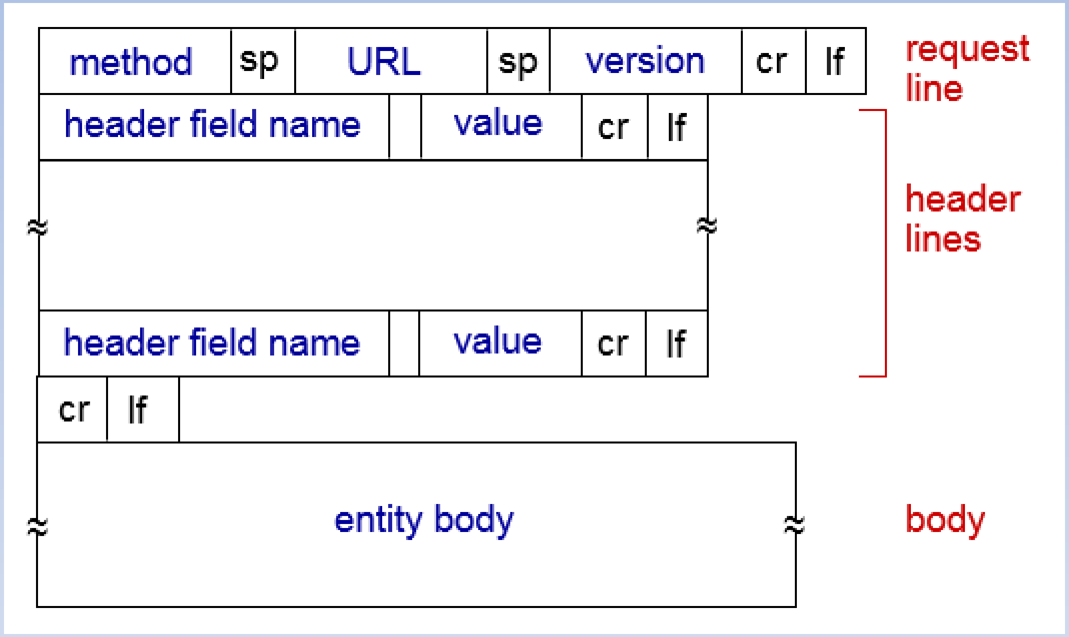
Creating Web Scrapers

- Retrieve HTML data from a domain name
- Parse that data for target information
- Store the target information
- Optionally, move to another page to repeat the process

Web Page Request

- Browser ←→ Server connection mechanism
  - Client sends stream of 0 and 1 bits (high & low voltages on a wire)
  - Bit sequence forms a message (or multiple) packet
  - Header contains client's router address as well as final destination which is the server IP address
  - Body contains specific request of web page from server

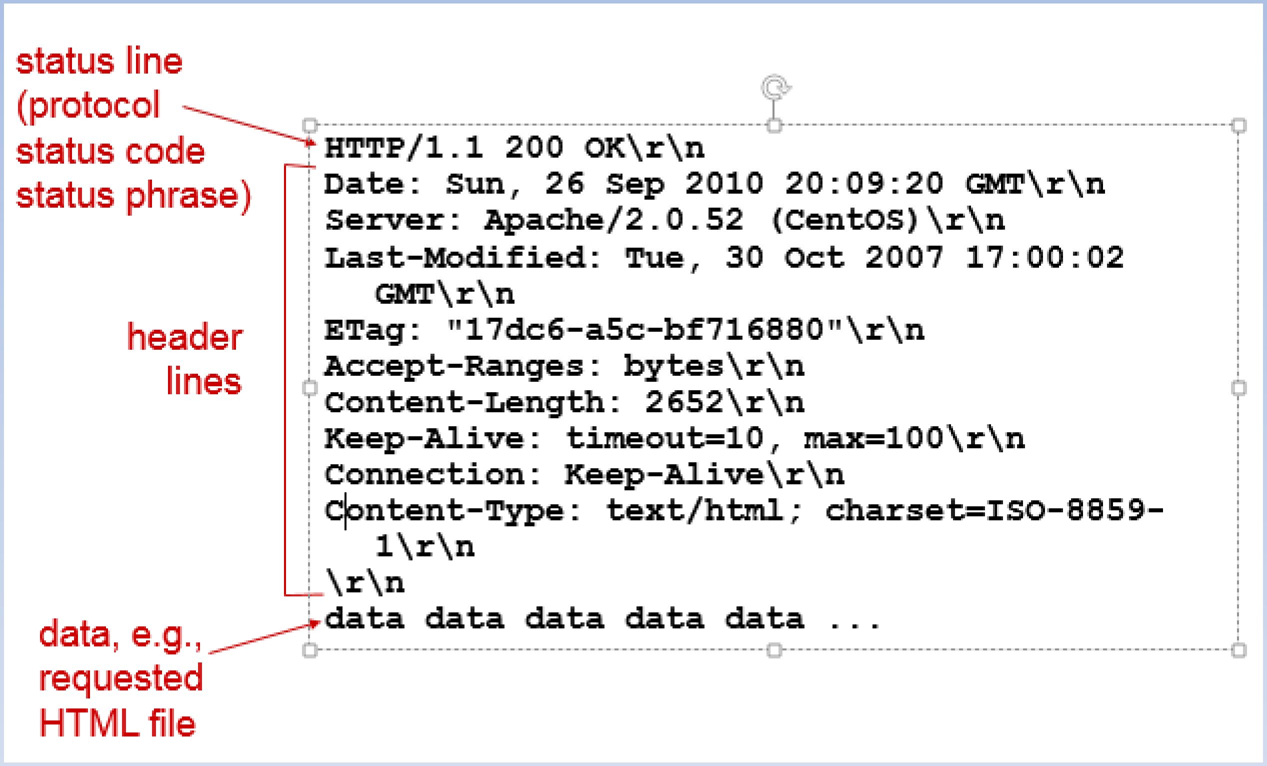
HTTP Request Message



HTTP Request

- Client’s router interprets 0s and 1s as a packet, and routes it across the internet
- Several intermediary servers later, it reaches the destination server
- Server reads the packet port destination and passes it to the appropriate application (web server application)
- Web server application receives the stream of data requesting for a specific file
- Web server locates the correct HTML file, bundles it into a packet to the client and sends it through is local router using the same transport mechanism

HTTP Response Message



```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

Mimicking the web browser

- So what does the browser do as far as these requests/responses are concerned?
  - Mainly instrumental in creating packets of information and sending them off
  - Interpreting the data coming back as pictures, sounds, videos and text
- Everything is handled by the HTTP protocol
- Can we mimic this action using Python?

BeautifulSoup

- It’s a Python library that makes sense of the nonsensical
- Installation instructions from crummy.com:

http://www.crummy.com/software/BeautifulSoup/

- Documentation @:

http://www.crummy.com/software/BeautifulSoup/bs4/doc/

- Optionally Python package manager *pip* can be installed and used to install any Python libraries

    (pip install beautifulsoup4)

- If python is in your Path, you can use it as a command explicitly from the Windows command prompt

Running BeautifulSoup

- Most common object used in the BeautifulSoup library is the BeautifulSoup object

```
import requests
from bs4 import BeautifulSoup
response = requests.get("http://www.unh.edu")
soup = BeautifulSoup(response.text,"lxml")
print(soup.title)
```

  HTML content is transformed into a BeautifulSoup object!

- The output is:

```
<title>University of New Hampshire</title>
```

Running BeautifulSoup

- Extracting header tags h1 through h6
- <h1> tag extracted is a few layers deep into the BeautifulSoup object structure
  - (html->body->h1)
  - But the “soup” object returned can use any tag directly
- Virtually any info can be extracted from any HTML (or XML) file, as long as it has some identifying tag surrounding it, or near it

Reliable Connections and Objects

- The web is a messy place to be!
  - Poorly formatted data
  - Unreliable web sites
    - Server is not found (“500 Internal Server Error”)
    - Page is not found (“404 Page Not Found”)
  - Closing tags are missing
    - Well formatted HTML is a discipline that needs to be followed by web developers
- You need to (and can) anticipate all of these exceptions by using Exception handlers

HTML Parsing

- There are several ways you can parse HTML
- lxml is the most feature-rich and easy-to-use library for processing XML and HTML in the Python language
- BeautifulSoup can use different HTML parsers and lxml is one of them, the combination of the two makes it very attractive