A Web Based Lunch Ordering System
by CC Ang (calcium@altavista.net) (version 1b, 16 May 2000).

In The Beginning
Maybe this article should be entitled "How I Discovered Python and Ditched Everything Else" instead. Anyway, I have always wanted to write web based applications but somehow found that getting started was quite intimidating. So having procrastinated for years, I finally got around to writing my first application. My work had required an intra-office application where some values were needed to be prompted on a a web page, which is sent to a CGI script, cross-verified via an SQL database, dispatched to a waiting process via sockets, and the results sent back to the web page1.

By luck, I stumbled upon a scripting language called Python. I was reading a recent issue of The Linux Journal (Dec 1999), in which they interviewed a computer science celebrity, Eric Raymond in which he mentioned that he now does only Python. At this stage, I was a day into trying to implement the above system in Perl and was not quite finished. If Python was good enough for Eric, it was worth a try. Well, I finished what I wanted to do in just over 2 hours. This was from zero Python, of which I had not heard of 2 hours prior. At the risk of losing my professional advantage, I thought I'd share with others as to how easy Python is, especially to do CGI (and almost everything else). As the above application would be too technical and boring to actually work through (and I'd probably get sued by my employers), I've decided to work through another much more interesting exercise.

The Problem Description.
Work being situated in a semi remote location2 (culinary wise, except for the place next door, which has excellent food but a bit expensive to eat lunch at every day), take-out lunch was organised to be delivered to us once a week. Each participant, in turn, organised the lunch orders. Being spread out over three floors, it was quite a chore and no one looked forward to doing it. A web based ordering system seemed the obvious solution but having not done any CGI programming before, it was never attempted as it seemed quite overwhelming to me3. The others did not seem to care. But writing CGI web systems can be quite simple especially when one can do it using Python. (OK, Perl gurus may disagree but that’s the whole point, ie a Perl guru versus a Python novice!)

The Initial Requirements.
I roughly knew how I wanted it structured. There’d be a web page with a pull down list of the restaurant menu and on clicking on a submit button, an e-mail with the persons order would be sent to the nominated lunch organiser.

Based on hearsay and some cursory research on the net, I decided to use the following tools.

- Javascript for the client end (the web page).
- Python for the server side.
- Apache for the web server, which is distributed with Linux (well, it was with my copy of RedHat Linux 6.2). There is also a Windows version too if one is so inclined.
- Linux for the webserver OS.

1) Designing the web pages.
   Anyway, I decided to keep it fairly simple. A pull down box with some radio buttons. See Figure 1.

---

1 Yes, this is being written by a novice.
2 East Melbourne, Australia, for the curious
3 To do this, knowledge of CGI, HTML, a scripting language and mail is required.
I could have used some HTML editor but as it was, I decided that I could not handle learning another new package so I did it by hand, which was not too bad as what I wanted was not too complicated.

2) Installing the web server.
   In Linux, it was very easy. When I was installing Linux, the option to install Apache was ticked and when I typed in "localhost" as the URL to Netscape, it displayed the Apache page saying that if I saw that page, everything was A-OK! Whoo-hooo... so far so good. (See http://www.apache.org for more details). You'll probably need to be user root (the superuser) to do the install.

3) Writing the web page.
   I think the Javascript is semi-obvious. There are several functions for data input verification (ValidLength() and ValidEmail()). MenuHeader() displays the header part of the page. Each call to MenuEntry() displays an input row. In this case, it is called 4 times, once for each lunch order item.

```html
<! doctype html public "-//w3c//dtd html 4.0 transitional//en">
<HTML>
```
<HEAD>
<title>Colloqui Lunch Orders</title>
</HEAD>

<SCRIPT language="JAVASCRIPT">
//global variable for error flag
var errfound = false;

//function to validate by length
function ValidLength(item, len) {
    return (item.length > len);
}

//function to validate an email address
function ValidEmail(item) {
    if (!ValidLength(item, 5)) return false;
    if (item.indexOf('@', 0) == -1) return false;
    return true;
}

// display an error alert
function error(elem, text) {
    // abort if we already found an error
    if (errfound) return;
    window.alert(text);
    elem.select();
    elem.focus();
    errfound = true;
}

// main validation function
function Validate() {
    errfound = false;
    if (!ValidLength(document.order.username.value, 0))
        error(document.order.username, "Need value for name.");
    if (!ValidLength(document.order.email.value, 0))
        error(document.order.email, "Need value for email.");
    if (!ValidEmail(document.order.email.value))
        error(document.order.email, "email incorrect. Need an @.");
    return !errfound; /* true if there are no errors */
}

// this displays the menu header
function MenuHeader() {
    today = new Date();
    document.write("<B> Date : ", today, " <br>");
    document.write("<INPUT TYPE="HIDDEN" NAME=date1 VALUE="", today, "">");
    document.write("<BR>");
    document.write("<TABLE BORDER="0">");
    document.write("<TR>");
    document.write("<TD><B> First Name (lower case): </B></TD>\n"<TD><INPUT TYPE="text" NAME="username" SIZE=32></TD> <br>");
    document.write("<TR>");
    document.write("<TD><B> Email : </B></TD>\n"<TD><INPUT TYPE="text" NAME="email" SIZE=32></TD> <br>");
    document.write("<BR>");
    document.write("<TABLE>\n"<TR>");
    document.write("<BR>");
}

// This displays the menu entries.
// This would probably make more sense if one looks at the displayed form
// theNumber is displayed on the form.
// theOrder is the variable this order will be associated with.
// theGroup is the variable rice type will be associated with ie fried/steamed.
function MenuEntry(theNumber, theOrder, theGroup) {
    document.write("<B>\n"<SELECT NAME="", theOrder, "> SIZE=6 >");
}
The most tricky lines would be the ON-SUBMIT statement. There are 2 ways a web page can communicate to a CGI script. These are GET and POST. In a nutshell, GET sends the information as part of the URL (eg. you might have seen some URLs which resemble http://localhost/script.cgi?param1=value1&param2=value2 in your surfing of the internet).
The form information when POST is used get sent via the standard input. i.e. the CGI script needs to read in standard input and then parse the input read to separate out the various parameters.

It is generally accepted that POST is better (more robust, not limited by the max char limit of shell used). In extraction of the data, the methods differ according to whether POST or GET is used but Python hides this from you (which is good).

**Viewing lunch.html via the web server.**
I then placed the file in the directory /home/httpd/html.

$ cp lunch.html /home/httpd/html

(This is the default place Apache looks for html files. It can be configured to look elsewhere.) Then you can see what lunch.html looks like by browsing "http://localhost/lunch.html" using Netscape (or any browser).

4) Writing the CGI script.
Python comes distributed with the Red Hat Linux distribution (see http://www.python.org). After consulting the Python documentation, which also came with the system, my first script looked like the following (Figure 3). It is essentially a cannibalised version of an example found in the Python documentation. To make use of this script, you'll need to point the CGI script outlined in the ACTION statement in the HTML file to this script instead i.e. change the cgi script specified in the ACTION statement from lunch.cgi to first.cgi.

**Testing the installation.**
I then copied first.cgi to the directory /home/httpd/cgi-bin

$ cp first.cgi /home/httpd/cgi-bin

Essentially, I interrogated all the variables sent to the script by the form and printed it back out. All output printed out will be displayed by the browser.

```
1 #!/usr/bin/env python
2 # first.cgi
3 import cgi  # import the cgi module
4 5 print "Content-Type: text/plain\n\n"  # necessary for the browser
6 7 lunchForm = cgi.FieldStorage()  # retrieve the values
8 9 for name in lunchForm.keys():
10    print "Key= " + name + " Value= " + lunchForm[name].value + " ">
11 12 print "bye."
```

**Figure 3. Listing of first.cgi**

When the Go button on the lunch html page is clicked and the first.cgi script is activated, the output returned to the web browser looks like the following.
You will notice that the keys found in the CGI script correspond to the variables I used in lunch.html.

Once I got this simple script working, I then expanded it to do what I wanted (Figure 5). The Python code quite straightforward and is quite self-explanatory (I think). It imports the `cgi` module. It then calls the member function `FieldStorage()` of `cgi`. Whether the information is sent using the `GET` or `POST` method is hidden from you. That's how all the information sent by the web page is retrieved. The information can then be extracted by accessing `lunchForm`. 
The body of the mail sent is then constructed via a series of writes to sendmail, the Unix sendmail program.

I decided to mail the lunch order to user lunch@localhost. An alias can be inserted in file /etc/aliases.

lunch: chai@localhost

where user chai@localhost is organising the orders. This way, if the lunch organiser gets changed, the file /etc/aliases needs to be changed and not the CGI script. (newaliases needs to be run for changes to /etc/aliases to take effect).

Easy, eh? Well, it could be much much worse.

**Hooking everything together**

I then copied lunch.cgi to the directory /home/httpd/cgi-bin.

$ cp lunch.cgi /home/httpd/cgi-bin

Opened up netscape, typed in http://localhost/lunch.html as the URL, filled in the form, selected my order and clicked on "Go".

Some time later, an e-mail would arrive outlining the order.
This is what the mail sent out by the CGI script looks like.

```
From nobody@localhost Wed Apr 26 11:01:50 2000
Delivered-To: ccang@localhost
Date: Wed, 26 Apr 2000 11:01:48 +1000
To: lunch@localhost
From: chai <calcium@altavista.net>
Subject: loi loi

Sender: nobody@localhost

SourceIP 194.118.1.1
calcium@altavista.net
Wed Apr 26 11:01:01 GMT+1000 (EST) 2000
cchai wants
1. L39 with Steamed rice.
2. NONE with NA rice.
3. NONE with NA rice.
4. NONE with NA rice.

Special Instructions: Make it extra yummy please!
```

Figure 6. Received E-mail of Lunch Order

This is what the web page looks like after the request has been sent.

Figure 7. Snapshot of resulting web page after lunch order made.

Conclusion.

Is the on-line system better than write-order-on-scrap-paper method? Debatable, but it certainly is more fun (was for me).

Improvements? The web page is geared towards an individual making an order as opposed a person ordering for multiple people. In hindsight, the web page could have been laid out with the latter in
mind, and being a superset of the former, would satisfy those requirements as well. A simple compromise could be having a multiples box which would allow a person to order more than 1 item of the same dish per order row. In the current scheme, this would still only allow 4 different orders per email. So make it 10? 20? How long is a piece of rope? (paraphrased to make it more sinister). A design problem left to the reader.4

I suppose I could also hook it into an SQL database (http://www.mysql.org) and print out a histogram of the past orders of this particular person. A by-product of using a database is that one could print out reports, ie what is to be ordered for that week.

I suppose if there is enough interest and if I have enough time, I'll write a second part to this where the CGI script would interact with a SQL database and return html code to display a frequency list. And with perhaps some cookie interaction.

And finally on a personal note, I've seen the future and it is Python. Look it up (and JPython too!).

**Disclaimers:**

I have no financial stake in the restaurant5. However, if you do decide to order food from there, mention my name, Chai, and if this happens often enough, I might get a free meal there some day, that is if the cook keeps his word.

**References:**

Lutz, M & Ascher, D, Learning Python, O'Reilly (1999)  
Good book.

van Rossum, G et al, Python Reference Manuals, [http://www.python.org](http://www.python.org)  
Remarkably complete. I only bought the books because it was easier to handle.

Lemay, Laura, Javascript, SAMS Publishing  
Was not altogether happy with this book. But used it nevertheless.

---

4 8-)

5 Product placement spots still available for my next article.
The Web Based Supply Chain Management (WBSCM) system is an integrated, Internet-based commodity acquisition, distribution, and tracking system built on Systems, Applications and Products in Data Processing (SAP) commercial software and an Oracle platform. The eight domestic programs, including the National School Lunch Program (NSLP), the Emergency Food Assistance Program (TEFAP), and Food Distribution Program on Indian Reservations (FDPIR), serve over 30 million Americans and are administered through 98 State Distributing Agencies (SDAs), supporting over 100,000 Recipient Agency (RA) school districts, food banks, and feeding centers, and 110 Online Ordering Systems and Consumer Acceptance - Using Box Lunch Order Systems as a Case Study. Ling-Hui Yang1, Ting-Sheng Weng2 & Hsiang-Pang He3 1 Department of Food and Beverage Management, Tatung Institute of Commerce and Technology, Chiayi County, Taiwan 2 Department of Business Administration, National Chiayi University, Chiayi County, Taiwan 3 Department of Exercise and Health Promotion, Chung Chou University of Science & Technology, Changhua County, Taiwan. Based on the findings, managerial implications are discussed in this paper and directions for future research are also highlighted. Toward an understanding of the behavioral intention to use a web site.