

## Lesson 5 Outline – Script Authoring

This is really the nuts and bolts of CGI programming. If you want to be able to write your own scripts, study this lesson thoroughly.

- I. Using forms with scripts
  - A. Hopefully it is clear at this point that forms and CGI script work hand in hand. To do much meaningful work using CGI, you will need some type of form to send input to the script. Review your favorite HTML book to refresh your memory on the syntax for the various types of fields supported.
  - B. `<FORM ...>`
    1. METHOD => GET or POST
    2. ACTION => URL of the CGI script to process the form data
  - C. `<INPUT TYPE="text" ...>`
  - D. `<TEXTAREA ...>`
  - E. `<INPUT TYPE="radio" ...>`
    1. Note that to behave properly, all related radio buttons must have the same "NAME". This is a frequent mistake.
    2. I can't break my 6<sup>th</sup> grade outlining rules by having a 1. without a 2., but I don't have anything else to say here ☺
  - F. `<SELECT ...>`
    1. Can be either one of many or multiple options behavior
    2. If "MULTIPLE" is enabled, you will have to collect the related information into an array or a string of values when decoding the script input
      - a. CGI.pm creates an array for you in this case
      - b. cgi-lib.pl will create a null ("\0") delimited list (I frequently rewrite my copy of cgi-lib.pl to use commas instead of the null character – this just makes it easier to split the string into parts)
  - G. `<INPUT TYPE="checkbox" ...>`
    1. Note that you can handle checkboxes two ways (the book only mentions the first)
      - a. You can give all radio buttons separate names, and the value for a checked box will just be "on" ("on" is the default; the book assigns the value to "1" instead).
      - b. You can give related radio buttons the same name and change the value to a meaningful expression. (For example: `<INPUT TYPE="checkbox" NAME="color" VALUE="red">`)
    2. When decoding the input data for the 2<sup>nd</sup> approach you need to collect the related information into an array or a string of values (see notes on multiple select above)
  - H. `<INPUT TYPE="hidden" ...>`
    1. Hidden fields are the work horse of complex CGI scripts. You can use them to pass information between different scripts, which is particularly useful. More on that in lesson 6.
    2. These should not be used to store sensitive information (ex. Credit card numbers) since they are visible with the HTML source code from the browser.
  - I. `<INPUT TYPE="submit" ...>`
    1. You can change the name of the submit button if you want to; this enables you to have several submit buttons on a single form, and your CGI can take different actions by looking for the various names.
    2. You can use an image for the submit button by changing this to `<input TYPE="image" BORDER="0" NAME="name_of_button" SRC="/yourimage.gif" VALUE="Whatever">`
- II. Script responsibilities
  - A. Parse input (from GET or POST)
    1. Manually
    2. cgi-lib.pl

3. CGI.pm
  - B. Return content header followed by a blank line (before you send any other data!)
  - C. Return main body
    1. Text (remember that the output can be another form)
    2. Binary data (ex. images)
  - D. Error handling
    1. Remember that the perl “die” function will not exit gracefully for CGI scripts. It will result in a 500 error (and you’d have to run from the command line to see the die message) unless you are using something like CGI:carp
    2. It is often useful to have a “debug” flag in your script that prints additional information about what the script is doing
- III. Using Libraries
- A. Many of the fundamental tasks in CGI scripts are laborious and make the script prone to security errors. It’s wise to use one of the pre-written CGI libraries
  - B. cgi-lib is the previous standard and is still widely used. Make sure you review it’s weaknesses.
  - C. CGI.pm is the most robust solution.
    1. Simplifies decoding form data
    2. Has shortcuts for printing the content header and various HTML constructs
    3. Improves error handling
    4. Functions for various useful tasks – cookies, etc.
    5. Does have a few limitations (slower load time, possible denial of service attacks)