HTTP, HTML, and related practice exercises

My web server is running two pages for these exercises:

http://garrod.isri.cmu.edu/webapps/methods/check-get
expects two GET parameters: **andrewid** and **color**

http://garrod.isri.cmu.edu/webapps/methods/check-post
expects two POST parameters: **andrewid** and **color**

1. Write an HTML5 page containing two forms that send appropriate requests to check-get and check-post, respectively.

2. Validate your HTML5 page from part 1 using the 'Validate by File Upload' tab on http://validator.w3.org/. What detail did you miss? It's missing from my examples. What is the relevance of the detail you missed?

3. Use telnet to manually generate a valid HTTP version 1.1 request to check-get. (What header line is required, and why? What happens if you don't provide the required header line?) Hint: Many telnet tutorials are available online. Use telnet on **unix.andrew.cmu.edu** if your laptop does not include telnet.

4. Use telnet to manually generate a valid HTTP version 1.1 request to check-post. Warning! This is a bit tricky, we didn't give you an explicit example of this, and real web servers do not provide good feedback. You might need a bit of research from the internet or the Chrome Developer Tools to figure this out. Rhetorical but highly relevant questions: What header lines are required, and why? What is a MIME type, and how is that relevant to an HTTP POST request? How is POST form data encoded?

content-type
content-length
query string: parameter=value&also=another

```
POST /path/script.cgi HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

home=Cosby&favorite+flavor=flies
```

When receiving a POST request, you should always expect a "payload", or, in HTTP terms: a message body. The message body in itself is pretty useless, as there is no standard (as far as I can tell. Maybe application/octet-stream?) format. The body format is defined by the Content-Type header. When using a HTML FORM element with method="POST", this is usually application/x-www-form-urlencoded.

Generally the Content-Length header is used for HTTP 1.1 so that the receiving party knows when the current response* has finished, so the connection can be reused for another request.

For application/x-www-form-urlencoded, the body of the HTTP message sent to the server is essentially one giant query string -- name/value pairs are separated by the ampersand (&), and names are separated from values by the equal symbal (=).

non-alphanumeric characters are replaced by `%HH', a percent sign and two hexadecimal digits representing the ASCII code of the character