

**Web programming**  
**Programming Course for Biologists at the**  
**Pasteur Institute**  
**Practical Work**

by Catherine Letondal

# **Web programming: Programming Course for Biologists at the Pasteur Institute: Practical Work**

by Catherine Letondal

Published February, 28 2005

Copyright © 2005 Pasteur Institute [<http://www.pasteur.fr/>]

This material consists in exercises to help students understand basic and fundamental mechanisms of the Web: what is a client, and what is the role of a client? what is a server? What is a protocol? How to understand main differences between various architectures and tools? These exercises are not intended *at all* for future Webmasters.

PDF version of this course [[support.pdf](#)]

*This course is still under construction.* Comments are welcome.

Contact: [ieb@pasteur.fr](mailto:ieb@pasteur.fr)

# Table of Contents

1. HTTP .....	1
1.1. Interactive telnet session with a Web server .....	1
1.2. A tiny Web server to play with .....	5
1.3. A tiny Web client to play with .....	7
2. CGI .....	11
2.1. CGI scripts environment .....	11
2.2. A stupid one-purpose Web server .....	12
2.3. CGI scripts that do something useful .....	14
2.4. Debugging CGI scripts .....	16
2.5. Controls to perform in a CGI script .....	18
2.6. CGI for helicasas .....	20



## List of Exercises

1.1. Fetch a document	1
1.2. HTTP headers	2
1.3. HTTP status code	3
1.4. HTTP GET: dynamic content	4
1.5. HTTP POST	4
1.6. The Web server code	5
1.7. Using our server	6
1.8. Using our server with a standard Web browser	7
1.9. The Web client code	7
1.10. Use the Web client	9
1.11. Enhance the Web client	9
2.1. Installing a CGI program	11
2.2. A simple CGI program called from a form	11
2.3. MIME type	12
2.4. CGI environment	12
2.5. A Web server that just returns a sequence	13
2.6. A CGI to fetch a sequence with the golden program	14
2.7. Testing input	15
2.8. Getting raw input	16
2.9. Tracing	16
2.10. Redirecting standard error to the browser	17
2.11. Debugging your script directly	17
2.12. Testing for unchecked radio button	19
2.13. Testing for improper user input	19
2.14. Testing for improper user input (continued)	20
2.15. Submit an helicase	20
2.16. Use a persistent Helicase DB	21
2.17. Use a persistent Helicase DB (2)	21
2.18. Load the database	21
2.19. Fetch an helicase	22
2.20. Browse helicases	22
2.21. Search helicases by keyword	22



## Chapter 1. HTTP

This practical session will enable you to explore important concepts regarding the Web:

- The HTTP protocol (Section 1.1).
- The role of the server (Section 1.2).
- The role of the client (Section 1.3).

### 1.1. Interactive telnet session with a Web server

We will first explore the HTTP protocol by using a good old program called **telnet**, that can connect on any port service by just providing the port number as an argument on the command line. This will let us see what is normally hidden by a standard Web browser: actual request and server responses, before being handled by a graphical end-user browser.

#### Exercise 1.1. Fetch a document

Let us fetch a document from the Pasteur Institute Web server:

```
% telnet www.pasteur.fr 80
Trying 157.99.64.12...
Connected to www.pasteur.fr.
Escape character is '^]'.
GET /formation/infobio/web/cours/data/page1.html HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 24 Feb 2004 18:01:05 GMT
Server: Apache/1.3.26 (Unix) mod_perl/1.24_01 mod_ssl/2.8.10 OpenSSL/0.9.5a
Last-Modified: Tue, 18 Feb 2003 13:40:14 GMT
ETag: "101e6a1-cd-3e5237be"
Accept-Ranges: bytes
Content-Length: 205
Connection: close
Content-Type: text/html; charset=iso-8859-1

<html>
  <head>
    <title>A sample Web page</title>
  </head>

  <body>
    <h1>A first header</h1>
    And some text...
```

```

</body>
</html>
Connection closed by foreign host.
%
```

You can as well get important information from the Web server, before getting the entire document:

```

% telnet www.pasteur.fr 80
Trying 157.99.64.12...
Connected to www.pasteur.fr.
Escape character is '^]'.
HEAD /formation/infobio/web/cours/data/page1.html HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 24 Feb 2004 18:01:05 GMT
Server: Apache/1.3.26 (Unix) mod_perl/1.24_01 mod_ssl/2.8.10 OpenSSL/0.9.5a
Last-Modified: Tue, 18 Feb 2003 14:38:31 GMT
ETag: "101e6a1-cd-3e5237be"
Accept-Ranges: bytes
Content-Length: 205
Connection: close
Content-Type: text/html; charset=iso-8859-1

Connection closed by foreign host.
%
```

What could this kind of information be useful for?



## Exercise 1.2. HTTP headers

Let us use date information from the server. For instance, say you do not want a too recently modified file: you can use an HTTP header for this purpose.

```

% telnet www.pasteur.fr 80
Trying 157.99.64.12...
Connected to www.pasteur.fr.
Escape character is '^]'.
GET /formation/infobio/web/cours/data/page1.html HTTP/1.0
If-Modified-Since: Tue, 18 Feb 2003 14:38:31 GMT

HTTP/1.1 304 Not Modified
Date: Tue, 24 Feb 2004 18:01:05 GMT
Server: Apache/1.3.26 (Unix) mod_perl/1.24_01 mod_ssl/2.8.10 OpenSSL/0.9.5a
Last-Modified: Tue, 18 Feb 2003 14:38:31 GMT
Connection: close
ETag: "101e6a1-cd-3e5237be"
```

## Chapter 1. HTTP

Connection closed by foreign host.

Another useful header let you chain several requests. Try:

```
% telnet www.pasteur.fr 80
Trying 157.99.64.12...
Connected to www.pasteur.fr.
Escape character is '^]'.
GET /formation/infobio/web/cours/data/page1.html HTTP/1.0
Connection: keep-alive
```

What can you do after server's response?



### Exercise 1.3. HTTP status code

HTTP return codes help you know whether a request has been successful. Explain what happens in the following:

```
% telnet bioweb.pasteur.fr 80
Trying 157.99.64.11...
Connected to rosalind.sis.pasteur.fr.
Escape character is '^]'.
GET /formation/infobio/web/cours/data/page1.html HTTP/1.0

HTTP/1.1 404 Not Found
Date: Tue, 24 Feb 2004 18:01:05 GMT
Server: Apache/1.3.20 (Unix)
Last-Modified: Tue, 18 Feb 2003 14:38:31 GMT
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /formation/infobio/web/cours/data/page1.html was not found on this server.<P>

<HR>
<ADDRESS>Apache/1.3.20 Server at bioweb.pasteur.fr Port 80</ADDRESS>
</BODY></HTML>
Connection closed by foreign host.
%
```

And here:

```
% telnet www.pasteur.fr 80
Trying 157.99.64.12...
Connected to www.pasteur.fr.
Escape character is '^]'.
HEAD /formation/infobio/web/cours/data/page2.html HTTP/1.0

HTTP/1.1 403 Forbidden
Date: Tue, 24 Feb 2004 18:01:05 GMT
Server: Apache/1.3.26 (Unix) mod_perl/1.24_01 mod_ssl/2.8.10 OpenSSL/0.9.5a
Last-Modified: Tue, 26 Nov 2002 15:21:19 GMT
ETag: "ee325-c4c-3de3916f"
Accept-Ranges: bytes
Content-Length: 3148
Connection: close
Content-Type: text/html; charset=iso-8859-1

Connection closed by foreign host.
```

### Exercise 1.4. HTTP GET: dynamic content

A Web server does not only serve static document. You can also issue request to get dynamically computed document. At the HTTP protocol level, there are several ways to achieve this. Firstly, try:

```
% telnet www.pasteur.fr 80
Trying 157.99.64.12...
Connected to www.pasteur.fr.
Escape character is '^]'.
GET /cgi-bin/biology/bnb_s.pl?query=biopython HTTP/1.0
```

what do you get?

Try another one (put your own email in place of YOUR\_EMAIL):

```
% telnet bioweb.pasteur.fr 80
Trying 157.99.64.11...
Connected to rosalind.sis.pasteur.fr.
Escape character is '^]'.
GET /cgi-bin/seqanal/pdbsearch.pl?email=YOUR_EMAIL&query=1crn HTTP/1.0
```

### Exercise 1.5. HTTP POST

Describe the difference between Exercise 1.4 and the following (do not forget to reset the

`<variable>Content-length</variable>`

according to the total length of the request character string, e.g: ):

```
% telnet bioweb.pasteur.fr 80
Trying 157.99.64.11...
Connected to rosalind.sis.pasteur.fr.
Escape character is '^]'.
POST /cgi-bin/seqanal/pdbsearch.pl HTTP/1.0
Content-type: application/x-www-form-urlencoded
Content-length: 35

email=YOUR_EMAIL&query=1crn
```

## 1.2. A tiny Web server to play with

Now that we have played with the HTTP protocol, e.g by issuing requests and specifying HTTP headers, let us see how a Web server work by programming a small server, that is only able to handle **GET** requests.

### Exercise 1.6. The Web server code

```
from BaseHTTPServer import HTTPServer
from BaseHTTPServer import BaseHTTPRequestHandler

class myHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        self.printCustomHTTPResponse(200)
        self.wfile.write("<html>\n<body>\n")
        self.wfile.write("<p>GET string: " + self.path + "</p>" )
        self.printBrowserHeaders()
        self.wfile.write("</body>\n</html>\n")

    def printBrowserHeaders(self):
        keys = self.headers.dict.keys()
        self.wfile.write("\n<ul>")
        for key in keys:
            self.wfile.write("\n<li><b>" + key + "</b>: ")
            self.wfile.write(self.headers.dict[key] + "\n</li>\n")
        self.wfile.write("</ul>\n")

    def printCustomHTTPResponse(self, respcode):
        self.send_response(respcode)
        self.send_header("Content-Type", "text/html")
```

```

        self.send_header("Server", "myHandler")
        self.end_headers()

    def log_request(self, code='-', size='-'):
        user_agent = self.headers.dict['user-agent']
        self.log_message('%s %s %s %s',
                        self.requestline, str(code), str(size), user_agent)

if __name__ == "__main__":
    server = HTTPServer(("",2122), myHandler)
    for lp in range(5000):
        server.handle_request()

```

- ❶ This is the method that is called when a **GET** is issued.
- ❷ This method displays the headers that have been set by the client.
- ❸ This method set the server response headers.
- ❹ Run the server: provide the `myHandler` class as your custom requests handler, and tell that you want to listen on port 2122.

We want to run this server of course. From the Unix prompt, run:

```
% python tiny_server.py
```

and let the program wait for client requests (requests will be displayed on the terminal).



### Exercise 1.7. Using our server

Now, we can request our server (defined and run as shown in Exercise 1.6) with ... a telnet interactive session as in the first exercises of this practical session.

```

% telnet feu.sis.pasteur.fr 2122
Trying 157.99.60.151...
Connected to feu.sis.pasteur.fr.
Escape character is '^]'.
GET tralalaitou HTTP/1.0
User-Agent: telnet + my fingers
Connection: keep-alive

HTTP/1.0 200 OK
Server: BaseHTTP/0.2 Python/2.2
Date: Tue, 18 Feb 2003 17:15:49 GMT
Content-Type: text/html

```

```
Server: myHandler

<html>
<body>
<p>GET string: tralalaitou</p>
<ul>
<li><b>connection</b>: keep-alive
</li>
<li><b>user-agent</b>: telnet + my fingers
</li>
</ul>
</body>
</html>
Connection closed by foreign host.
%
```

### Exercise 1.8. Using our server with a standard Web browser

Now, we can of course also request our simplistic server with a proper Web browser. First check that the server is running, and launch a browser on the following url: <http://feu.sis.pasteur.fr:2122/tralalilaere>.

## 1.3. A tiny Web client to play with

And, at last, we have to explore the client side. Instead of using **telnet** or a standard Web browser, we will write our own Web client. The two main tasks of a Web client are:

- To issue requests.
- To display results. This is the most difficult part of the client, and our simple client will just display raw HTML without any formatting in a text window.

## Exercise 1.9. The Web client code

```

import browser ❶
from httpplib import HTTP ❷
import sys

from sys import argv
if len(argv) > 1:
    host=argv[1]
if len(argv) > 2:
    port=argv[2]
else:
    port = '80'
if len(argv) > 3:
    document = argv[3]
else:
    document = '/'

print host+':' + port + ' ' + document

def fetch(host, document, port=None): ❸
    if port:
        req = HTTP(host + ':' + port)
    else: ❹
        req = HTTP(host)

    req.putrequest("GET", document) ❺

    req.putheader("Accept", "text/html") ❻
    req.putheader("User-Agent", sys.argv[0])
    req.endheaders()

    status_code, status_phrase, headers = req.getreply() ❼
    f=req.getfile() ❽
    result=f.read() ❾
    f.close() ❿
    return result, str(status_code) + ' ' + status_phrase

text, status = fetch(host, document, port)
b = browser.Browser(fetch)
url = "http://" + host + ":" + port + document ⓫
b.display(text, url, status)

```

❶ browser is a just module defining a class to display text (see code/browser.py).

- ② `httplib` is a module defining - among others things - a class (`HTTP`) to issue `http` requests.
- ③ A function to fetch document.
- ④ Instantiation of `HTTP` class: this creates a client, or an agent to issue requests.
- ⑤ Put the request.
- ⑥ Write headers.
- ⑦ Actually send the request with headers.
- ⑧ Print `HTTP` status code and message.
- ⑨ Get the actual requested document.
- ⑩ Display it in the browser. Note that the `fetch` function is passed to the browser at instantiation.

### Exercise 1.10. Use the Web client

Use the client script defined in Exercise 1.9 to request the server defined in Exercise 1.6:

```
% python -i tiny_client.py feu.sis.pasteur.fr 2122 tralala
```

Then Use this client script to get the home page of the Pasteur Institute Web server: what happens?

```
% python -i tiny_client.py www.pasteur.fr
```

Issue a more precise request, such as:

```
% python -i tiny_client.py www.pasteur.fr 80 /formation/infobio/
```

Is that better? It seems OK, but it is still non-handled HTML. So, what about asking for a non-HTML document:

```
% python -i tiny_client.py www.pasteur.fr 80 /formation/infobio/2003/python/solutions/sol_cod
```

### Exercise 1.11. Enhance the Web client

Add a field displaying the date of the fetched document to the browser (defined in Exercise 1.9). The date of the document lies in the last-modified header. For this purpose, you will have to:

- add a `Label` field to the browser,
- change the `fetch` function to return an additional information (the `last-modified` header); for instance, use the `dir()` to look at the returned `headers` object in order to find out this header.



## Chapter 2. CGI

### 2.1. CGI scripts environment

#### Exercise 2.1. Installing a CGI program

Put the following code in a file called `hello.py` and install it in your `cgi-bin` directory (`/usr/lib/cgi-bin/you`).

```
#!/local/bin/python
print "Content-type: text/plain"
print
print "Hello, world"
```

Check that your file has the appropriate permissions.

Once installed, try it with the proper url, e.g: <http://leeloo.sis.pasteur.fr/cgi-bin/IEB/hello.py>.

#### Exercise 2.2. A simple CGI program called from a form

We want to write a CGI script to handle the `cgi/form1.html` form seen during the HTML course. Put the following code in a file called `form1.py` and install it in your `cgi-bin` directory. This script uses the `cgi` Python module.

```
#!/local/bin/python
import cgi

# get input
query = cgi.parse()

# document returned
print "Content-type: text/html"
print
print '<html>\n'

print "<head>\n"
print "<title>Form1</title>\n"
print "</HEAD>\n"

print "<body>\n"
print "<p>seq_type: ", query['seq_type'], "</p>\n"
print "<p>seq_name: ", query['seq_name'], "</p>\n"
print "<p>seq: ", query['seq'], "</p>\n";
print "</body>\n"
```

```
print "</html>\n"
```

Copy the form in your own documents directory on the Web server and click on submit.



### Exercise 2.3. MIME type

What's wrong with the following CGI (saved it in `hello_html.py`):

```
#!/local/bin/python
print "Content-type: text/plain"
print
print "<html>\n"

print "<head>\n"
print "<title>Hello, world</title>\n"
print "</head>\n"

print "<body>\n"
print "<h1>Hello, world</h1>\n"
print "Hello, world"
print "</body>\n"

print "</html>\n"
```

- Try it: [http://leeloo.sis.pasteur.fr/cgi-bin/IEB/hello\\_html.py](http://leeloo.sis.pasteur.fr/cgi-bin/IEB/hello_html.py).
- Also try the fixed script: [http://leeloo.sis.pasteur.fr/cgi-bin/IEB/hello\\_html\\_fixed.py](http://leeloo.sis.pasteur.fr/cgi-bin/IEB/hello_html_fixed.py).

### Exercise 2.4. CGI environment

Write a CGI, `print_env.py`, that prints some of the CGI environment variables:

- `PATH`
- `REMOTE_ADDR`
- `CONTENT_LENGTH`
- `REQUEST_URI`
- `QUERY_STRING`

Try it: `http://leeloo.sis.pasteur.fr/cgi-bin/IEB/print_env.py`. How do you specify something to get it in the `QUERY_STRING` environment variable?

Then write another variant that prints all the environment variables. Try it: `http://leeloo.sis.pasteur.fr/cgi-bin/IEB/print_all_env.py`.

## 2.2. A stupid one-purpose Web server

### Exercise 2.5. A Web server that just returns a sequence

The following Web server just returns a FASTA formatted sequence, according to a given sequence identifier.

```
import os
import string

from BaseHTTPServer import HTTPServer
from BaseHTTPServer import BaseHTTPRequestHandler

class SeqHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        seq_id = self.path
        result = self.get_seq(seq_id)
        if result is not None:
            self.printCustomHTTPResponse(200)
        else:
            self.printCustomHTTPResponse(404)
        self.wfile.write("<html>\n<body>\n")
        self.wfile.write("<p>Request: " + seq_id + "</p>\n" )
        if result is not None:
            self.wfile.write("<pre>\n" )
```

```

        self.wfile.write(result)
        self.wfile.write("</pre>\n" )
self.wfile.write("\n</body>\n</html>\n")

def printCustomHTTPResponse(self, respcode):
    self.send_response(respcode)
    self.send_header("Content-Type", "text/html")
    self.send_header("Server", "myHandler")
    self.end_headers()

def get_seq(self, seq_id, db='sp'):
    os.environ['PATH'] = '/usr/local/gensoft/bin' + ':' + os.environ['PATH']
    cmd="golden " + db + ":" + seq_id
    handle = os.popen(cmd, 'r')
    entry = handle.readlines()
    status = handle.close()
    if status is not None:
        return None
    return string.join(entry, ")

if __name__ == "__main__":
    server = HTTPServer(("",2113), SeqHandler)
    for lp in range(5000):
        server.handle_request()

```

How do you use this Web server?

How do you call the sequence identifier in Web parlance? Which standard part of the request does it represent?

What do you think about this server? What is generally the role of a Web server?

## 2.3. CGI scripts that do something useful



### Exercise 2.6. A CGI to fetch a sequence with the golden program

Write a CGI script called `fetch_seq.py` that fetches a sequence given its sequence identifier provided in the query part of the url, e.g: `http://leeloo.sis.pasteur.fr/cgi-bin/IEB/fetch_seq.py?seq_id=MALK_ECOLI`. This script should return the entry as plain text, without any HTML formatting.

For this purpose, you might need to write a `get_seq` function in your `sequences` module, such as:

```
import commands
```

## Chapter 2. CGI

```
def get_seq (id, db='sp'):
    cmd="golden " + db + ":" + id
    status, output = commands.getstatusoutput(cmd)
    if status != 0:
        return status, "A problem occurred: " + output + '\nstatus: ' + str(status) + '\n'

    return None, output
```

You can use this function from within another `get_seq` function in your CGI:

```
def get_seq (id, db='sp'):
    os.environ['PATH'] = '/usr/local/bin:' + os.environ['PATH']
    os.environ['GOLDENDATA'] = '/local/cours/share/golden'
    status, output = sequences.get_seq(id=id, db=db)
    if status is None:
        return output
    return "A problem occurred: " + output + '\nstatus: ' + str(status) + '\n'
```

Also check that the `PATH` to find the **golden** command: it is not necessarily present in the Web server `PATH` (see Exercise 2.4).

To let you handle correctly the query string, the `cgi` module provides a `parse` function that returns a dictionary. Each entry of this dictionary contains a *list* of values associated with a given field. So, given the above url, the dictionary will have an entry for `'seq_id'`, containing:

```
[ 'MALK_ECOLI' ]
```

When would you have more than one value in this list? Guess the url that would leads to:

```
[ '100K_RAT', 'MALK_ECOLI' ]
```

Write the form to invoke this CGI and call it: `fetch_seq_post.html`.

### Exercise 2.7. Testing input

Write the `control_ok` function, a function that controls an input field. An input field should contain only letters, digits, and the following characters: `, - _`. The dot (`.`) is authorized, provided there is only one (e.g: `..` is not allowed).

```
seq_id = query['seq_id'][0]
if control_ok(seq_id):
    print get_seq(seq_id)
else:
```

```
print "Your input is not valid. Please provide a correct entry_name"
print
```

solution [cgi/fetch\_seq\_ctl.py]

## 2.4. Debugging CGI scripts

### Exercise 2.8. Getting raw input

Write a script called `debug CGI .py` that prints the input as received by the script:

```
#!/local/bin/python
import sys
input=sys.stdin.readlines()
print "Content-type: text/plain"
print
print input
```

❶  
❷

❶ Get data from standard input.

❷ Specifying a `text/plain` content type let you get the raw input string.

Call this script from a form (`debug CGI .html`). This form should have at least two input fields (one entry, called `first`, and one checkbox, called `second`).

Does this script use the `cgi` module? Would it be possible to get the standard input while using this module?

What do you conclude about the way for a Web server to provide input to a CGI script (in case of a **POST**)?

### Exercise 2.9. Tracing

You can put print statements that will display trace information in the server log file (for instance `/var/log/apache/error.log`), by writing on the `sys.stderr` file handle:

```
#!/local/bin/python
import sys

query = cgi.parse()
print >>sys.stderr, "(debug from " + sys.argv[0] + ") second field: ", query['second']

print "Content-type: text/plain"
print

print "First field is: ", query['first']
```

- ❶ Trace statement: print data on standard error, without disturbing the normal output. Notice that the statement refers to the name of the script, which can help if there are several CGI scripts trace statements intertwined in the server log file.

### Exercise 2.10. Redirecting standard error to the browser

Another very useful feature is to be able to access Python error messages directly on the CGI output, instead of the server log file. The way to achieve this is to redirect `sys.stderr` to `sys.stdout`:

```
#!/local/bin/python
import sys
sys.stderr = sys.stdout

print "Content-type: text/plain"
print
# the following statement will produce an error
print input
```

- ❶ Errors normally displayed in the log file will be displayed in the client browser, because this statement changes the `sys.stderr` value to be the same as `sys.stdout`.

### Exercise 2.11. Debugging your script directly

An easy way to debug your script can be to run it directly from the Unix prompt:

```
% ./test.py
```

but in this case, you need to simulate input from the Web server (and you will have many environment variables unset). One way to achieve this can be:

```
% QUERY_STRING='first=val1&second=val2' ./test.py
```

## 2.5. Controls to perform in a CGI script

### Exercise 2.12. Testing for unchecked radio button

The following script tests for the value of a radio-button. What will happen when the button is not checked?

```
#!/local/bin/python

import cgi

query = cgi.parse()

print "Content-type: text/html"
print

print "<html>\n"
print "<body>\n"
if query.has_key('radiol'):
    print "<p> radiol checked</p>\n"
print "</body>\n"
print "</html>\n"
```

You can try this script here: [test\\_checked.html](#) [[cgi/test\\_checked.html](#)]

### Exercise 2.13. Testing for improper user input

The following script is dangerous for security reasons, why?

```
#!/local/bin/python
```

```
import cgi
import sys

query = cgi.parse()

print "Content-type: text/html"
print

if not query.has_key('filename'):
    print "<P> the file name is required</P>\n"
    print "</BODY>\n"
    print "</HTML>\n"
    sys.exit()

filename=query['filename'][0]
try:
    h = open(filename)
except IOError:
    print "<P> file: ", filename, "does not exist</P>\n"
    print "</BODY>\n"
    print "</HTML>\n"
    sys.exit()

print "<P> <PRE>\n"
for line in h.readlines():
    print line

h.close()

print " </PRE></P>\n"
```

### Exercise 2.14. Testing for improper user input (continued)

Does the setting of this input form [cgi/test\_input\_correct.html] help? Why not?

## 2.6. CGI for helicases

### Exercise 2.15. Submit an helicase

Write a CGI that submits a protein as an helicase. The CGI should return either a message telling that the protein is not an helicase, or if the protein is accepted as an helicase, an html-formatted output of the helicase data. Define a `tohtml` method in the `Helicase` class for this purpose. The output could look like this [[http://www.pasteur.fr/formation/infobio/web/2005/helicases/submit\\_helicase\\_output.html](http://www.pasteur.fr/formation/infobio/web/2005/helicases/submit_helicase_output.html)].

### ? Exercise 2.16. Use a persistent Helicase DB

Write a `dump` method in the `HelicaseDB` class, that stores a persistent version of an `Helicase` object:

```
import helicase
... (load the database with a fasta file) ...
helicedb.dump(sys.argv[2])
```

(the database will be stored into the file provided by `sys.argv[2]`). For this purpose, you can use either the `pickle` or the `shelve` module:

```
import shelve
data = shelve.open(filename)
data['key'] = an_object
data.close()
```

### ? Exercise 2.17. Use a persistent Helicase DB (2)

Write a `load_db` function in the `helicase` module, that restores a persistent version of an `Helicase` object.

```
import helicase
helicedb = helicase.load_db(sys.argv[1])
```

For this purpose, you can use either the `pickle` or the `shelve` module (use the same as in Exercise 2.16):

```
import shelve
data = shelve.open(filename)
an_object = data['key']
data.close()
```


From now on, you will use this function to load the helicases.

### ? Exercise 2.18. Load the database

Submit the sequences from the following file: `helicasesbig.fasta` [<http://www.pasteur.fr/formation/infobio/web/2005/helicases/data/helica>] to your class, and stores the resulting `HelicaseDB` instance for later use.

### ? Exercise 2.19. Fetch an helicase

Write a CGI that returns an helicase according to its ID. Use `load_db` function written in Exercise 2.17.

 **Exercise 2.20. Browse helicases**

Write a CGI that returns a list of helicases IDs, classified by family. The HTML output should enable the user to click on the ID to get the corresponding helicase displayed. The output could look like this [[http://www.pasteur.fr/formation/infobio/web/2005/helicases/browse\\_helicases\\_output.html](http://www.pasteur.fr/formation/infobio/web/2005/helicases/browse_helicases_output.html)].

 **Exercise 2.21. Search helicases by keyword**

Write a CGI that returns a list of helicases matching a keyword. You will need to add a `search` method in the `HelicaseDB` class. The output could look like this [[http://www.pasteur.fr/formation/infobio/web/2005/helicases/search\\_keyw](http://www.pasteur.fr/formation/infobio/web/2005/helicases/search_keyw)].

