SY306 Lab Six

CGI with Python

Introduction

This week we started CGI programming. Now, you will go back to the HTML sign up form you created for your website, and finally put some computation behind it to really keep track of users.

SETUP - DO THIS FIRST!

You must create a folder in your `public_html` called "Lab06" (without the quotes) and store your work in that directory. Although you originally created your form in Lab01, **you should copy your work from Lab04 instead**, in order to benefit from your later CSS additions.

Tip: an *easy* way to copy your Lab04 files to Lab06 is to run the following command from your `~/public_html$` directory:

```
cp -p -r Lab04 Lab06
```

PART A: VERY IMPORTANT: To allow the webserver to later create files when running a Python script, the webserver user needs to have extra permissions on your Lab06 directory (and others as needed). To enable this, open a terminal window on your Unix machine, SSH into **`midn.cyber.usna.edu`**, and type the following (ensure your mALPHA is used):

```
chmod 711 ~/
chmod 700 ~/public_html
setfacl -R -m u:www-data:rwx ~/public_html
setfacl -d -R -m u:www-data:rwx ~/public_html
setfacl -R -m u:m21xxxx:rwx ~/public_html
setfacl -d -R -m u:m21xxxx:rwx ~/public_html
find ~/public_html -type d -exec chmod 775 {} \;
find ~/public_html -type f -exec chmod 664 {} \;
find ~/public_html -type f -iname "*.py" -exec chmod 755 {} \;
```

You will have to repeat this step every time you create a new directory where you want to allow the webserver to create files!

PART B: IMPORTANT: In Python, indentation is very important. Make sure your editor uses spaces and not tabs for indentation, and the line endings are UNIX (LF, not CRLF)

THE ACTUAL LAB:

Read the entire lab so you see the requirements and know what is coming.

1. You should have the file signup.html (or whatever file contains your signup form) from Lab04 (or earlier), copied into Lab06. Your form must contain at least one element of the following input types: text, password, checkbox or radio buttons, textarea, submit. Your form will likely contain more than that. If your form does not contain the required input types modify it so that it does.
2. [5 points] **Form:** Modify the form (in your Lab06 directory) so that when you click the submit button, it invokes a new CGI script process_signup_start.py that you will write next. To make testing easier for this lab, change the "method" in the form to `GET` instead of `POST`. You should change it back to `POST` before turning in the lab.
3. [15 points] **Script:** Create the process_signup_start.py Python script that reads in the values submitted by the user and prints out a thank you message. Your script should also output the value of one of the fields back to the user (you will do more later, but do this to get started). If you need help with debugging, see the "Additional Hints/Clarifications" section below.
4. [15 points] **Validity checks:** Create a copy of process_signup_start.py and call it process_signup_validate.py. Modify process_signup_validate.py to validate *some* of the input that your form provides to your CGI program. If an error is detected, your program should state explicitly what the error was, and tell the user to hit the back button and try again (see extra credit for a better approach). You may find it useful to go back to Lab01/signup.html, fill in some values, and click submit to see how your data is received by the CGI program we gave you earlier. For the validation you must at a minimum check the following (you can of course do more if you like):

   - For your first (or only) text field, ensure it is filled out (not empty)
   - For your first (or only) textarea, ensure it is filled out (not empty)
   - For your first (or only) set of checkboxes or radio buttons, ensure at least one checkbox /radio button is selected. See the "Hints" section for a discussion about checkboxes.
   - For your first (or only) password field, ensure it is filled out (not empty)
   - For your first (or only) password field, ensure that the password contains at least one number.

   Hint: To bypass the HTML checks and JavaScript checks that you have in your form and test your Python code, you should modify the parameter values in the URL, rather than use the form. You could also use a program like Postman which would allow testing with either `GET` or `POST`. Do not expect assistance if you use this.
5. [10 points] **Confirmation:** Create a copy of process_signup_validate.py and call it process_signup_confirm.py. Modify process_signup_confirm.py code so that, if the variables pass all the validation tests above, the program prints out a user friendly

confirmation. This confirmation should display the value of all the variables, except passwords, that were provided, in a user-friendly manner. **A page with a raw list of variables and their values is not so friendly** -- you should at least have a reasonable title, some welcome text, then a reasonable confirmation with the values. Imagine this was on your website and you wanted to present a reasonable appearance to someone that just submitted your form. Example: "Thank you for signing up with us. Your information was recorded as follows:"

6. [15 points] **USERS file:** Create a copy of process_signup_confirm.py and call it process_signup_record.py. Modify process_signup_record.py so that it records information about valid users to a file called USERS.txt. You will want to append to this file (so it contains all the users in your system). You should write to the USERS.txt as follows:

- If the values provided in the sign up form do not pass all the "Validity" tests above (e.g. if some required values are missing), then write nothing to the USERS.txt
- If the values pass your "Validity" tests then write all the parameters provided by the user to USERS.txt file. The values for a single user should all be on a single line, use a tab ("\t") as a separator.

Here's an example of how a simple USERS.txt file might look after 4 users submitted forms that passed the Validity tests. (Tabs, spaces, and new lines are shown for clarity.)

```
m211111 ⟶ Password123 ⟶ John·P.·Jones
⟶ 5 ⟶ Alpine·Skiing ⟶ West·Virginia
¬
m212222 ⟶ qwerty321 ⟶ Chesty·P. ⟶ 33
⟶ Luge ⟶ South·Dakota ¬
m213333 ⟶ 123456!@#$%^ ⟶ J.P.·Sousa
⟶ 6 ⟶ Curling ⟶ Idaho ¬
m214444 ⟶ qwertyyuiop1 ⟶
Francis·S.·Key ⟶ 71 ⟶
Freestyle·Skiing ⟶ Massachusetts ¬
```

7. [15 points] **Username check:** As users sign-up for your website, you should ensure that you have some unique identifier for each user, otherwise it will get very confusing for everyone. You can use a username, email, id, or something else for that. Whatever you are using must be unique. Create a copy of process_signup_record.py and call it process_signup_check.py. Modify your process_signup_check.py as follows:

1. Check that there is no entry with your chosen identifier already in the USERS.txt file.
    - If you are using for your identifier the username field value , check that there is no user with that username in the USERS.txt file.
2. If a user with that identifier exists, your script should not write that user's information to the USERS.txt file. Your script should produce an output that states explicitly what the error was, and tells the user to hit the back button and try again (see extra credit for a better approach).

3. If the username is unique, and the previous validity checks are passed, the confirmation message including the provided values you created before should be displayed, and the information for the user should be added to the USERS.txt file.

8. [15 points] **Security:** Create a copy of process_signup_check.py and name it process_signup_safe.py. Research and implement a CGI/Python/HTML module that can be used in your Python script (process_signup_safe.py) to escape character inputs before either displaying them back in the confirmation message or writing them to USERS.txt.

- At a minimum `&`,< and > need to be escaped to HTML safe sequences using a class method.

9. [10 points] **Documentation**: Ensure you have appropriate comments in your Python script.

10. **Important final steps:**

1. Create three links in your top-level default.html page under the heading "Lab06":

    1. Under the name "Form", make a link to your Lab06/signup.html page or whatever name your page with the sign-up form has. Make sure the action attribute of the form on that page is set to process_signup_safe.py.
    2. Under the name "Good submission" make a link to your process_signup_safe.py file with all of form variables specified in the URL, such that variables all validate and the username is unique. Hint: if your form uses the GET method (change this temporarily), then you can create the needed URL for this by filling out your form correctly and hitting submit
    3. Under the name "USERS.txt" make a link to your USERS.txt file

2. Make sure you change the "method" for your form back to `post`

Extra Credit

For a nominal amount of extra credit do some/all of the following:
(NOTE: saving a backup copy of your working lab is recommended before starting on this)

1. If your program finds a validation problem with an input (such as a missing value or a duplicate username), a much better way to handle this is to have your CGI program regenerate the form with all of the values provided by the user filled in, and values that had a problem highlighted. Of course there should be a submit button so the user can modify the values and resubmit back to the CGI program.

2. Write a new CGI program (in Python) called signup_report.py that reads your USERS file and generates a summary report of the submissions. Be sure that your USERS.txt has enough data in it to make this report at least a little interesting. Provide a link to this from your default.html page under the heading Lab06.

Deliverables

1. You should have all the pieces working as described in sections 1-9 above.
2. You should have the three links in default.html that are described in section 10 above.
3. You have created the following Python files in your Lab06 directory:

    1. process_signup_start.py
    2. process_signup_validate.py
    3. process_signup_confirm.py
    4. process_signup_record.py
    5. process_signup_check.py
    6. process_signup_safe.py

4. All of your files should be in a folder called "Lab06" (without the quotes) on your public_html. **Your instructor will assume that your web pages are viewable at http://midn.cyber.usna.edu/~m21xxxx/Lab06/signup.html** (or another name). You may want to check that this URL is viewable and that everything works correctly *from a computer where somebody else is logged in*.
5. **Turn in (due before lab on Friday):**

    1. **Paper submission:** turn in the following at the beginning of Lab07, stapled together in the following order (coversheet on top):

        1. A completed assignment coversheet. Your comments will help us improve the course.
        2. A printout of the source of your process_signup_safe.py file.
        3. A screen-shot of your browser window showing the confirmation message you displayed after a user submitted a form that passed the validity checks.

    ii. **Electronics submission:** Submit all Lab06 files via the online system: `submit.cs.usna.edu` by the following deadlines:

        - Section **4341**: 23:59, Wednesday, February 20th
        - Sections **1151, 3351, 5551**: 23:59, Thursday, February 21st

Additional Hints/Clarifications

- **Debugging CGI Python:** You will write a CGI Python script for this lab and more later. It is important to be able to debug your programs. Here are some instructions for that:

    1. Make sure your scripts for SY306 are in the correct folder. In particular for this lab, your scripts should be in your Lab06 folder.
    2. Let's say that you want to debug a script called process_signup.py in your Lab06 folder. From the terminal window:

```
cd public_html/Lab06
```

If your program expects no CGI parameters, just type:

```
./process_signup.py
```

this will execute your file. Any syntax errors will be reported to the screen.

If your program expects some parameters, for example "name" and "age", type something like:

```
./process_signup.py "name=Fred&age=27"
```

3. If the file does not execute at all, you probably have Windows line endings instead of Unix line endings, or maybe you have the wrong shebang line. Fix those (see the example in your notes).

4. Executing the file might identify a few syntax errors in the program. Fix them. Re-run Python from the command line until you get no more errors.

5. Try fetching the correct URL in the browser: http://midn.cyber.usna.edu/~m21xxxx/Lab06/process_signup.py?name=Fred&age=72 (this is just a sample URL, you will likely have different parameters)

6. To find logic errors, try print statements and print the values of the different variables

7. Lastly, to aid with troubleshooting, you can import and enable the CGI traceback module which activates a special exception handler that will display detailed reports to the browser if any errors occur. The following lines provide us that functionality:

```
import cgitb      # import CGI traceback module
cgitb.enable()    # enable CGI traceback module
```

- **Checkboxes** are interesting because more than one can be checked. If you give the same name (in HTML) to all your checkboxes, for example `mychecks`, and write something like this in Python:

```
checks = form.getvalue("mychecks");
```

it will cause the script to crash because there is more than one field with the same name. Instead, you can use the `getlist()` function which will return a list with the values of all of the `mychecks` checkboxes that were checked. The list will be empty if no checkboxes were selected. Sample code:

```
checks = form.getlist("mychecks");
```

- **Visit the SY306 Python Debugging Guidance link for additional troubleshooting and debugging tips.**

External Resources

- For more information on CGI support in Python, please visit this link.
- For Python REGEX documentation, please visit this link.