HyperText Transfer Protocol - HTTP

sockets

The network protocol that powers the web is actually quite simple and there is builtin support in Python called sockets which makes it very easy to make network connections and retrieve data over those sockets in a Python program. A *socket* is much like a file, except that a single socket provides a two-way connection between two programs. You can both read from and write to the same socket. If you write something to a socket, it is sent to the application at the other end of the socket. If you read from the socket, you are given the data which the other application has sent.

Importance of protocol

Iif you try to read a socket when the program on the other end of the socket has not sent any data, you just sit and wait. If the programs on both ends of the socket simply wait for some data without sending anything, they will wait for a very long time. So an important part of programs that communicate over the Internet is to have some sort of protocol. A protocol is a set of precise rules that determine who is
to go first, what they are to do, and then what the responses are to that message,and who sends next, and so on.

The World's Simplest Web Browser

Python program to show the working of HTTP protocol by making a connection to a web server and following the rules of the HTTP protocol to request a document and display what the server sends back.

```
import socket
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)
while True:
data = mysock.recv(20)
if (len(data) < 1):
break
print(data.decode(),end='')
mysock.close()
```

Explanation

First the program makes a connection to port 80 on the server www.py4e.com. Since our program is playing the role of the "web browser", the HTTP protocol says we must send the GET command followed by a blank line. Once we send that blank line, we write a loop that receives data in 512-character chunks from the socket and prints the data out until there is no more data to read (i.e., the recv() returns an empty string).

Output:

```
HTTP/1.1 200 OK
Date: Sun, 14 Mar 2010 23:52:41 GMT
Server: Apache
Last-Modified: Tue, 29 Dec 2009 01:31:22 GMT
ETag: "143c1b33-a7-4b395bea"
Accept-Ranges: bytes
Content-Length: 167
Connection: close
Content-Type: text/plain
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

The output starts with headers which the web server sends to describe the document. For example, the Content-Type header indicates that the document is aplain text document (text/plain). After the server sends us the headers, it adds a blank line to indicate the end of the headers, and then sends the actual data of the file romeo.txt.

Retrieving an image over HTTP

```
import socket
import time
HOST = 'data.pr4e.org'
PORT = 80
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((HOST, PORT))
mysock.sendall(b'GET http://data.pr4e.org/cover3.jpg HTTP/1.0\r\n\r\n')
count = 0
picture = b""
while True:
data = mysock.recv(5120)
if (len(data) < 1): break
time.sleep(0.25)
count = count + len(data)
print(len(data), count)
picture = picture + data
mysock.close()
# Look for the end of the header (2 CRLF)
pos = picture.find(b"\r\n\r\n")
print('Header length', pos)
print(picture[:pos].decode())
```

```
# Skip past the header and save the picture data
picture = picture[pos+4:]
fhand = open("stuff.jpg", "wb")
fhand.write(picture)
fhand.close()
```

output:

```
$ python urljpeg.py
2920 2920
1460 4380
...
2920 67160
1460 68620
1681 70301
Header length 240
HTTP/1.1 200 OK
Date: Sat, 02 Nov 2013 02:15:07 GMT
Server: Apache
Last-Modified: Sat, 02 Nov 2013 02:01:26 GMT
```

12.3. RETRIEVING AN IMAGE OVER HTTP 149

```
ETag: "19c141-111a9-4ea280f8354b8"
Accept-Ranges: bytes
Content-Length: 70057
Connection: close
Content-Type: image/jpeg
```

We can see that for this url, the Content-Type header indicates that body of the document is an image (image/jpeg). Once the program completes, you can view the image data by opening the file stuff.jpg in an image viewer. As the program runs, you can see that we don't get 5120 characters each time
we call the recv() method. We get as many characters as have been transferred across the network to us by the web server at the moment we call recv().We can slow down our successive recv() calls by uncommenting the call to time.sleep().

Retrieving web pages with urllib

While we can manually send and receive data over HTTP using the socket library,there is a much simpler way to perform this common task in Python by using the urllib library.Using urllib, you can treat a web page much like a file. You simply indicate which web page you would like to retrieve and urllib handles all of the HTTP protocol and header details.

The equivalent python code to read the romeo.txt file from the web using urllib is as follows:

```
import urllib.request
fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
```

```
print(line.decode().strip())
```

Once the web page has been opened with urllib.urlopen, we can treat it like a file and read through it using a for loop.When the program runs, we only see the output of the contents of the file. The
headers are still sent, but the urllib code consumes the headers and only returns the data to us.

Output:

But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief

Python program to write a program to retrieve the data for romeo.txt and compute the frequency of each word in the file as follows:

```
import urllib.request, urllib.parse, urllib.error
fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
counts = dict()
for line in fhand:
words = line.decode().split()
for word in words:
counts[word] = counts.get(word, 0) + 1
print(counts)
```

Parsing HTML and scraping the web

One of the common uses of the urllib capability in Python is to *scrape* the web. Web scraping is when we write a program that pretends to be a web browser and retrieves pages, then examines the data in those pages looking for patterns.
As an example, a search engine such as Google will look at the source of one web page and extract the links to other pages and retrieve those pages, extracting links, and so on. Using this technique, Google *spiders* its way through nearly all of the pages on the web. Google also uses the frequency of links from pages it finds to a particular page as one measure of how "important" a page is and how high the page should appear in its search results

Parsing HTML using regular expressions

One simple way to parse HTML is to use regular expressions to repeatedly search for and extract substrings that match a particular pattern.

Example:

```
<h1>The First Page</h1>
<p>
If you like, you can switch to the
```

```
<a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>.
</p>
```

We can construct a well-formed regular expression to match and extract the link values from the above text as follows: **href="http://.+?".**

Note: We add parentheses to our regular expression to indicate which part of our matched string we would like to extract.

Python program to*search* for lines that start with From and have an at sign

```
import urllib.request, urllib.parse, urllib.error
import re
url = input('Enter - ')
html = urllib.request.urlopen(url).read()
links = re.findall(b'href="(http://.*?)"', html)
for link in links:
print(link.decode())
```

The findall regular expression method will give us a list of all of the strings that match our regular expression, returning only the link text between the double quotes.

Output:

```
python urlregex.py
Enter - http://www.dr-chuck.com/page1.htm
http://www.dr-chuck.com/page2.htm
```

Parsing HTML using BeautifulSoup

BeautifulSoup tolerates highly flawed HTML and still lets you easily extract the data you need. We will use urllib to read the page and then use BeautifulSoup to extract the href attributes from the anchor (a) tags.

```
# To run this, you can install BeautifulSoup
# https://pypi.python.org/pypi/beautifulsoup4
# Or download the file
# http://www.py4e.com/code3/bs4.zip
# and unzip it in the same directory as this file
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl
# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE
url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).read()
```

```
soup = BeautifulSoup(html, 'html.parser')
# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
print(tag.get('href', None))
```

The program prompts for a web address, then opens the web page, reads the data and passes the data to the BeautifulSoup parser, and then retrieves all of the anchor tags and prints out the href attribute for each tag. When the program runs it looks as follows:

Output:

```
python urllinks.py
Enter - http://www.dr-chuck.com/page1.htm
http://www.dr-chuck.com/page2.htm
```

Python program to use BeautifulSoup to pull out various parts of each tag

```
# To run this, you can install BeautifulSoup
# https://pypi.python.org/pypi/beautifulsoup4
# Or download the file
# http://www.py4e.com/code3/bs4.zip
# and unzip it in the same directory as this file
from urllib.request import urlopen
from bs4 import BeautifulSoup
import ssl
# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE
url = input('Enter - ')
html = urlopen(url, context=ctx).read()
# html.parser is the HTML parser included in the standard Python 3 library.
# information on other HTML parsers is here:
# http://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser
soup = BeautifulSoup(html, "html.parser")
# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
# Look at the parts of a tag
print('TAG:', tag)
print('URL:', tag.get('href', None))
print('Contents:', tag.contents[0])
print('Attrs:', tag.attrs)
# Code: http://www.py4e.com/code3/urllink2.py
```

Output:

```
python urllink2.py
Enter - http://www.dr-chuck.com/page1.htm
TAG: <a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>
URL: http://www.dr-chuck.com/page2.htm
Content: ['\nSecond Page']
Attrs: [('href', 'http://www.dr-chuck.com/page2.htm')]
```

Reading binary files using urllib

Sometimes we want to retrieve a non-text (or binary) file such as an image or video file. The data in these files is generally not useful to print out, but you can easily make a copy of a URL to a local file on your hard disk using urllib. The pattern is to open the URL and use read to download the entire contents of
the document into a string variable (img) then write that information to a local file as follows:

```
import urllib.request, urllib.parse, urllib.error
img = urllib.request.urlopen('http://data.pr4e.org/cover3.jpg').read()
fhand = open('cover3.jpg', 'wb')
fhand.write(img)
fhand.close()
```

This program reads all of the data in at once across the network and stores it in the variable img in the main memory of your computer, then opens the file cover.jpg and writes the data out to your disk. This will work if the size of the file is less than the size of the memory of your computer.
However if this is a large audio or video file, this program may crash or at least run extremely slowly when your computer runs out of memory. In order to avoid running out of memory, we retrieve the data in blocks (or buffers) and then write each block to your disk before retrieving the next block. This way the program can read any size file without using up all of the memory you have in your computer.

```
import urllib.request, urllib.parse, urllib.error
img = urllib.request.urlopen('http://data.pr4e.org/cover3.jpg')
fhand = open('cover3.jpg', 'wb')
size = 0
while True:
info = img.read(100000)
if len(info) < 1: break
size = size + len(info)
fhand.write(info)
print(size, 'characters copied.')
fhand.close()
```

In this example, we read only 100,000 characters at a time and then write those characters to the cover.jpg file before retrieving the next 100,000 characters of
data from the web.

This program runs as follows:
python curl2.py
568248 characters copied.