# CGI and Perl

**CSE/ISE 102: Intro to Web Design (section 02)**
**Stony Brook University**

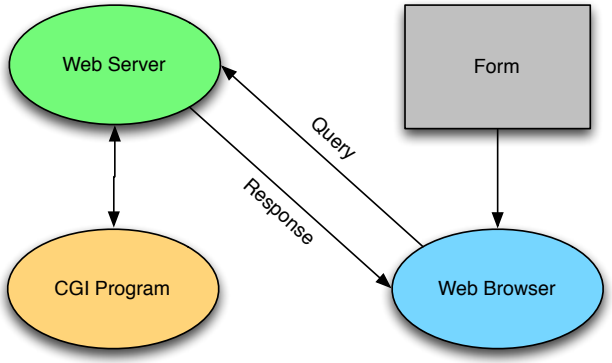
## What is Perl?

- **P**ractical **E**xtraction and **R**eport **L**anguage
  - developed in 1987 by Larry Wall
  - a programming/scripting language
- A perl script consists of a sequence of commands
  - these commands are executed sequentially by the Perl interpreter

## What is CGI?

- The Common Gateway Interface
  - controls the way a Web server interacts with external programs (scripts)
- CGI programs can be written in any language
- This is only one of the ways a Web server can apply customized processing to user data
  - e.g., ASP, PHP, Java servlets, server modules

## Form Processing



# The HTTP Protocol

- Hypertext Transfer Protocol
- Governs how the Web server and the client exchange information
- Uses TCP/IP for a reliable bidirectional communication channel

# HTTP Event Sequence

1. Connection: client opens a connection to a server
2. Query: client sends HTTP request to server
3. Processing: server receives and processes request
4. Response: server sends an HTTP response to client containing requested resource or result
5. Transaction finished: connection may be closed or kept open for a follow-up request

# HTTP Message Format

initial line (different for query and response)
HeaderKey1: value1
HeaderKey2: value2

Optional message body containing query/response
(Amount and type of data in the body are specified in headers)

# The Query Line

- Query line
  - query method name, a server-side path (URI), and HTTP version number
  - `GET /path/to/file/index.html HTTP/1.1`
  - `POST /cgi-bin/script.cgi HTTP/1.1`
  - `HEAD /path/to/file/index.html HTTP/1.1`

# The Response Line

- Version number, status code, and text description of status
  - `HTTP/1.1 200 OK`
  - `HTTP/1.1 404 Not Found`
- There are other status codes as well


# Sample POST Query

```
POST /cgi-bin/register-user.cgi HTTP/1.1
HOST: www.SymbolicNet.org
From: jdoe@great.enterprise.com
User-Agent: Netscape 6.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 132

name=John+Doe&address=678+Main+Street&...
```

# CGI Program Outline

1. Determine request method and receive input data
2. Decode and check input data
3. Perform tasks
4. Produce output (usually as an HTML document)

# Output Formats

- HTML
  - Content-Type: text/html
    (empty line)
    HTML page
- URL
  - Location: url

# Sparky & CGI

- First, we need to set up your Sparky account
  - add a new directory called “cgi-bin” to your www directory
- In all HTML forms, you will reference your CGI programs using the URL:
  - http://www.sinc.sunysb.edu/cgi-bin/cgiwrap/rmckenna/???.cgi
  - instead of rmckenna use your Sparky login name
  - instead of ??? use the name of the cgi script

# File Permissions

- Use the Unix command 'chmod' to set file permissions
  - owner, group, and other/world
  - Each type of permission is a number in octal that corresponds to the sum of the permissions desired
  - 4 = read, 2 = write, 1 = execute
- e.g., `chmod 755 myFile.txt`

# CGI Script Permissions

- Your CGI scripts should have permissions 755
- This lets you (the owner) read, modify, and execute them, but everyone else can only read and execute them
  - in particular, this lets the Web server execute your scripts

# Testing Your Scripts

- To test your scripts before deploying them, run them from the Unix command line:
  - perl hello.cgi
- Put form data after the script name:
  - perl hello.cgi name=Mike email=foo@bar.com
- See what output your script produces

# A Toy CGI Program

- Consists of two parts:
  - An HTML form
  - A CGI script that receives/processes form data
    - Our sample CGI script will be written in Perl

```
<form method="post" action="http://www.sinc.sunysb.edu/cgi-
bin/cgiwrap/???/hello.cgi">
   <table width="400">
    <tr> <td><label for="name">Full Name:</label></td>
         <td><input id="name" name="name" size="35" /></td>
    </tr>
    <tr> <td><label for="email">Email:</label></td>
         <td><input id="email" name="email" size="35" /></td>
    </tr>
    <tr> <td></td>
         <td><input type="submit" value="Send" /></td>
    </tr>
   </table>
</form>
```

This indicates that the Perl interpreter should be used to process this script

```
#!/usr/bin/perl
## hello.cgi -- a toy CGI program

use CGI qw(:standard); ## cgi perl module

var $name = param('name');
var $email = param('email');
```

Use # to indicate the start of a comment

The Perl module *CGI* makes it easy to write Perl CGI scripts

These lines declare two variables and assign them values from the form data

\r\n represents a newline; we use two of them to add a blank line between the content type and the body of the response

```
## send response to standard output
print "Content-type: text/html\r\n\r\n";
print <<END;
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Hello</title>
</head>
<body>
<h1>Hello, $name</h1>
<p>The e-mail address you submitted is</p>
<p>$email</p>
</body>
</html>
END
```

This tells the script to print out everything until it sees the label "END"

We replace the variable names with their actual values in the output

## CGI Environment Variables

- Allow the the Web server to communicate with the CGI program
  - these are set when the program starts to run
  - SERVER_NAME, SERVER_ADDR, PATH, GATEWAY_INTERFACE, SERVR_ADMIN, SERVER_SIGNATURE, DOCUMENT_ROOT, SERVER_SOFTWARE

## Per-Request Variables

- SERVER_PORT, SERVER_PROTOCOL, TZ, REQUEST_URI, REQUEST_METHOD, CONTENT_LENGTH, CONTENT_TYPE, QUERY_STRING, SCRIPT_NAME, SCRIPT_FILENAME, REMOTE_ADDR
- There are more, but these are the most commonly-used ones
- To access these in Perl, use $ENV{*var_name*}

# A Quick Perl Tutorial

## Script Basics

- Perl scripts should be written using a plain text editor (TextWrangler, Notepad, Emacs, vim, pico)
- Perl scripts normally use a .pl extension
- Each Perl statement ends with a semicolon
- Statements execute sequentially

# Perl Variables

- Three types: scalar, array (list), and association array (hash)
- *Scalar* variables have a $ prefix, and hold a single value of any type
  - $var = 'a string'; # a quoted string
  - $x = 12;
  - $abc = "$var$x"; # 'a string12'

# Single vs. Double Quotes

- Single quotes indicate a string of characters
  - contents are printed literally
- Double quotes also indicate a string
  - substitutions are made for variable names
- Use backslashes to include literal quotes
  - \' and \" produce ' and " respectively

# Arrays

- Use @ in front of an array variable
- Put starting values in a parenthesized list
  - @arr = ("aa", "bb", "cc", "dd");
- Use $ to assign values inside an array
  - Positions are numbered starting with 0
  - $arr[2] = 76; # store 76 in third position
- Assign an array to a scalar to get its length

# Aside: Perl Output

- Use `print` to display output
  - `print` does not add a newline at the end
- Use \n to add a newline to output:
  - print "$myVar\n";

# Perl Hashes

- A hash is an array with an even number of elements (key-value pairs)
  - Notation: (key1 => value1, key2 => value2, etc.)
  - keys serve as indices for the values
- `%asso = ("a" => 7, "b" => 11);`
- `print "$asso['a']\n" # prints 7`
- `$asso{'c'} = 13; # adds 13 to asso`

# Arithmetic and String Operators

- Standard arithmetic: + - * / ** %
- $a = $b . $c; # concatenates b and c as strings
- $a = $b x $c; # repeats $b $c times
- Use = to assign values
- Boolean operators: == != eq ne && || !

# Conditional Statements

- ```
  if ( test )
  {  ...  }
  else     # optional
  {  ...  }
  ```
- `elsif` is also available:

```
if ( test ) {  ...  }
elsif ( test2 ) {  ...  }
else {  ...  }
```

# Conditional Execution

- Add a *modifier* at the end of a statement to make its execution conditional
- e.g., `statement if ( test );`
- e.g., `statement unless ( test );`
```

# Standard I/O in Perl

- STDIN = standard input (keyboard)
  - abbreviated as <>
- STDOUT = standard output (to screen)
- A Perl CGI script receives data by reading STDIN
- read(STDIN, $input, num_characters});

# More Perl I/O

- We can read one line at a time from STDIN:
  - $var = <STDIN>;
  - $var = <>;
  - This leaves the newline at the end
- Use 'chomp' to get rid of trailing newlines:
  - chomp($var);

# Iterations

- Four forms: foreach, while, do-while, and for
  - while ( test condition ) {  statements  }
  - do {  statements  } while { condition };
  - for ( initialization; test; update) { statements }
  - We'll skip foreach loops for now

# Perl Subroutines

- Start with the sub keyword
- sub name
  {
  a sequence of statements
  }
- Declare subroutines before using them
  - predeclare them using 'sub name;'
- Call subroutines by their name, followed by ( )

# Subroutines and Values

- Input arguments are stored in a special array named @_
- Use 'return' to send back a value
  - e.g., return $answer;
  - this also immediately terminates the subroutine

# Next Time

- More Perl examples