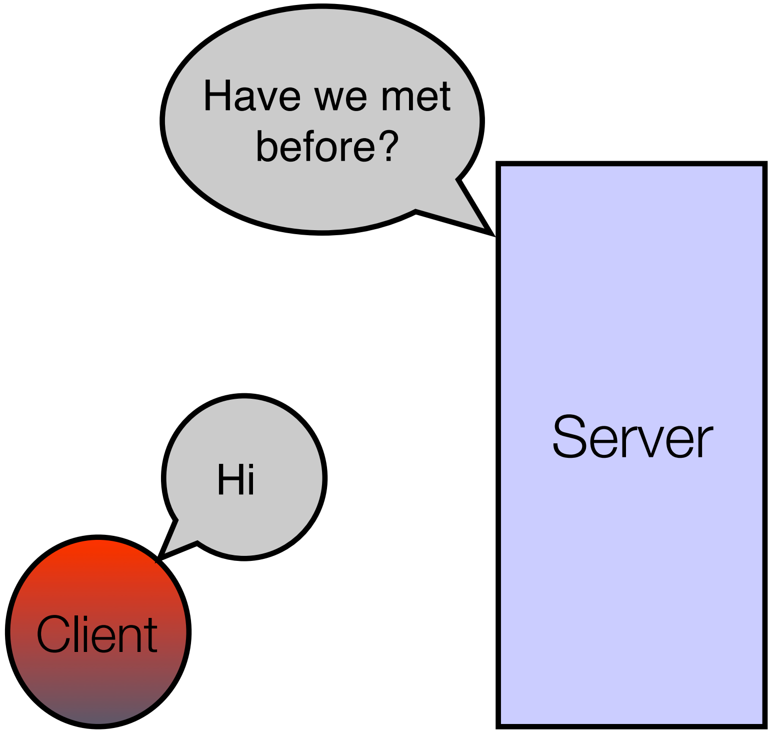
HTTP cookies

Providing state in web applications



Outline

- Motivation for cookies.
- How they work in HTTP.
- Generating unique identities for clients or sessions.
- A small example application.

Motivation for cookies

- HTTP is a stateless protocol.
- Successive invocations of CGI scripts are independent events, with no direct means of communication.
- Without state it is hard for the server to keep track of the client over a series of requests.
- Without state it is hard to make interactive web applications that “remember” what happened in previous interactions.
- Without state it is hard for the server to give each client a unique identity.

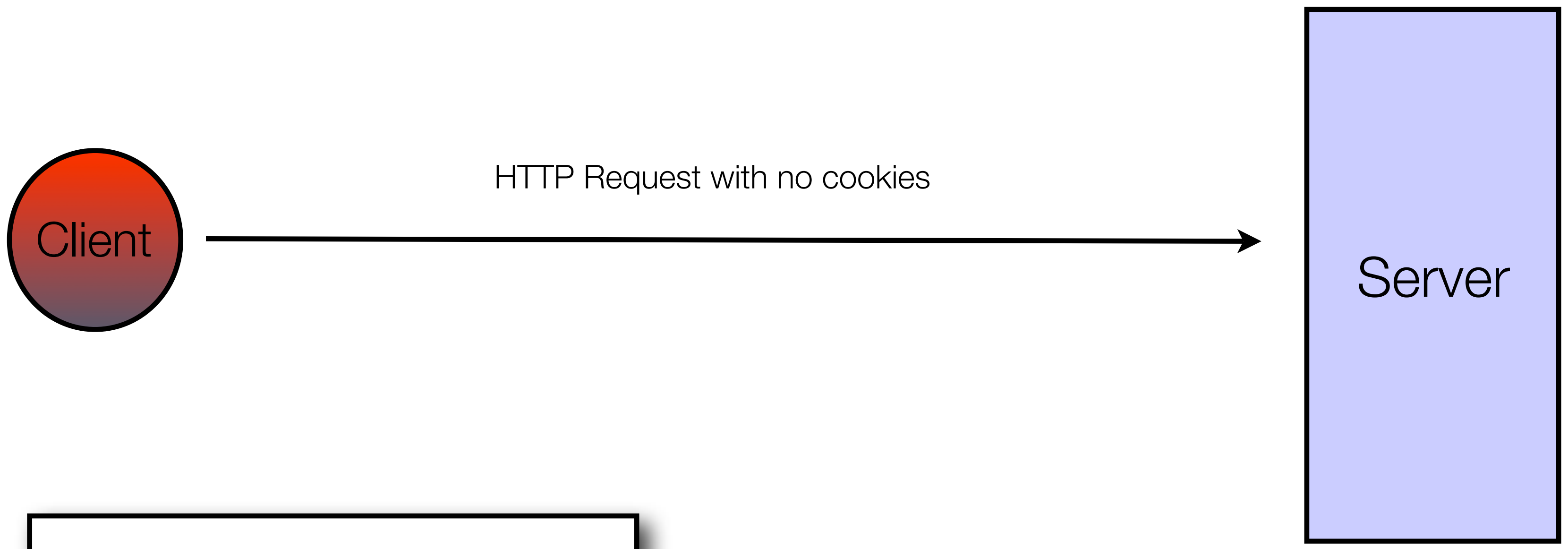
What is a cookie?

- A cookie is simply a small piece of (textual) data which is sent from the server to the client, and from the client back to the server.
- The data is often a string which represents a unique identifier for the client.
- The data is sent as part of the HTTP request and response headers.
- Cookies are stored (for a period) on the client.
- The content of cookies generally means nothing to the client.
- Each cookie is associated with the domain name of the server from where it originated.

What is a cookie?

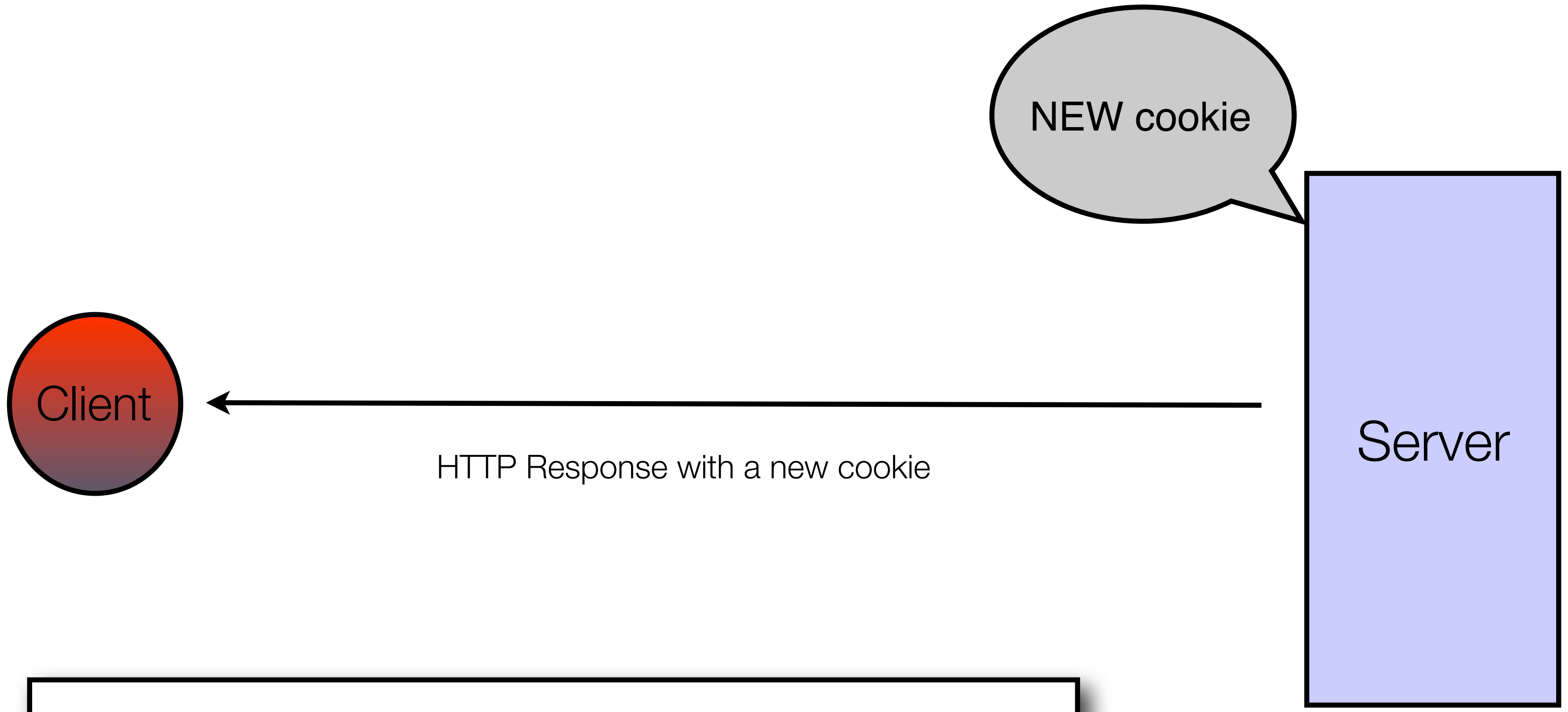
- When the client makes a request to a server, it checks if it has stored any cookies from that server.
- If the client does have cookies for a particular server, it automatically sends them back as part of each HTTP request to that server.
- Cookies have expiry dates. The client deletes them from its store of cookies when they expire.
- A server can send multiple cookies to the client (but clients have limits on how many they will keep, and how big they can be).
- A client can refuse to accept cookies, but it might make some websites unusable.

A simple scenario, step one



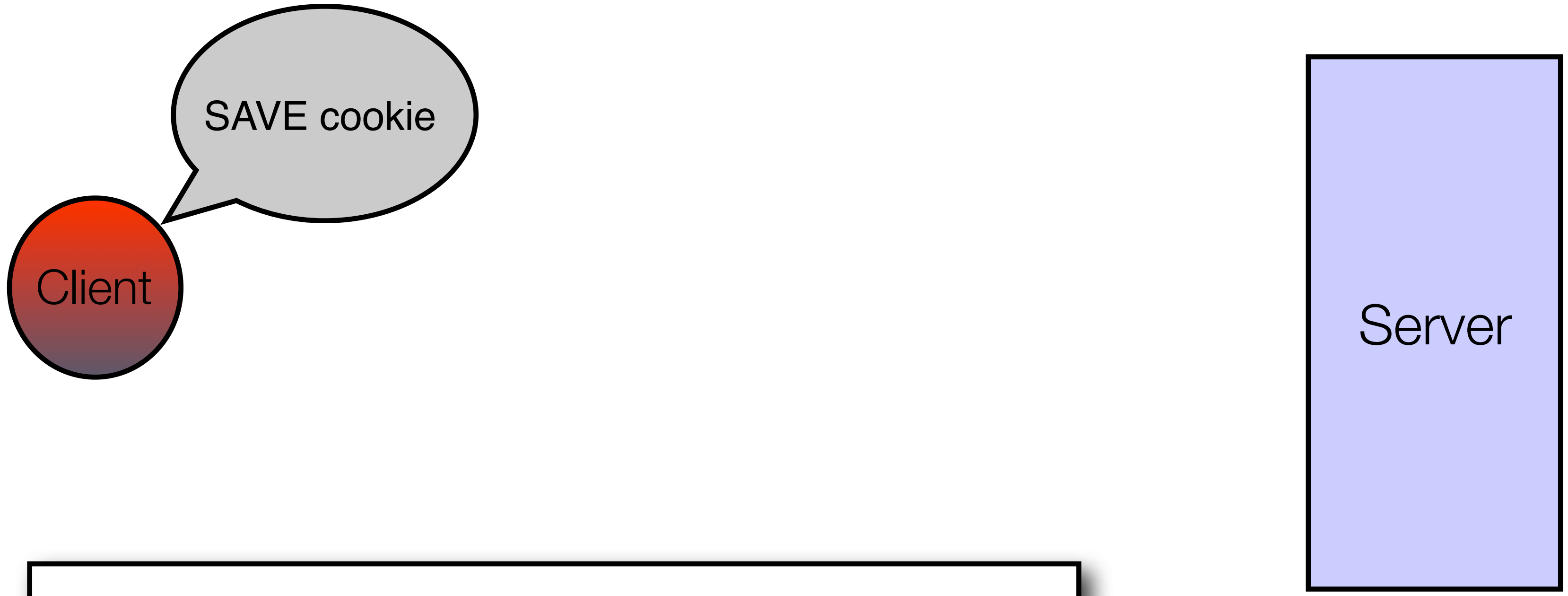
The client (typically a web browser) makes a request to the server for the first time.

A simple scenario, step two



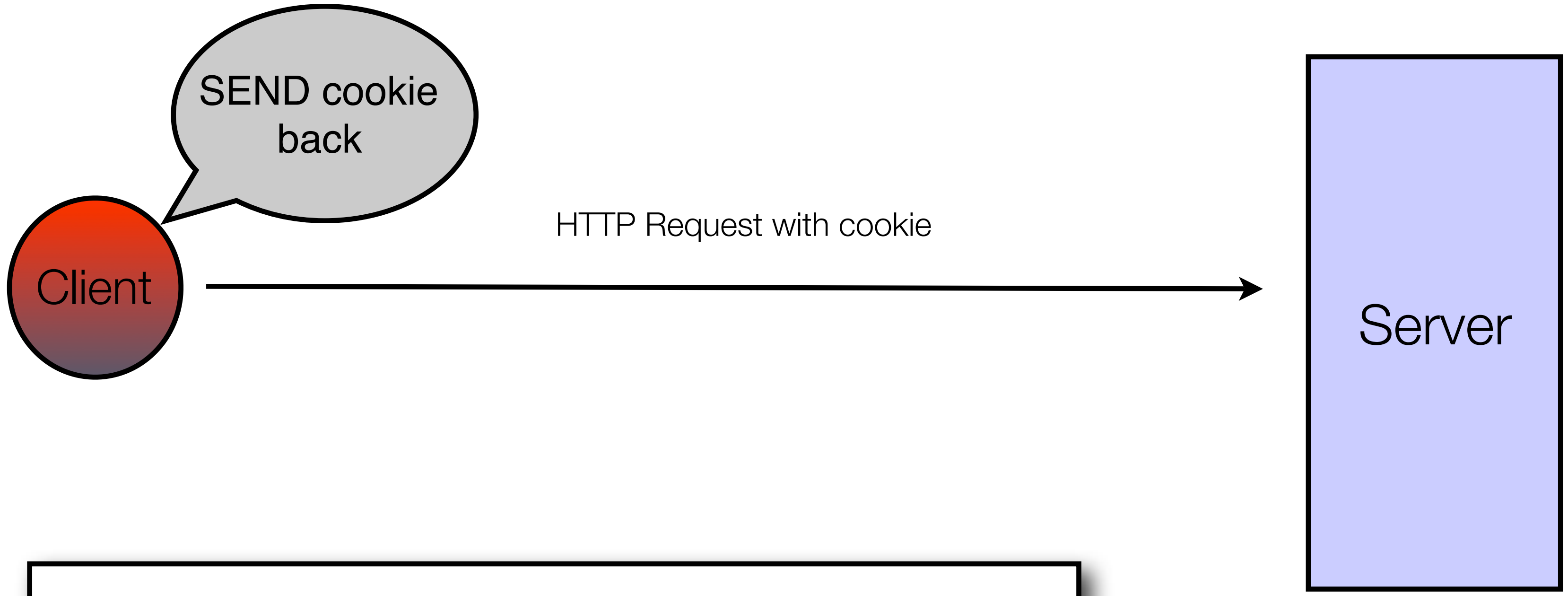
The server observes that the client has not sent a cookie. So the server generates a new unique cookie and sends it back to the client as part of its response.

A simple scenario, step three



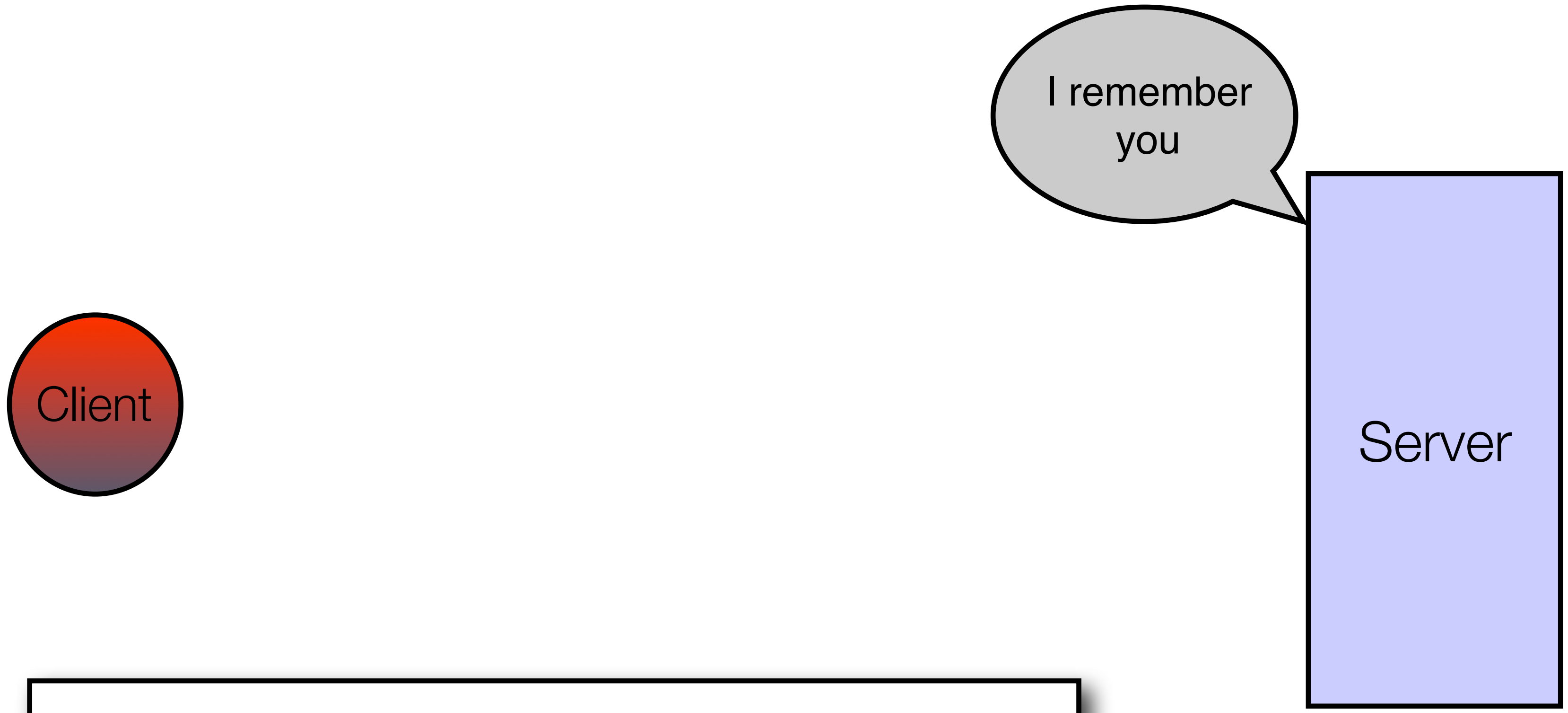
The client saves the new cookie from the server in its private store of cookies. It associates the cookie with the domain name of the server.

A simple scenario, step four



Later, the client sends another request to the server. This time it observes that is has a cookie saved for the domain name of the server, so it sends the cookie as part of its request.

A simple scenario, step five



The server observes the cookie in the request and remembers that it sent the cookie in a previous response. The cookie allows the server to identify the client.

A little CGI script to send a cookie to the client

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

seconds_in_week = 60 * 60 * 24 * 7

print 'Set-Cookie: foo=1234; Max-Age=%d;' % seconds_in_week
print 'Content-Type: text/plain'
print

print 'Enjoy the cookie!'
```

Suppose this script is called “cookie_send.py”.

A little CGI script to send a cookie to the client

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

seconds_in_week = 60 * 60 * 24 * 7

print 'Set-Cookie: foo=1234; Max-Age=%d;' % seconds_in_week
print 'Content-Type: text/plain'
print

print 'Enjoy the cookie!'
```

The data in a cookie is of the form “name=value”.

A little CGI script to send a cookie to the client

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

seconds_in_week = 60 * 60 * 24 * 7

print 'Set-Cookie: foo=1234; Max-Age=%d;' % seconds_in_week
print 'Content-Type: text/plain'
print

print 'Enjoy the cookie!'
```

Max-Age specifies how many seconds the cookie should be kept by the client.

Demonstration of cookie sent from server to client using HTTP.
The “telnet” program is used to demonstrate the interaction.

```
telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /~bjpop/cookie_send.py HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 11 Sep 2008 12:07:31 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
Set-Cookie: foo=1234; Max-Age=604800;
Connection: close
Content-Type: text/plain

Enjoy the cookie!
Connection closed by foreign host.
```

Client connects to port 80 on the server.

```
telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /~bjpop/cookie_send.py HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 11 Sep 2008 12:07:31 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
Set-Cookie: foo=1234; Max-Age=604800;
Connection: close
Content-Type: text/plain

Enjoy the cookie!
Connection closed by foreign host.
```

Client sends a GET request for the CGI script called “~bjpop/cookie_send.py”.

```
telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /~bjpop/cookie_send.py HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 11 Sep 2008 12:07:31 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
Set-Cookie: foo=1234; Max-Age=604800;
Connection: close
Content-Type: text/plain

Enjoy the cookie!
Connection closed by foreign host.
```

Response from the server.

```
telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /~bjpop/cookie_send.py HTTP/1.0
HTTP/1.1 200 OK
Date: Thu, 11 Sep 2008 12:07:31 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
Set-Cookie: foo=1234; Max-Age=604800;
Connection: close
Content-Type: text/plain

Enjoy the cookie!
Connection closed by foreign host.
```

Server asks the client to save this cookie “foo=1234”. It should expire after a week.

```
telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /~bjpop/cookie_send.py HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 11 Sep 2008 12:07:31 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
Set-Cookie: foo=1234; Max-Age=604800;
Connection: close
Content-Type: text/plain

Enjoy the cookie!
Connection closed by foreign host.
```

A little CGI script to receive a cookie from the client

Suppose this script is called “cookie_receive.py”.

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

import os

print 'Content-Type: text/plain'
print

if 'HTTP_COOKIE' in os.environ:
    cookie_value = os.environ['HTTP_COOKIE']
    print 'Thanks for the cookie: %s' % cookie_value
else:
    print "Where's my cookie?"
```

A little CGI script to receive a cookie from the client

If the client sends a cookie to the server, the CGI script will find the value of the cookie in the environment variable called HTTP_COOKIE.

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

import os

print 'Content-Type: text/plain'
print

if 'HTTP_COOKIE' in os.environ:
    cookie_value = os.environ['HTTP_COOKIE']
    print 'Thanks for the cookie: %s' % cookie_value
else:
    print "Where's my cookie?"
```

Client sends GET request to the server for the CGI script called "~bjpop/cookie_receive.py".

```
telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /~bjpop/cookie_receive.py HTTP/1.0
Cookie: foo=1234

HTTP/1.1 200 OK
Date: Thu, 11 Sep 2008 12:40:40 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
Connection: close
Content-Type: text/plain

Thanks for the cookie: foo=1234
Connection closed by foreign host.
```

```
telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /~bjpop/cookie_receive.py HTTP/1.0
Cookie: foo=1234

HTTP/1.1 200 OK
Date: Thu, 11 Sep 2008 12:40:40 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
Connection: close
Content-Type: text/plain

Thanks for the cookie: foo=1234
Connection closed by foreign host.
```

Client sends cookie value to the server as part of the request header.

Server responds to the client. The body of response indicates that the CGI script received the cookie value “foo=1234”

```
telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /~bjpop/cookie_receive.py HTTP/1.0
Cookie: foo=1234

HTTP/1.1 200 OK
Date: Thu, 11 Sep 2008 12:40:40 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
Connection: close
Content-Type: text/plain

Thanks for the cookie: foo=1234
Connection closed by foreign host.
```

Generating a unique identity for a client

- Cookies are often used to give identities to clients.
- When a new cookie is created, a unique identity must be created.
- How can you do this?
- You could use an integer counter; 0 is the first user, 1 is the second user and so on.
- To keep track of the counter you'd have to store it in a file (or database) on the server.
- Will be very slow.

Generating a unique identity for a client

- Python provides a library to generate unique identifiers efficiently.

```
>>> import uuid
>>> x = uuid.uuid4()
>>> print x
f9fa4ae3-2405-4dd3-942d-16c06d319574
>>> y = uuid.uuid4()
>>> print y
1c57d13d-65dd-47e7-bdaf-bbf9e2b26857
```

- See: `http://www.python.org/doc/lib/module-uuid.html`

A CGI script which keeps a count of how many times it is visted by each client.

WARNING: not a robust program. Only intended to demonstrate a simple example. You should do more error checking in real life. Remember what you learned about defensive programming!

```
import os
import uuid

if 'HTTP_COOKIE' in os.environ:
    cookie = os.environ['HTTP_COOKIE']
    uid = cookie[4:]
    file = open(uid, "r")
    count = int(file.read())
    file.close()
else:
    uid = str(uuid.uuid4())
    ten_minutes = 60 * 10
    print 'Set-Cookie: uid=%s; Max-Age=%d;' % (uid, ten_minutes)
    count = 0

file = open(uid, 'w')
file.write(str(count+1))
file.close()

print 'Content-Type: text/plain'
print

print "Your id is %s, I've seen you %d times before." % (uid, count)
```

```
import os
import uuid

if 'HTTP_COOKIE' in os.environ:
    cookie = os.environ['HTTP_COOKIE']
    uid = cookie[4:]
    file = open(uid, "r")
    count = int(file.read())
    file.close()
else:
    uid = str(uuid.uuid4())
    ten_minutes = 60 * 10
    print 'Set-Cookie: uid=%s; Max-Age=%d;' % (uid, ten_minutes)
    count = 0

file = open(uid, 'w')
file.write(str(count+1))
file.close()

print 'Content-Type: text/plain'
print

print "Your id is %s, I've seen you %d times before." % (uid, count)
```

Test whether we have seen the client before.

We have seen the client before. Get their unique identity from the cookie. Read the counter from the file whose name is equal to the unique identity.

```
import os
import uuid

if 'HTTP_COOKIE' in os.environ:
    cookie = os.environ['HTTP_COOKIE']
    uid = cookie[4:]
    file = open(uid, "r")
    count = int(file.read())
    file.close()
else:
    uid = str(uuid.uuid4())
    ten_minutes = 60 * 10
    print 'Set-Cookie: uid=%s; Max-Age=%d;' % (uid, ten_minutes)
    count = 0

file = open(uid, 'w')
file.write(str(count+1))
file.close()

print 'Content-Type: text/plain'
print

print "Your id is %s, I've seen you %d times before." % (uid, count)
```

```
import os
import uuid

if 'HTTP_COOKIE' in os.environ:
    cookie = os.environ['HTTP_COOKIE']
    uid = cookie[4:]
    file = open(uid, "r")
    count = int(file.read())
    file.close()
else:
    uid = str(uuid.uuid4())
    ten_minutes = 60 * 10
    print 'Set-Cookie: uid=%s; Max-Age=%d;' % (uid, ten_minutes)
    count = 0

file = open(uid, 'w')
file.write(str(count+1))
file.close()

print 'Content-Type: text/plain'
print

print "Your id is %s, I've seen you %d times before." % (uid, count)
```

Not good programming style. Relies on the fact that cookie will be of the form: “uid=something”

We have not seen the client before. Generate a new unique identifier for them. Send it to them in a cookie which expires in ten minutes. Set the counter to zero.

```
import os
import uuid

if 'HTTP_COOKIE' in os.environ:
    cookie = os.environ['HTTP_COOKIE']
    uid = cookie[4:]
    file = open(uid, "r")
    count = int(file.read())
    file.close()
else:
    uid = str(uuid.uuid4())
    ten_minutes = 60 * 10
    print 'Set-Cookie: uid=%s; Max-Age=%d;' % (uid, ten_minutes)
    count = 0

file = open(uid, 'w')
file.write(str(count+1))
file.close()

print 'Content-Type: text/plain'
print

print "Your id is %s, I've seen you %d times before." % (uid, count)
```

```
import os
import uuid

if 'HTTP_COOKIE' in os.environ:
    cookie = os.environ['HTTP_COOKIE']
    uid = cookie[4:]
    file = open(uid, "r")
    count = int(file.read())
    file.close()
else:
    uid = str(uuid.uuid4())
    ten_minutes = 60 * 10
    print 'Set-Cookie: uid=%s; Max-Age=%d;' % (uid, ten_minutes)
    count = 0

file = open(uid, 'w')
file.write(str(count+1))
file.close()

print 'Content-Type: text/plain'
print

print "Your id is %s, I've seen you %d times before." % (uid, count)
```

Increment the counter and write it to the file which has the same name as the unique identifier.

```
import os
import uuid

if 'HTTP_COOKIE' in os.environ:
    cookie = os.environ['HTTP_COOKIE']
    uid = cookie[4:]
    file = open(uid, "r")
    count = int(file.read())
    file.close()
else:
    uid = str(uuid.uuid4())
    ten_minutes = 60 * 10
    print 'Set-Cookie: uid=%s; Max-Age=%d;' % (uid, ten_minutes)
    count = 0

file = open(uid, 'w')
file.write(str(count+1))
file.close()
```

Print a text document which reports the unique identity of the client, and the number of times we have seen them.

```
print 'Content-Type: text/plain'
print

print "Your id is %s, I've seen you %d times before." % (uid, count)
```

Your first big lesson in concurrency

- The previous CGI script for counting the visits by clients suffers from a classic problem in computer science called a "race condition".

- Consider what happens if the same client visits the CGI script at nearly the same time.

- Correct behaviour of the script requires that increments to the counter file must be atomic events, but there is nothing in the code to guarantee this!

- Technically, we should "lock" the counter file, so that only one instance of the script can read and write to it at any one time.

- However, the likelihood of an error is slim in this case, and the consequences are not severe (we just get the wrong count for a client). But in real life the consequences could be far worse (think: banking transaction).

Where to now?

- Cookies can store more things, such as “path” information, and “domain” information. Useful when you have more than one CGI application running on a particular server.
- Next week’s workshop will cover CGI, forms and cookies.
- Python provides some library support for handling cookies, you will see it in the workshop.
- Debugging CGI scripts is hard work. Use the logging facility from the debugging workshop.