COMP 4021
Internet Computing

Cookies and HTTP

Stateless Protocols

- HTTP is a stateless protocol, and so are all internet protocols (TCP, IP, etc.)
- The protocol does not remember the "state/history" of a conversation
  - An action is fully specified by the command and parameters
  - The action does not depend on result of previous interactions
  - Stateless: Send the packet with id=…. to destination with ip address=…
    - For HTTP: GET http://www.cse.ust.hk/faculty/dlee/index.html
    - The request will return the same page whether you issue it today or tomorrow or from any browser
  - Stateful: Must know previous actions to complete current command:
    - Send next packet; retry previous packet; sent to next address

Stateful Protocols

- Stateful protocol remembers the state of the interaction
  - Typically requires opening a channel between the communicating nodes and maintaining state variables for each channel
- In the previous examples, what states are remember?
  - Send next packet; retry previous packet; sent to next address

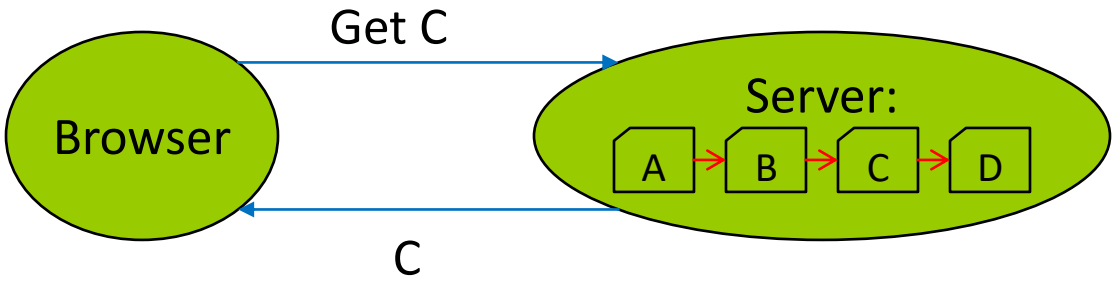
Stateless vs Stateful Protocols

Get a page given exact URL	Stateless	
Get the current page	Stateful	"current" depends on where you are
Get the "next" page	Stateful	"next" depends on where you are

- Stateful protocols are more efficient from the user or functional point of view, but stateless protocols are more scalable and recoverable (i.e., suitable for internet) WHY?

HTTP is Stateless

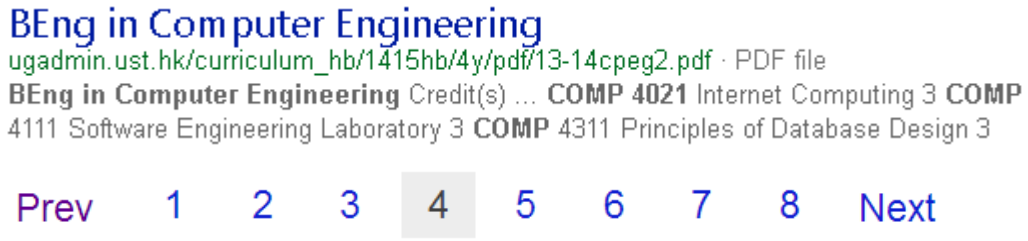
- GET www.ust.hk/faculty/dlee/index.html
- The command and URL are fully specified



- There is no "get next page" in HTTP
  - The "next" page (D) is hardcoded in the previous page (C)
  - GET …/…/c.html ⇒ Returns c.html with a link to …/…/d.html
  - Clicking the link …/…/d.html will issue GET …/…/d.html
- Question: does this interaction requires memory of any "state"?
- Question: What if the server or network is down?

Example: Search Engine Result Page

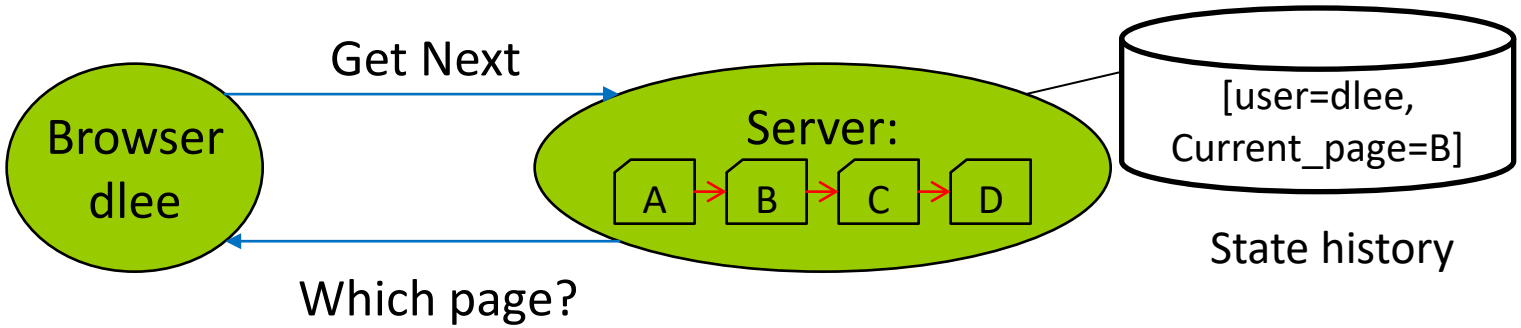
- Bing's navigation bar for the result pages



  - Current page showing results from 31 to 40
  - Prev URL:
    http://www.bing.com/search?q=4021&go=&qs=n&pq=comp+4021&sc=0-0&sp=-1&sk=&**first=21**&FORM=PQRE
  - Next URL:
    http://www.bing.com/search?q=4021&go=&qs=n&pq=comp+4021&sc=0-0&sp=-1&sk=&**first=41**&FORM=PORE
  - Question: What are the current and next state(s) of this interaction?

Keeping States in Server Application

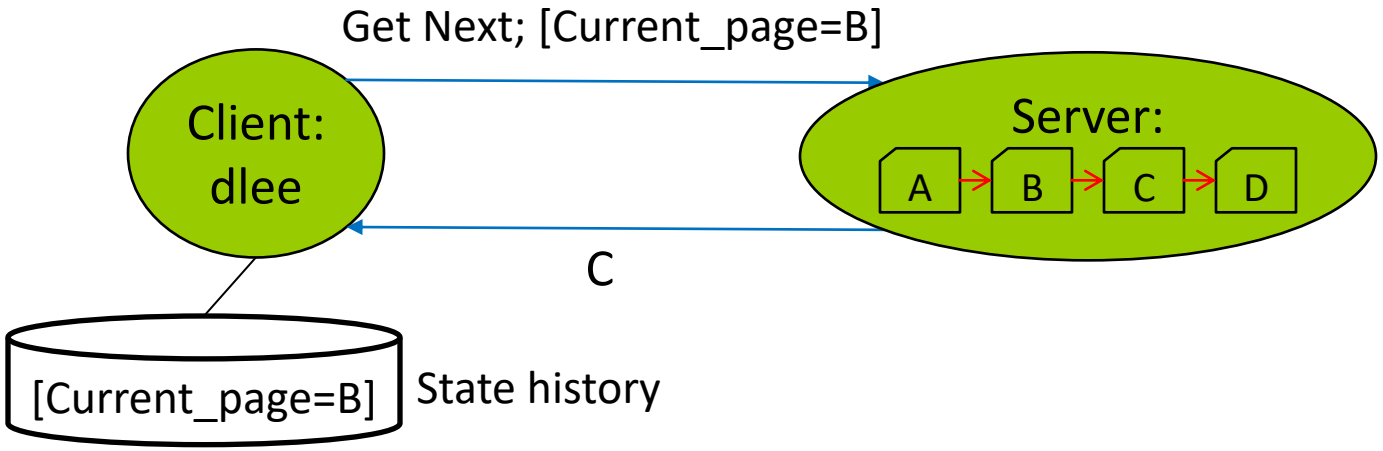
- The result of a stateful request depends on the "state" of the interaction (i.e., the current page
  - The meaning of "next page" depends on what the "current page" is



- Server remembers the current state, which consists of two values [user ID, current_page], for each conversation
  - State: [user=dlee, current_page=B]
  - Get Next received from user=dlee => returns C

Keeping State in Client

- Instead of the server keeping the states for all interactions, the client keeps its own state; server does not remember state



- Client sends the current state to server with each request
- When the page ("C" in this example) is received, update state [Current_page=C]

Three ways to remember States

- Hardcoded in webpage (as in the Bing example)
- Stored in browser (as cookies)
  - Server application embeds JavaScript functions or issues HTTP requests to create/update/delete cookies in browser [to be covered next]
- Stored in server (e.g., in a file or backend database)
- A combination of the above

Pros and Cons of keeping states in browser or server side?

What Are Cookies?

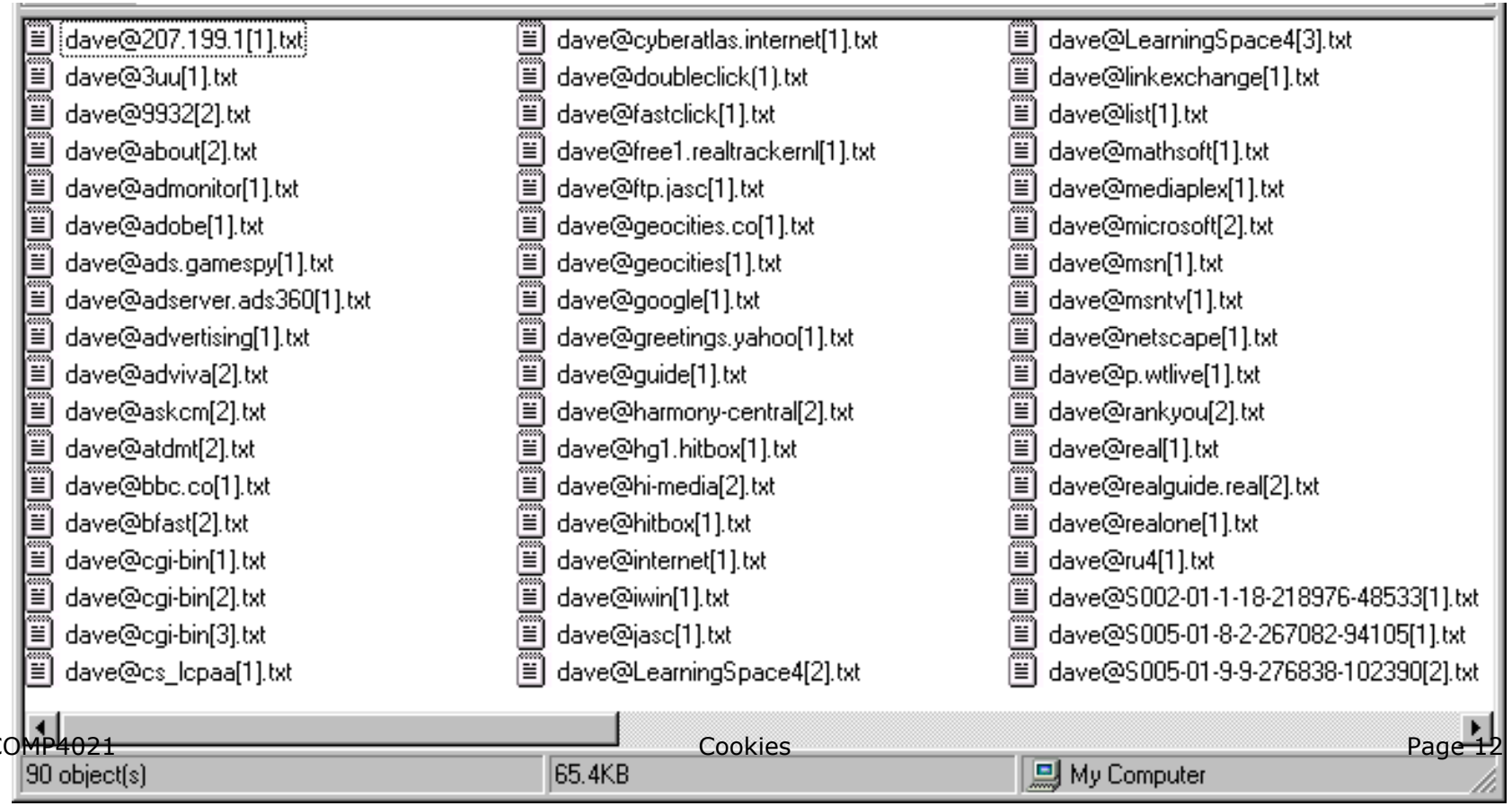
- Basically, cookies are just text strings that allow the browser/server to remember the "states" of the interaction
  - E.g., Whom am I talking to (user name), visit history, etc.
- A website can store cookies in a browser and later read and modify them
- If you visit the web site again, then JavaScript code in the web page can read the cookies that were stored earlier
- Not just websites – files loaded from local disk can also use cookies (If cookies are handled by client-side JavaScript)

Things Stored in Cookies

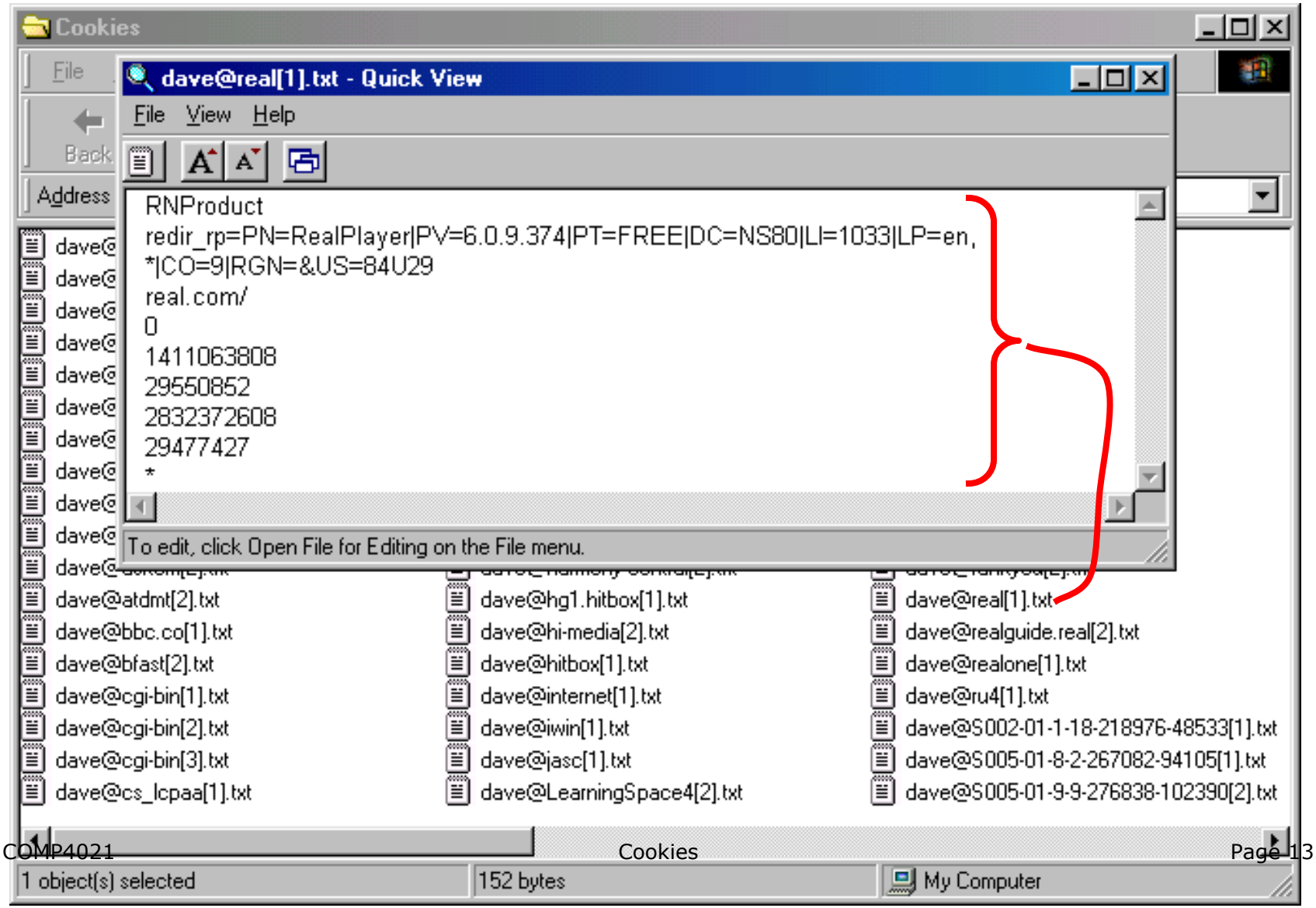
- You can store anything you want in a cookie
- Examples of things commonly stored:
  - The date/ time you last visited the web site
  - How many times you have been to the web site
  - What things you clicked on (e.g. books, etc)
  - The highest score so far – for a game

Example Cookie Data - IE

- For IE, each web site is given its own cookies file
- All cookie data for that web site is stored in that file



Example Cookie Data - IE



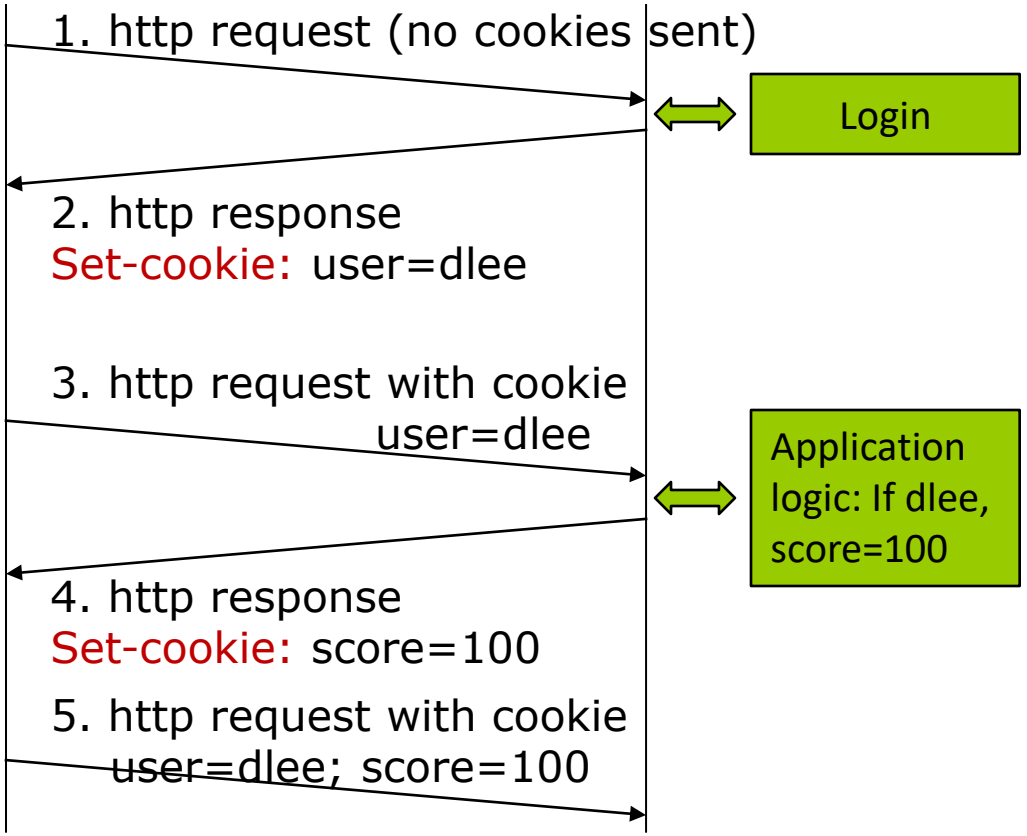
Cookie Limitations

- A browser can hold up to 300 cookies (totally 4K bytes)
  - If you have one cookie, the cookie can be 4K bytes in size
  - If you have two cookies, the average size is 2K bytes, etc.
  - If you already have a cookie, and your new cookie would make the total size exceeding 4K, the old cookie will be deleted
- There is a maximum of 20 cookies per web site
  - When you add the 21st cookie, the oldest one would be deleted
- These limitations are browser dependent, so use a smaller limit if you want to be compatible with more browsers
- 4K is not really 4K, leave room for overhead, delimiters, etc
- A cookie is not just name+value, leave room for expire, path, etc.

Browser-Server Interaction Example

- All cookies created by a website are sent to the server with each request, whether the server needs/wants them or not !

1. Initially, there is no cookie set in browser
2. Application sets a cookie (via http response header or JavaScript function)
3. Browser sends request and all cookies
4. Application sets new cookie
5. Browser sends request and all cookies



JavaScript on Cookies

- JavaScript controls cookies through the document.cookie property

- You can read cookies by reading it
- You can make a cookie by changing it

Reading the Cookies

- document.cookie is a string containing all of the cookies associated with the current web site
- To see all the cookies do: alert(document.cookie)
- If document.cookie has three cookies it will have this structure:

cookieName1=cookieValue1;cookieName2=cookieValue2;cookieName3=cookieValue3

  - For example: name=dave;score=8900;total_time=46
- So you need JavaScript string handling to extract the individual data out of the string

Saving/Updating a Cookie

- Because document.cookie is just a string, changing it is straightforward
- For example, the following JavaScript statement sets two cookies:
  document.cookie = "name=peter;score=260";

Example JavaScript

- Put the date/time in a cookie, and shows it in the web page

```
<head>   <script>
  var today = new Date();
  now = today.getHours() + ":" + today.getMinutes() + ":" +
     today.getSeconds();

  document.cookie = "last_visit=" + now;
  document.writeln(document.cookie );
</script>  </head>
```

Same as “write” but with a newline

Cookie Expiry Time

- It is a good idea to also set a time for the cookie to ‘die’ (expire)
- Here a cookie is made with a specific time to expire:
  document.cookie = "name=peter;

  expires=Tue, 17-Mar-08 00:00:01 GMT";

Deleting a Cookie

- The way to delete a cookie is to set the date/time of the cookie to a date/time that has already finished (i.e. 1970, or one second ago, or one hour ago)
- The browser will then automatically remove the cookie
- For example:
  document.cookie = "name=peter;
  expires=Thu, 01-Jan-70 00:00:01 GMT";

Altering a Cookie

- What if you have already made a cookie, but now you want to change it?
- Cookies can be altered by simply
  1. reading the document.cookie string
  2. changing the string as appropriate
  3. copying the string back to document.cookie

Functions for Handling Cookies

- You can define functions to help with handling cookies
- The source code for a set of routines for handling cookies is shown on the next few slides
  - setCookie()
  - getCookie()
  - deleteCookie()

Cookie Handling - setCookie

```
function setCookie(name, value, expires, path, domain, secure) {
    var curCookie = name + "=" + escape(value) +
      ((expires) ? "; expires=" + expires.toGMTString() : "") +
      ((path) ? "; path=" + path : "") +
      ((domain) ? "; domain=" + domain : "") +
      ((secure) ? "; secure" : "");
    document.cookie = curCookie;    }
```

- This code creates a cookie (using parameters passed to it)
- The expiry time needs to be given to it in milliseconds

Advanced Cookie Properties

- path: URL path must exist in the requested resource before sending the Cookie header
- domain: Host to which the cookie will be sent. Defaults is the host of the current document location
  - A server cannot set a cookie and request it to be sent to a server with a different domain
- secure: A flag (no value) indicating the cookie will only be sent to the server when HTTPS is used

name=dlee;path=/faculty/dlee;domain=www.cse.ust.hk;secure

Cookie Handling - getCookie

```
function getCookie(name)  {
    var dc = document.cookie;
    var prefix = name + "=";
    var begin = dc.indexOf("; " + prefix); // search ";score"
    if (begin == -1) {
      begin = dc.indexOf(prefix);
      if (begin != 0) return null;
    } else  begin += 2;
    var end = dc.indexOf(";", begin);
    if (end == -1) end = dc.length;
    return unescape(dc.substring(begin + prefix.length, end));    }
```

Use string functions to extract data from the cookie string, e.g., let's get "score":
name=dave;score=8900;total_time=46

Position (or Index) where ";prefix" appears in cookie string

cookie not found

Convert %20 to " ", etc.

Start and end positions of the name cookie in cookie string

Cookie Handling - deleteCookie

- **path** - path of the cookie (must be same as path used to create cookie)
- **domain** - domain of the cookie (must be same as domain used to create cookie)

```
function deleteCookie(name, path, domain) {
    if (getCookie(name)) {
     document.cookie = name + "=" +
     ((path) ? "; path=" + path : "") +
     ((domain) ? "; domain=" + domain : "") +
     "; expires=Thu, 01-Jan-70 00:00:01 GMT";    }    }
```

Using the Functions

- Two examples follow
- Example 1 – A web counter
  - Each time you visit the page, it adds one to a counter stored in a cookie
- Example 2 – A name tracker
  - The name of the user is stored in a cookie and is shown every time the page is visited

Example 1 - Web Counter

- Use a cookie to count how many times someone has visited a particular web page
- The following script displays the number of times the user has visited, assuming just one person uses the browser)
- Reload the page to see the counter increment

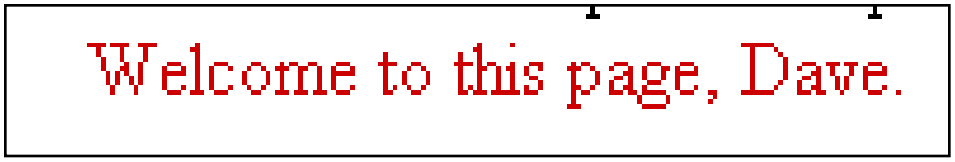
Example 1 - Web Counter

```
var now = new Date();       // create an instance of the Date object
now.setTime(now.getTime() + 365*24*60*60*1000);  // expires in 365 days
                                                 // getTime() and setTime() work in msec
var visits = getCookie("counter");
if (!visits) {
    visits = 1;        // if the cookie wasn't found, this is the first visit
    document.write("By the way, this is your first time here.");
    } else {
            visits = parseInt(visits) + 1; // increment the counter
            document.write("By the way, you have been here " + visits + " times.");
    }
setCookie("counter", visits, now); // set the new cookie
```

name, value, expire

Example 2 - Name Tracker

- The following script asks the user for his/ her name, and "remembers" the input
- It then welcomes the user each time he/ she accesses the page, without asking again for the name



Example 2 - Name Tracker

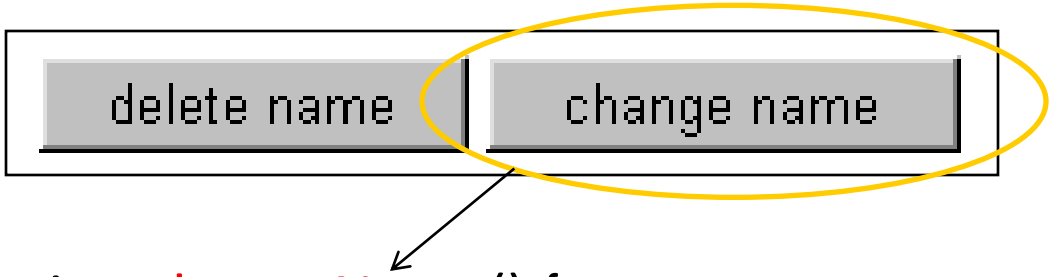
```
var now = new Date();       // create an instance of the Date object

now.setTime(now.getTime() + 365 * 24 * 60 * 60 * 1000);
var username = getCookie("username");
 // if the cookie wasn't found, ask for the name
if (!username) username = prompt("Please enter your name:", "");

setCookie("username", username, now);      // set the new cookie
if (username) {
    document.write("Welcome to this page, " + username + ".");
    setCookie("username", username, now);
    } else document.write("You refused to enter your name.");
```

Example 2 - Name Tracker

- Altering a cookie, e.g., allow user to change to a new name




```
function changeName() {
    var now = new Date();
    // cookie will expire one year later
    now.setTime(now.getTime() + 365 * 24 * 60 * 60 * 1000);
    username = prompt("Please enter your name:", "");
    setCookie("username", username, now);     }
```

Set-Cookie: HTTP Response Header

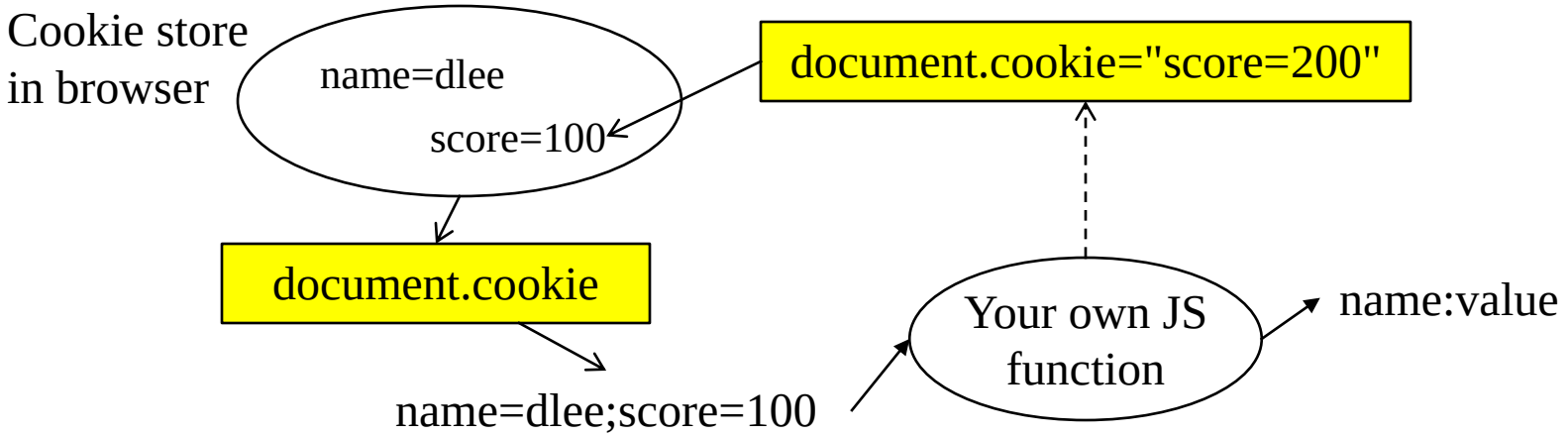
- When the server sends a response to the browser, it will send a bunch of HTTP response headers in front of the content

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
… …
Set-Cookie: <cookie-name>=<cookie-value>;Expires=<date>
```

- Set-Cookie sets cookies in the browser
- Set-Cookie is a response header, so it must be in the header section of the response (i.e., before any content)

Important Clarification

- Cookies are stored in a cookie store in the browser
- How the cookies are stored in the browser is browser dependent, but they are stored in disk files for persistence
- In fact, the cookie store implementation is irrelevant to most end users, since DOM provides a uniform interface for creating, updating, deleting and reading cookies:



The Cookie Interface

- document.cookie returns a string containing the name/value pairs of all cookies in the cookie store, e.g.,

  name=dlee;score=1300;last_visit=22:55:18;

- document.cookie is the only interface for getting cookie values
  - You need to writing your own JS code to get cookie values and implement all error checking functions
  - E.g., to get the value of the “score” cookie, check if "score" exists in dococument.cookie; if yet, get and return the value; if not, return NULL
  - See getCookie() function
- document.cookie is not a regular string -> See next slide

Creating and Updating Cookie

- To create a new cookie :
  - document.cookie = "grade=A"
  - Since grade is not an existing cookie, it is created; document.cookie is:

    name=dlee;score=1300;last_visit=22:55:18;grade=A
- document.cookie is not a regular string, if it is, "grade=A" should replace the old value and document.cookie becomes "grade=A"; now "grade=A" is appended into document.cookie
- Update an existing cookie:
  - document.cookie = "score=1000"
  - Since score exists, its value is updated; document.cookie becomes: name=dlee;last_visit=22:55:18;grade=A;score=1000
  - Order of the list is typically based on last-modified-time; latest is last

Wrong Cookie Update

- Although document.cookie returns:
  - name=dlee;score=1300;last_visit=22:55:18;grade=A
- If you want to update the values of all four cookies, you cannot do:
  - document.cooke="name=john;score=1200;visit=00:00;grade=A+"
- Only the first name=value is recognized, but the rest is ignored
- If original cookie is:

  name=dlee;score=1300;last_visit=22:55:18;grade=A
- document.cookie becomes:

  name=john;score=1300;last_visit=22:55:18;grade=A

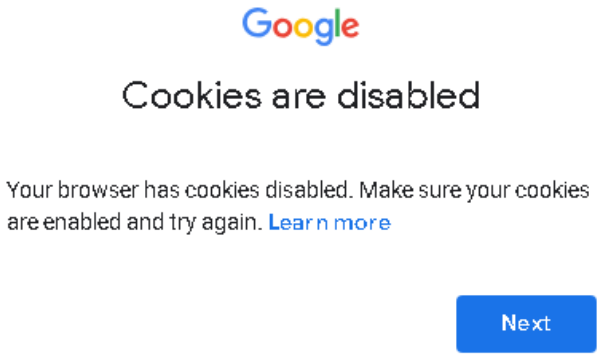
Correct Way to Update Cookies

- If you want to change all four cookies, do it one by one (you can also add expires or path attributes):

```
document.cooke="name=john";
document.cookie="score=1200;expires=Thu, 01-Jan-70 00:00:01 GMT";
document.cookie="visit=00:00";
document.cookie="grade=A+"
```

Take Home Message

- Cookies is a quick way of storing information (or states) about an interaction
  - Number and size of cookies are limited, but enough for most applications
  - Storing cookies as strings make cookies hard to access and maintained but very easy to send
- Cookies are not just for displaying friendly messages to the users (e.g., greetings), they are required in many web applications, e.g. gmail requires cookies to work



Take Home Message (Cont.)

- Cookies are stored in browsers by a web application
  - Switching browsers will lose the cookies
  - Users cannot change the cookies (or understand their meaning), but can delete or refuse to accept them
  - When a user accesses a website, all of the website's cookies (if any) will be sent to the website
- Application can set cookies by (i) triggering a JavaScript function or (2) using Set-Cookie HTTP Response Header