Lab 1

Purpose:

- To do low-level socket programming on a standard high-level application protocol (`HTTP`)
- To use a socket-level service (`html2text`)

Handin:

You are to hand in the following file to the Lab 1 Dropbox folder on D2L:

- `client.py`

Getting Started

This is the first time you'll be working directly with sockets in Python. You need to do some basic research and testing to get used to how to work with these specialized objects. Here are some things you should read up on before you really get started with this lab:

- Python 3 socket programming:
    - Official docs of the socket module: https://docs.python.org/3/library/socket.html
    - socket constructor, socket.connect(), socket.send() and socket.recv()
- You are writing "client" software in this lab. Consider googling "Python3 socket client" and reviewing some examples.
- You are implementing the "HTTP 1.1" for the first part of this lab; you will not be given the details of this protocol -- you have to research it and try it out yourself!
    - This might help. Telnet is a program that lets you talk to text-based servers (web servers are text-based servers, for example). We don't care about Telnet itself, but it shows you the way to "talk" to the server: https://stackoverflow.com/questions/15772355/how-to-send-an-http-request-using-telnet

Specification:

You are to write a client application that 1/ retrieves HTML text from a user-requested website, and 2/ displays that content in the terminal as formatted ASCII text.

NOTE: All text transfers over the network are assumed to be UTF-8 encodings. This applies to both the web server and the html2text server.

Although there are libraries in python that will retrieve HTML content from websites for you, for this lab you must connect to the web server directly using sockets and navigate the `HTTP` protocol manually.

The user will specify a full URL on the commandline. The URL will be the only commandline parameter, as in:

```
python client.py http://rtvm.cs.camosun.bc.ca/ics226/lab1test1.html
```

You may assume that the user starts the URL with "`http://`". All you have to do, then, is break up the rest of the URL, as in:

- `host = "rtvm.cs.camosun.bc.ca"`
- `resource = "/ics226/lab1test1.html"`

Of course, you have to support all "`http`" URL's, not just the one above! And note that even if the user doesn't specify an actual "resource" -- as in "`http://google.com`" -- you need to add the implied resource "`/`".

Once you have separated the host and the resource, you can connect to the host's port 80, and issue the proper `HTTP` (use version 1.1) request for the resource. (Try doing a search for "telnet HTTP to get some ideas of how to put together the proper commands.)

You do not have to do any error handling or manage redirect requests or anything like that! If the web server sends invalid HTML (that is, no <HTML> or </HTML> tags), you don't have to do any special processing or response.

Once you've read in some HTML content (that is, something after "`<HTML>`" and before "`</HTML>`" tags), you can send to the `html2text` service.

- `host = rtvm.cs.camosun.bc.ca`
- `port = 10010`

Note that this host is multi-threaded, and may put your client in a request queue, so don't assume it's not working if it doesn't respond right away!

When the `html2text` server is ready to process your request, it will send you a `READY` message. When you receive that text, go ahead and send the `HTML` content to the server.

The server will assume the content is complete once it sees a "`</HTML>`" tag (case insensitive). At that point it will do the `html -> text` conversion, and start sending back a data stream with the resulting content.

You must receive that converted content in blocks, until you see the text:

`ICS 226 HTML CONVERT COMPLETE`

Everything the server sent prior to this text is part of the converted content; once you receive this final text, the server will wait for you to send a final "`OK`", and then both client and server sockets will be closed.

Note that if your client takes more than 3 seconds to respond to any of the server's `recv()` commands, the server will simply terminate your connection and move on without necessarily sending any useful content back to you.

Sending And Receiving In Blocks

An important requirement for this lab is that you do not store the entire content -- either the HTML code or the formatted text result -- in memory at once. Therefore, you are required to send and receive in block sizes of 1024 bytes (or less).

This makes things a little bit tricky, as you will have to be receiving from the web server *at the same time as you are sending to the html->text converter*. You also have to print blocks out as you go, being careful not to print the final `ICS 226 HTML CONVERT COMPLETE` message.

It is fine to use whatever logic you like to make this work. It is suggested you create a "state" variable to keep track of what you're trying to do at the moment.

Your client logic could look something like this:

```
s1.connect( (webhost, 80) )
# navigate HTTP protocol until the resource is about to be transferred

s2.connect( (html2textHost, 10010) )
# navigate the html2text protocol until the server is expecting bytes of
# HTML code.

# now start reading and writing.
state = 1
while state != 4:
       # if state == 1 - read in headers from web server, basically ignoring,
       #                 looking for <HTML>.
       #               - when <HTML> is read, move to state = 2
       #
       # if state == 2 - read in block from web server, and sending to html2text
server
       #               - when </HTML> is read, move to state = 3
       #
```

```
        # if state == 3 - receive from html2text server, printing received block
        #                 as you go.
        #               - when ICS 226 HTML CONVERT COMPLETE is received, move to
        #                 state = 4

# Finish up
# - print any remaining output blocks
# - close the two sockets
```

You have to be especially careful when dealing with the transitions from one state to the next. You have to make sure:

- You send `<HTML>` and `</HTML>` tags to the `html2text` server, and everything between, but nothing outside of these tags.
- You print the result from `html2text` server, but not the `ICS 226 HTML CONVERT COMPLETE` ending tag.

A complete solution will take into account the fact that the "sentinels" (the text items you're looking for to move from state to state) could be broken up between blocks; also, `HTML` tags may be in any case (even mixed cases in a single tag! Yikes!). It is recommended that you program *first* without worrying about these complications; then try to tackle them after you have the main program working.

The basic strategy for dealing with broken-up sentinels is to always keep track of the *current* block and also the *previous* block. So you might have something like this:

```
if "<HTML>" in (previousBlock + currentBlock):
      # what now?
      # ...
previousBlock = currentBlock;
```

This complicates things a fair bit, and can make certain parts of the code pretty tricky! Make sure you save a version of your file that works without taking this into account, in case you don't get to finish it!

To handle issues of upper/lower-case matches, just convert the target string to `upper()` (or `lower()`), as in:

```
if "<HTML>" in block.upper():
      # // what now?
```

Be careful not to modify the original page text before sending it to the `html2text` server, or before printing it for the user! Just do the `.upper()` in the `if` statement for the test!

Sample Test Pages

There are a few pages provided for you that can test different aspects of your client program:

- http://rtvm.cs.camosun.bc.ca/ics226/lab1test1.html
- http://rtvm.cs.camosun.bc.ca/ics226/lab1test2.html
- http://rtvm.cs.camosun.bc.ca/ics226/lab1test3.html
- http://rtvm.cs.camosun.bc.ca/ics226/lab1test4.html