Homework 3

Due March 3, 2017

Submissions are due by 11:59PM on the specified due date. Submissions may be made on the Blackboard course site under the Assignments tab. Late submissions will be accepted up to two days late with a 10% penalty for each day (24 hours).

Make sure your name and FSUID are in a comment at the top of the file.
In this assignment, you may only use the *socket* and *string* packages. The *__future__* and *copy* modules are always allowed. Please check with the instructor before using any other packages.

1 fibonacci.py (40 points)

Create a module named fibonacci.py, which defines the Fibonacci class. An instance of the Fibonacci class is instantiated with a single integer argument which determines the number of Fibonacci sequence numbers that should be stored in the list attribute *nums*. The Fibonacci sequence includes the numbers 0 and 1. Every subsequent term in the sequence is defined as the sum of the two previous terms. For example, the first ten terms of the Fibonacci sequence are:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

The Fibonacci class must define a *get_nums()* method which returns the *nums* attribute, a list of Fibonacci numbers. The Fibonacci class must also be defined such that it can be used in the context of a for-loop, as shown below, as well as with the print statement. Your output should match mine exactly.

```
>>> from fibonacci import Fibonacci
>>> f = Fibonacci(10)
>>> print f
The first 10 Fibonacci numbers are [0, 1, 1, 2, 3, 5, 8, 13, 21,
34]
>>> f.get_nums()
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
>>> for item in Fibonacci(8):
...     print item,
...
0 1 1 2 3 5 8 13
```

Now, in the same module but *not within the Fibonacci class*, define the generator fibonacci_gen() which accepts a single **optional** integer argument which determines the number of Fibonacci sequence numbers that should be generated. If the argument is not supplied, the generator should produce Fibonacci numbers indefinitely. Your generator should exhibit the following behavior, for example:

```
>>> from fibonacci import fibonacci_gen
>>> for i in fibonacci_gen(12):
...     print i,
...
0 1 1 2 3 5 8 13 21 34 55 89
```

Note that a crucial feature of generators is that they can perform lazy evaluation – only generating items in a sequence as they are needed instead of all upfront beforehand. You will receive 0 points for this part if your fibonacci_gen generator simply creates a static list of Fibonacci numbers and returns items from that list. It must generate the elements on the fly as they are needed.

2 browser.py (60 points)

The HTTP, or Hypertext Transfer protocol, is the application protocol used for communication on the World Wide Web. HTTP uses a client-server model, where HTTP requests may be sent by a client and HTTP responses may be sent back from the server. A request or response has the following format:

- An initial line,
- Zero or more header lines,
- A blank line
- An optional message body (e.g. a file, or query data, or query output).

Put another way, the format of an HTTP message is:

```
<initial line, different for request vs. response>
Header1: value1
Header2: value2
Header3: value3

<optional message body goes here, like file contents or query data;
 it can be many lines long>
```

An initial line for a **request** has three parts, separated by spaces: a method name (always uppercase), the local path of the requested resource, and the version of HTTP being used (always uppercase). A typical request line is:

```
GET /path/to/file/index.html HTTP/1.0
```

The methods we will support for our requests include the following:

- GET: most common HTTP method; it says "give me this resource". The Host header should specify the server host name.

- HEAD: asks the server to return the response headers only, and not the actual resource (i.e. no message body).

The initial line for a **response**, called the status line, also has three parts separated by spaces: the HTTP version (always uppercase), a response status code that gives the result of the request, and an English reason phrase describing the status code. Typical status lines are:

HTTP/1.0 200 OK

or

HTTP/1.0 404 Not Found

A complete list of status codes can be found here. Following the initial line in the response, there is a set of headers, followed by a blank line, followed by the content requested.

For example, if we wanted to request the contents of the course webpage, the GET request would look like this:

```
GET /~carnahan/cis4930/ HTTP/1.1
Host: www.cs.fsu.edu
```

Keep in mind that there is a blank line under the header list. The response we get back looks something like this:

```
HTTP/1.1 200 OK
Date: Tue, 14 Jun 2016 21:33:51 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Mon, 13 Jun 2016 10:53:16 GMT
ETag: "af0167-2816-53526b2105f00"
Accept-Ranges: bytes
Content-Length: 10262
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<head>
  <title>CIS4930</title>
…
```

The body of the message continues on to give us the entire contents of www.cs.fsu.edu/~carnahan/cis4930/index.html.

For a more detailed description, check out this website (which contains way more information than you'll need for this assignment): https://www.jmarshall.com/easy/http/

The Python standard library includes a handy module for creating http requests, but we will ***not*** be using it for this exercise – because we know how to make it ourselves! Your task is to create some of the basic functionality of the httplib module from scratch using raw sockets. Your module must define two classes:

- HTTPConnection(host[, port])
  - A class representing a connection with an HTTP server.
  - Must be instantiated with a hostname, and optionally a port number, which defaults to 80.

- HTTPResponse(sock)
  - an instance of this class represents a response returned from an HTTP server.
  - The sock argument is the socket from which a response should be read and parsed.

Start by creating the definitions of your classes and their initialization methods. Once you've done that, add support for the following methods to the HTTPConnection class:

- HTTPConnection.connect()
  Connect to the server specified when the object was created.

- HTTPConnection.request(method, url [, headers])
  This will send a request to the server using the HTTP request method specified and the selector url. The headers argument should be a mapping of extra HTTP headers to send with the request. The HTTP version used for all requests should be HTTP/1.1.

- HTTPConnection.getresponse()
  Should be called after a request is sent to get the response from the server. Returns an HTTPResponse instance.

- HTTP.Connection.close()
  Closes the connection to the server.

Now, add the following methods and attributes to the HTTPResponse class:

- HTTPResponse.read([*amt*])
  Reads and returns the response body, or up to the next *amt* bytes. Once the entire response body has been read, subsequent calls to read() should return an empty string.

- HTTPResponse.getheader(*name*)
  Get the contents of the header *name* or return None if the header name is not in the header list of the response.

- HTTPResponse.getheaders()
  Return a list of (header, value) tuples. Return an empty list if no headers are included.

- HTTPResponse.version
  HTTP protocol version used by server. "1.0" for HTTP/1.0, "1.1" for HTTP/1.1.

- HTTPResponse.status
  Status code returned by server.

- HTTPResponse.reason
  Reason phrase returned by server.

A testing file browser_test.py and an example output from this test file, browser_test_output.txt is included on the course website to help you test your module. However, you are strongly encouraged to come up with more test cases to ensure the robustness of your module. If you need some inspiration, open up a developer console on your browser (on Firefox, go to Developer > Web Console in the main menu and choose the Network tab). This will allow you to see what http requests and responses are being generated as your browse websites.

Suggestions:

- **You are required to only use sockets to communicate with the server**. You may find that using the makefile() method on the socket object makes it slightly easier to read the response.