals with HTTP 301, 302, 303 and 307 redirect errors, and the HTTPDigestAuthHandler deals with digest authentication.

urlopen(url, data=None) -- Basic usage is the same as original urllib. Pass the url and optionally data to post to an HTTP URL, and get a file-like object back. One difference is that you can also pass a Request instance instead of URL. Raises a URLError (subclass of IOError); for HTTP errors, raises an HTTPError, which can also be treated as a valid response.

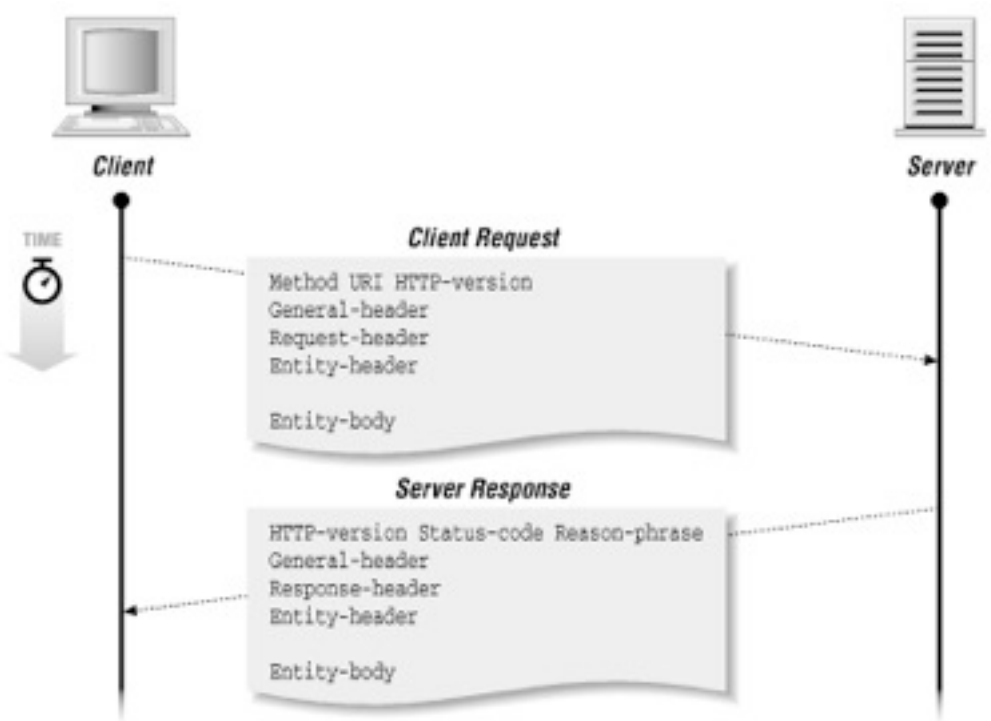
urllib & urllib2

Super simple webpage access

HTTP Overview

Hypertext Transfer Protocol

- HTTP takes place along TCP/IP sockets (typically port 80)
- HTTP is used to transmit ***resources***
  - resources can be files, query results, server side script output



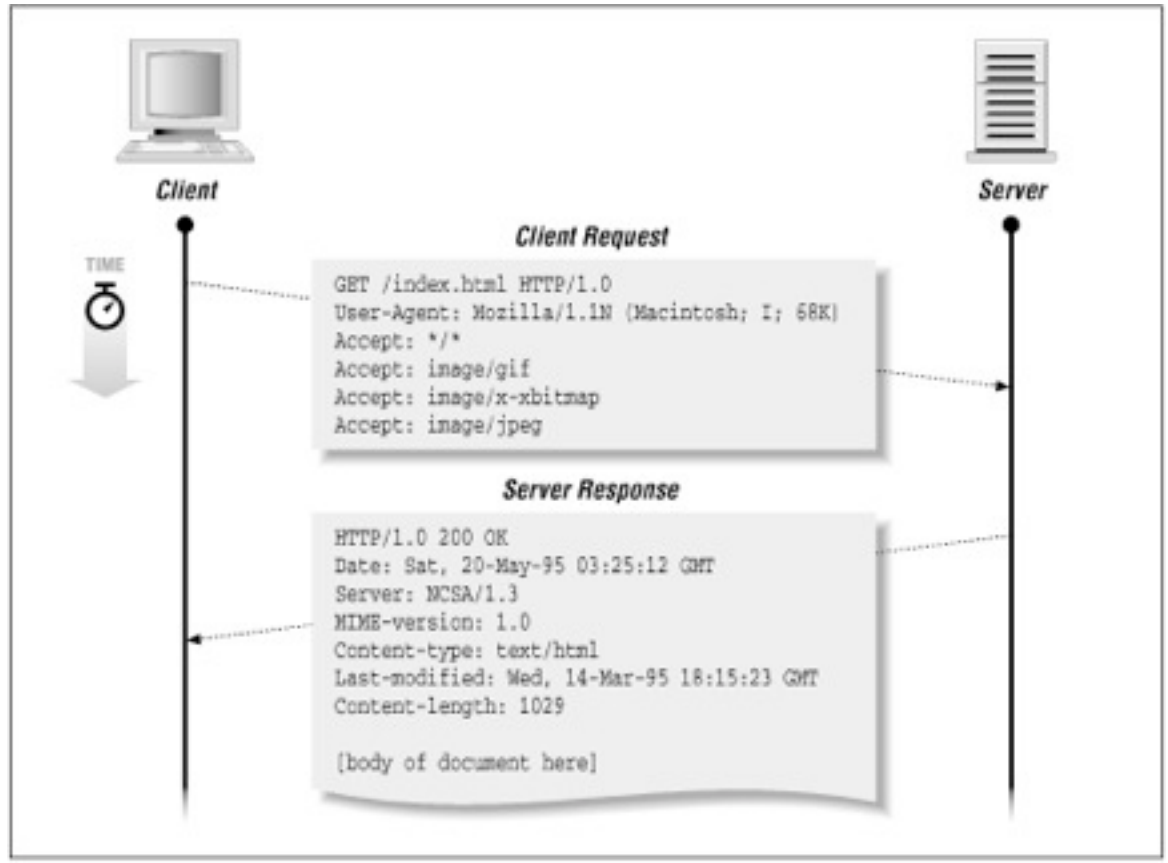
Communication initiated by *Client* opening connection & sending request message to *Server*. Server then returns a *response* message containing the resource that was requested. After delivering the response the Server closes the connection.

The two most used request methods are GET and POST, see next slide.

http://www.stepwise.hk/npwiki/Lecture/HttpWeb

HTTP GET and POST

GET: default method for retrieving resources. Form data is encoded in the URL. GET should be used when the form processing is “idempotent” - when it has no side effects. GET is basically just for retrieving data (static files).



POST places form data in the message body. It is more appropriate for wider range of processes, e.g., storing/updating data, ordering or sending a product, and sending email.

urllib & urllib2

Scripting an HTTP GET request

urllib & urllib2

Scripting an HTTP POST request

urllib & urllib2

Basic authentication

```
>>> import urllib2
>>> auth_handler = urllib2.HTTPBasicAuthHandler()
>>> auth_handler.add_password("realm", "example.com",
>>>     "username", "password")
>>> opener = urllib2.build_opener(auth_handler)
>>> response = opener.open("http://example.com/my/protected/page.html")
```

Browsers handle this by popping up a dialog box requesting you to “Enter user name and password for “realm” at http://example.com”.

urllib & urllib2

Form-based authentication

```
>>> import urllib2
>>> opener = urllib2.build_opener(urllib2.HTTPCookieProcessor())
>>> params = urllib.urlencode(dict(username="uname", password="pswd"))
>>> response = opener.open("http://example.com/login/", params)
>>> data = response.read()
>>> response.close()
>>> response = opener.open("http://example.com/my/protected/page.html")
>>> data = response.read()
>>> response.close()
```

- Login information is stored in a cookie and included in subsequent requests.
- The opener is used to POST to the login form and the protected page.

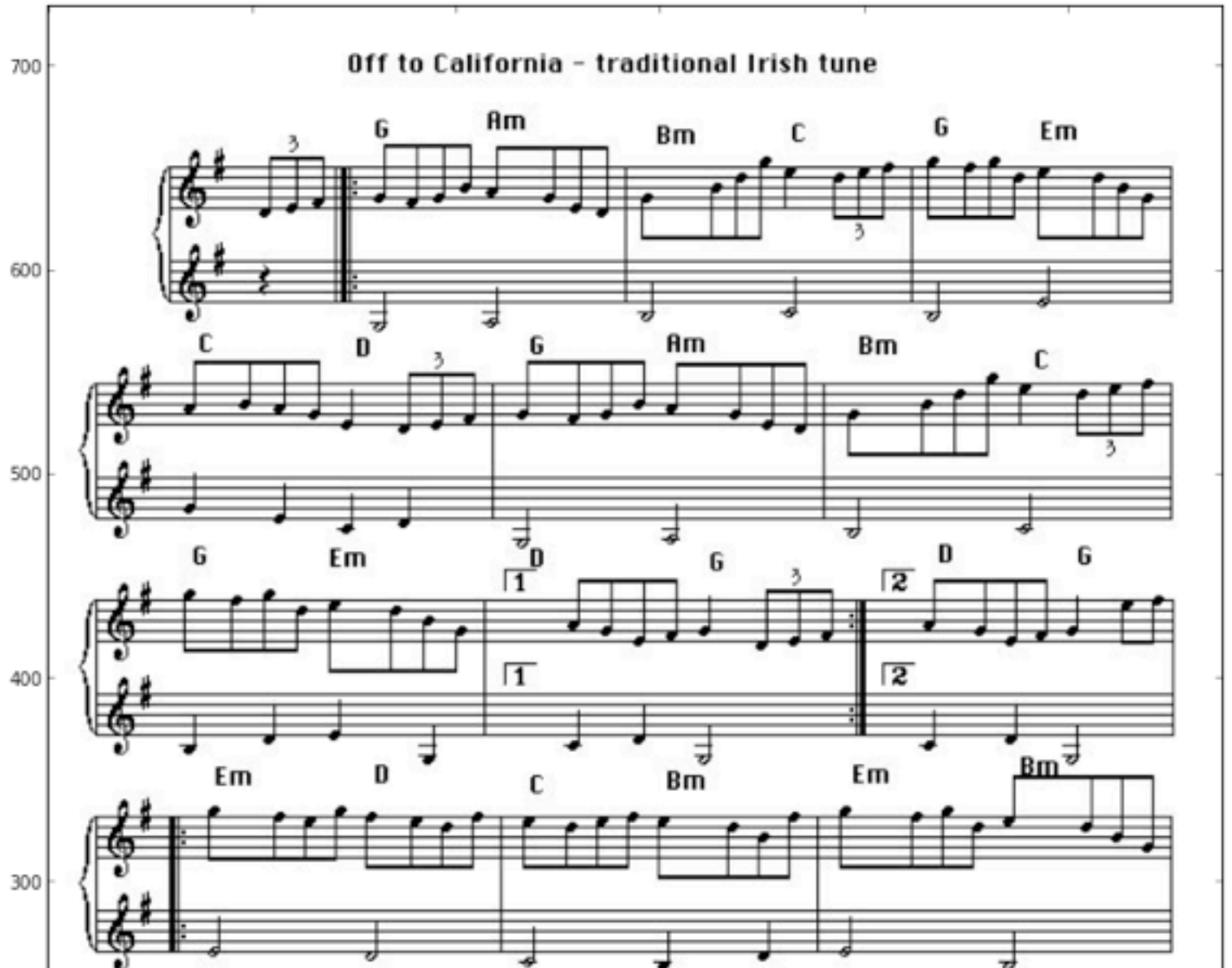
mechanize: http://wwwsearch.sourceforge.net/mechanize/doc.html

ftplib

Access an FTP server

ftplib

Sheet music file



HTML Overview

- HyperText Markup Language
  - The code in which webpages are written
  - Consists of *tags* surrounded by angled brackets, < and >
- An HTML document has a hierarchy enforced by the ordering and nesting of tags
  - It can be thought of like a tree with branches

```
1  <!doctype html>
2  <html>
3    <head>
4      <title>Hello HTML</title>
5    </head>
6    <body>
7      <p>Hello World!</p>
8    </body>
9  </html>
```

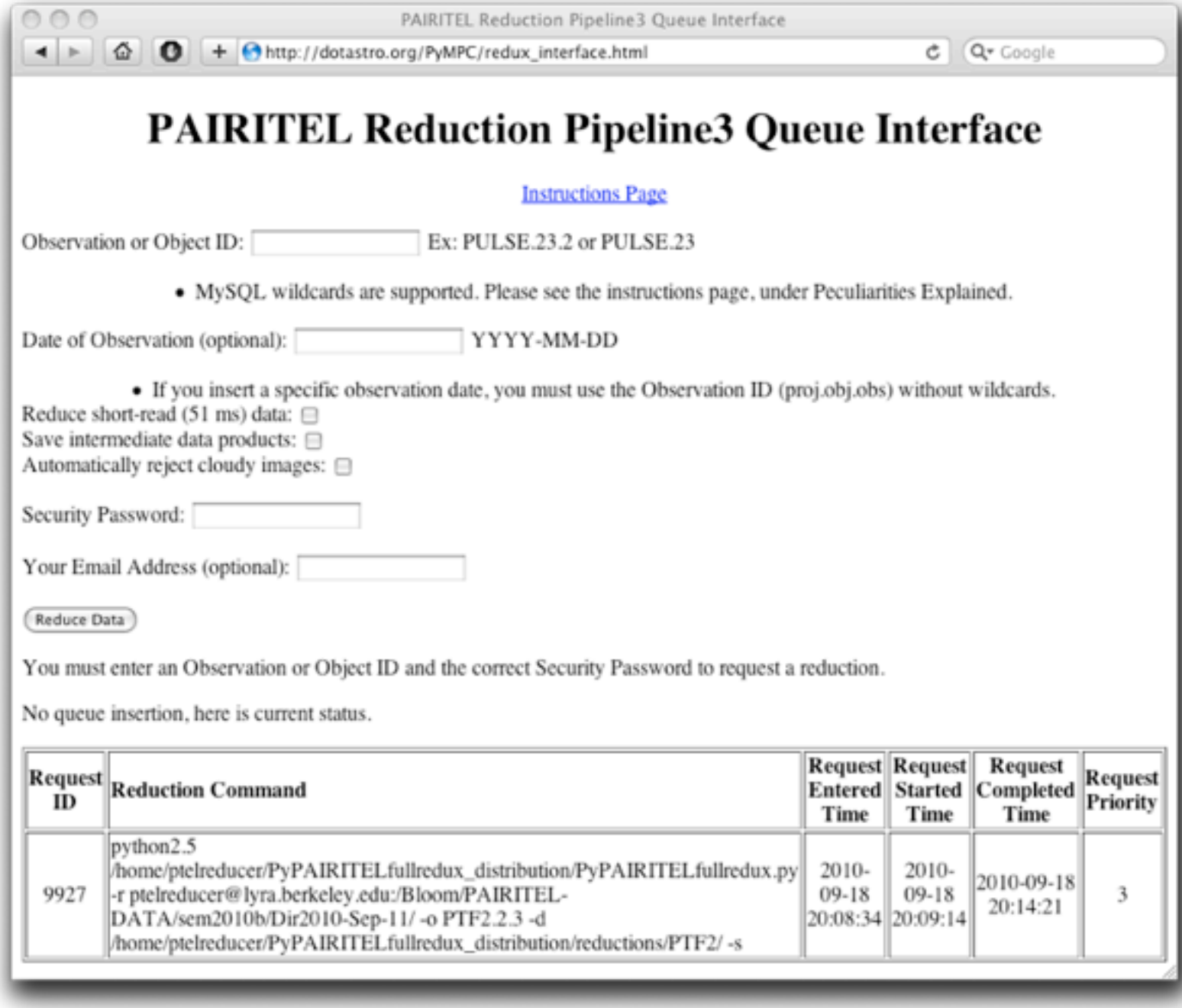
Examples at

http://www.w3schools.com/html/html_examples.asp
http://www.sheldonbrown.com/web_sample1.html

HTML Example - Code

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>PAIRITEL Reduction Pipeline3 Queue Interface</title>
</head>
<body>
  <h1 align="center">PAIRITEL Reduction Pipeline3 Queue Interface</h1>
  <p align="center"><a href="http://dotastro.org/PyMPC/pipeline3_readme.html">
  Instructions Page</a></p>
  <form action="/PyMPC/redux_interface.html" method="POST">
    <p><label>Observation or Object ID: </label>
    <input type="text" name="inr_1"> Ex: PULSE.23.2 or PULSE.23</p>
    <li align="center">MySQL wildcards are supported. Please see the instructions page,
    <p><label>Date of Observation (optional): </label>
    <input type="text" name="inr_2"> YYYY-MM-DD </p>
    <li align="center">If you insert a specific observation date, you must use the Obser
    Reduce short-read (51 ms) data:
    <input type="checkbox" name="inr_3">
    <br />

    Save intermediate data products:
    <input type="checkbox" name="inr_4">
    <br />

    Automatically reject cloudy images:
    <input type="checkbox" name="inr_5">

    <p><label>Security Password: </label>
    <input type="text" name="inr_6"></p>

    <p><label>Your Email Address (optional): </label>
    <input type="text" name="inr_7"></p>

```

HTML Example - Page



html5lib

html5lib docstring

HTML parsing library based on the WHATWG “HTML5” specification. The parser is designed to be compatible with existing HTML found in the wild and implements well-defined error recovery that is largely compatible with modern desktop web browsers.

html5lib

Parsing html

```
>>> import urllib2, urllib, html5lib
>>> response = urllib2.urlopen("http://words.bighugelabs.com/")
>>> html = response.read()
>>> response.close()
>>> doc = html5lib.parse(html, treebuilder="simpletree")
```

doc is now a tree in “simpletree” format. html5lib also supports minidom, ElementTree, lxml, and BeautifulSoup tree formats.

lxml, in particular, is good for creating well-formed html and xml.

Even web geeks need motivational speakers

Code Talkers

BeautifulSoup

BeautifulSoup docstring

Beautiful Soup parses a (possibly invalid) XML or HTML document into a tree representation. It provides methods and Pythonic idioms that make it easy to navigate, search, and modify the tree.

A well-formed XML/HTML document yields a well-formed data structure. An ill-formed XML/HTML document yields a correspondingly ill-formed data structure. If your document is only locally well-formed, you can use this library to find and process the well-formed part of it.

Beautiful Soup works with Python 2.2 and up.

<skip some verbose exposition>

For more than you ever wanted to know about Beautiful Soup, see the documentation:
http://www.crummy.com/software/BeautifulSoup/documentation.html

BeautifulSoup

More parsing

Sockets and Ports

An ***Internet socket*** is an endpoint of a bidirectional inter-process communication flow across an Internet Protocol-based computer network, such as the Internet. The term Internet sockets is also used as a name for an application programming interface (API) for the TCP/IP protocol stack, usually provided by the operating system. Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP addresses and port numbers. Each socket is mapped by the operating system to a communicating application process or thread.

A ***port*** is an application-specific or process-specific software construct serving as a communications endpoint.

Thanks to Wikipedia

socket

Provides access to the BSD socket interface using TCP or UDP.
Great for communication to any IP address (internal, LAN, or external).

Client

socket

socket docstring functions list

Functions:
socket() -- create a new socket object
socketpair() -- create a pair of new socket objects [*]
fromfd() -- create a socket object from an open file descriptor [*]
gethostname() -- return the current hostname
gethostbyname() -- map a hostname to its IP number
gethostbyaddr() -- map an IP number or hostname to DNS info
getservbyname() -- map a service name and a protocol name to a port number
getprotobyname() -- map a protocol name (e.g. 'tcp') to a number
ntohs(), ntohl() -- convert 16, 32 bit int from network to host byte order
htons(), htonl() -- convert 16, 32 bit int from host to network byte order
inet_aton() -- convert IP addr string (123.45.67.89) to 32-bit packed format
inet_ntoa() -- convert 32-bit packed format IP to string (123.45.67.89)
ssl() -- secure socket layer support (only available if configured)
socket.getdefaulttimeout() -- get the default timeout value
socket.setdefaulttimeout() -- set the default timeout value
create_connection() -- connects to an address, with an optional timeout

[*] not available on all platforms!

socket

Server

Breakout

1. grab and parse the Florida HTML from RealClearPolitics

http://www.realclearpolitics.com/epolls/2012/president/fl/florida_romney_vs_obama-1883.html#polls

2. Parse the polling data into a useable Python data structure (hint: look for `<div id="polling-data-full"`))

Breakout

3. Plot the data since the beginning of the year

4. Use `scipy.interpolate.UnivariateSpline` to get a weighted spline fit of the data for both candidates. Overplot the fits

5. What was a date when Romney was ahead?

If time, do the same for a red state and a blue state

json

JSON is a light-weight data interchange format.

Some web service APIs can output in JSON and the `json` Python module facilitates parsing.

http://www.json.org/

XML-RPC Overview

- Remote Procedure Call protocol which uses XML to encode its calls and HTTP as a transport mechanism
- A Client sends an HTTP request to a Server implementing the protocol
- The Client can pass multiple input parameters to the Server and request that the Server perform a method on those parameters and return the result (one value) in the response
- The parameters and result can be common data types (including lists of multiple values)
- Data is translated through XML (Extensible Markup Language) for transmission
    - In our usage the data is reformed into Python data structures by modules at the Client and Server

SimpleXMLRPCServer

SimpleXMLRPCServer docstring

This module can be used to create simple XML-RPC servers by creating a server and either installing functions, a class instance, or by extending the SimpleXMLRPCServer class.

It can also be used to handle XML-RPC requests in a CGI environment using CGIXMLRPCRequestHandler.

<truncated>

SimpleXMLRPCServer

Server & Client