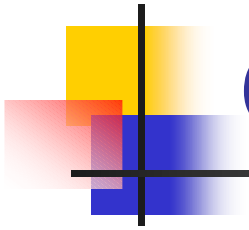
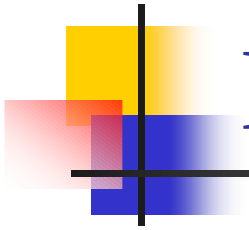


CSC 309 Tutorial - CGI

TA: Lei Jiang

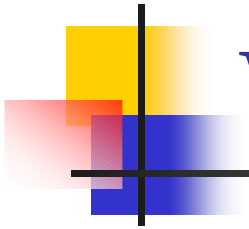
March 3, 2003



Introduction

- A HTML document that a Web server delivers is static, which means it doesn't change.

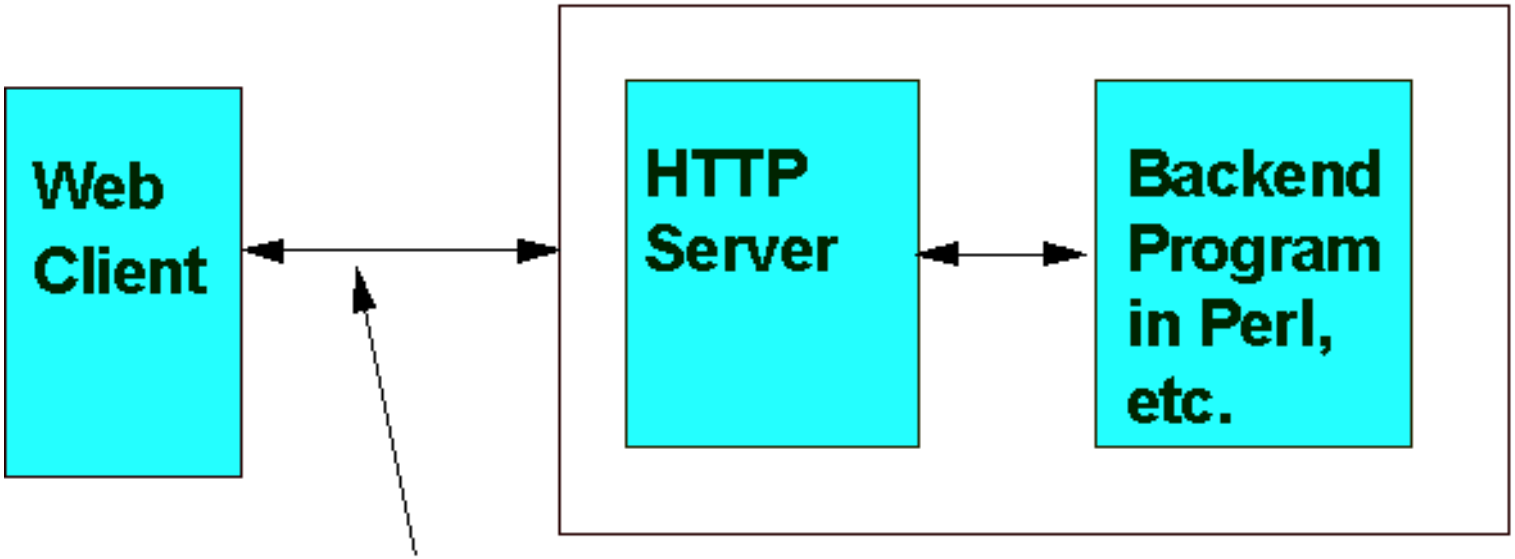
- A CGI program, on the other hand, is executed in real-time, so that it can generate and deliver dynamic HTML documents.



What is CGI?

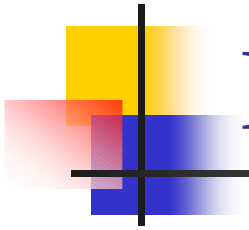
- The Common Gateway Interface, or CGI, is a standardized specification
  - For interfacing external applications with Web servers via the Hyper Text Transfer Protocol (HTTP).
- The specification is usually called the Standard CGI Interface
  - because it uses Standard Input (STDIN) and Standard Output (STDOUT) to read and send the data.

How it works?



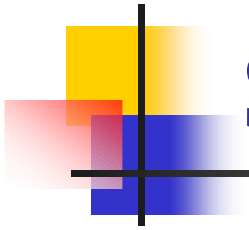
Ascii

Communication



Example uses for CGI

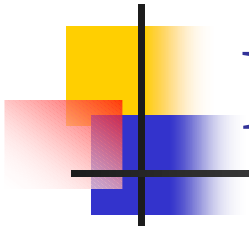
- Forms
- On the fly pages
- Database Interaction
- Logging/Counters
- Animation



Simple PERL Example

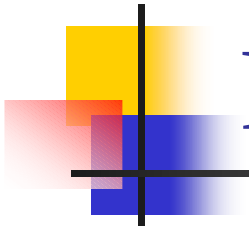
```
#!/usr/local/bin/perl
 #This is a html file
 print"Content-type: text/html\n\n";

 #The html output
 print "<html>\n";
 print "<head><title>Hello World !!!</title></head>\n";
 print "<body>\n";
 print "<h2>Hello World !!!</h2>\n";
 print "</body>\n";
 print "</html>\n";
 exit;
```



PERL Basics

- PERL (Practical Extraction & Reporting Language) is…
  - a text processing language
  - a natural to use for CGI
  - similar to Shell Script
  - interpreted
- PERL is Not…
  - a platform-dependent language

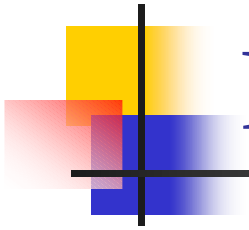


PERL Basics

- Every PERL program should begin with the first line like:

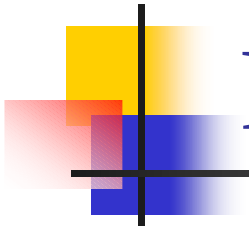
  #!/usr/local/bin/perl

  - Tell the location of your perl binary
- Everything after that first line is Perl,
  - except for lines beginning with #, which are comments.
- All perl statements end in a semicolon (;), like C.
- Don’t need to declare variables before using them.



PERL Basics – Data Types

- Three are three data types in PERL
  - scalar, array and hash
- A **scalar** variable contains a single value
  - preceded by a “$”
- A **array** variable contains an ordered list of values indexed by a positive number
  - preceded by a “@”
- A **hash** variable contains an unordered set of key/value pairs indexed by a string
  - preceded by a “%”



PERL Basics – Array

- Array elements can be accessed by $

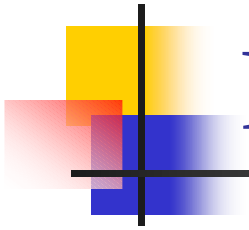
  @browser = ("NS", "IE", "Opera");

  $browser[2]="Mosaic";

- Array can be append at run-time

  @browser = ("NS", "IE", "Opera");

  $browser[3]="Mosaic";



PERL Basics – Array Functions

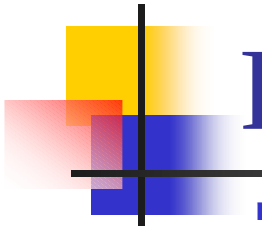
- **Splice**
  - *delete* or *replace* elements within the array.
  - delete example:

    @browser = ("NS", "IE", "Opera");

    splice(@browser, 0, 2);
    - starts splicing at list number zero, and splices two elements in a row.
  - replace example

    @browser = ("NS", "IE", "Opera");

    splice(@browser, 1, 2, "NeoPlanet", "Mosaic");
    - Replace the 2nd and 3rd elements with "NeoPlanet", "Mosaic"



PERL Basics – Array Functions

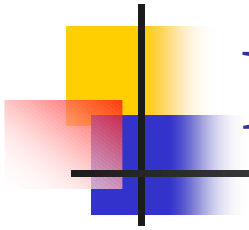
- **Unshift/Shift**
  - *add* or *delete* an element from the left side of an array (element zero).
  - unshift example:

    @browser = ("NS", "IE", "Opera");

    unshift(@browser, "Mosaic");
    - add "Mosaic“ to the array
    - now @browser is ("Mosaic", "NS", "IE", "Opera")
  - shift example:

    @browser = ("NS", "IE", "Opera");

    $old_first_element= shift(@browser);
    - delete "NS" from the array
- **Push/Pop**
  - like unshift and shift, except add / delete from the right side of an array.



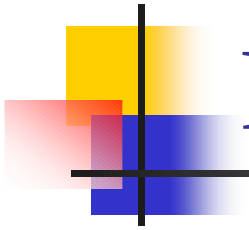
PERL Basics – Array Functions

- **Chop**
  - take the last character of each element in an array and delete it.
  - exampe:

    @browser = ("NS4", "IE5", "Opera3");

    chop(@browser);
    - Now, @browser is ("NS", "IE", "Opera")
- **Reverse**
  - reverse the order of the array elements.
  - example:

    @browser = ("NS", "IE", "Opera"); reverse(@browser);



PERL Basics – Array Functions

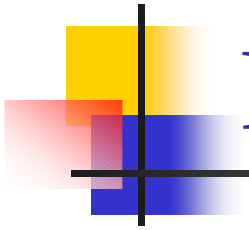
- Join
  - use a delimiter to separate array elements and output as a string.
  - example:

    @browser = ("NS", "IE", "Opera");

    $browsers = join(:, @browser);
    - Now, browsers contains “NS:IE:Opera”
- Split
  - create an array of elements by splitting a string using a delimiter
  - Example:

    $browser_list="NS:IE:Opera";

    @browser= split(‘:’, $browser_list);



PERL Basics – Hash

- Hash is also called associative array.
- Syntax:
  %array_name = ('key1', 'value1', 'key2', 'value2‘, …);
- Examples:

```
%our_friends = ('best', 'Don', 'good', 'Robert', 'worst', 'Joe');
$good_friend = $our_friends{'good'}; # use {} to access the hash values
$our_friends{'cool'} = "Karen";  # can add new key-value pair to the hash
delete ($our_friends{'worst'}); # delete the value associated with the key
```

PERL Basics – Quoting Rules

- **single quotes** (')
  - text placed between is interpreted literally
  - no variable expansion takes place.
  - to include an single quote ' in the string, you must escape it with a backslash '\'

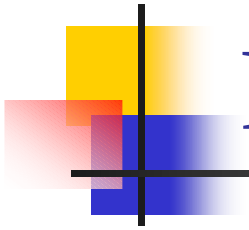
```
$course_title = ‘Programming on the Web';
$course = ‘CSC309 $course_title';
# the string value of $course includes the
# text "$course_title" - probably an error in this case.
```

PERL Basics – Quoting Rules

- **double quotes (")**
  - The double quote "interpolates" variables between the pair of quotes.

```
$course_title = ‘Programming on the Web';
$course = “CSC309 $course_title”;
# the string value of $course includes
# “CSC309 Programming on the Web “
```

  - to include an double quote " in the string, you must escape it with a backslash '\'



PERL Basics – Output

- **print** statement: prints the content between the quote marks to standard output.

  print "Your text and other things go here";

- If you want the content sent to the web browser to display, you need:

  print "Content-type: text/html\n\n";

  - This tells the browser the content will be text/html and to skip a line. The \n\n at the end is necessary.

PERL Basics – Output

- Remember to escape the quotes and other special characters, using backslash \

```
print "<A HREF=\"http://someplace.com\">Click Here</A>";

print "<A HREF=\"mailto:john\@pageresource.com\">E-mail Me</A>";
```

- Can use special print command to simplify your HTML output

```
#!/usr/bin/perl print "Content-type: text/html\n\n";
print <<ENDHTML;
<HTML> <HEAD> <TITLE>CGI Test</TITLE> </HEAD>
<BODY> <A HREF="mailto:john\@pageresource.com">Mail Me, or Else!</A>
</BODY> </HTML>
ENDHTML
```

  - don't need to escape quotes,
  - but other special characters still need to be escaped.

PERL Basics – Operators

Arithmetic Operators

Operator	Function
+	Addition
-	Subtraction, Negative Numbers, Unary Negation
*	Multiplication
/	Division
%	Modulus
**	Exponent

These operators are used to perform mathematical calculations on numbers. They are not used to combine strings.

Logical Operators

Operator	Function
&&	AND
||	OR
!	NOT

String Operators

Operator	Function
.	Concatenate Strings
.=	Concatenate and Assign

```
$word1="light";
$word2="house";
$full_string=$word1 . $word2;
print "If I had a $word1 and a $word2, would I be able to make a
$full_string?";
```

PERL Basics – Operators

String Comparison

Operator	Function
eq	Equal to
ne	Not Equal to
gt	Greater than
lt	Less than
ge	Greater than or Equal to
le	Less than or Equal to

Strings are equal if they are exactly the same.

```
$i_am="cool"; if ($i_am eq "cool") { ....more cool code.... }
```

The greater-than and less-than operators compare strings using alphabetical order.
Also, small letters are greater than capital letters.

```
$i_am="delightful"; if ($i_am lt "cool") { print "You are not very cool, dude."; }
```

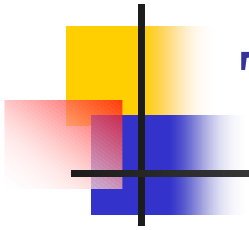
PERL Basics – Loop

- The foreach Loop
  - The foreach loop is almost always used with array.
  - Syntax:

```
foreach variable_name (array_name)
{
  code to repeat
}
```

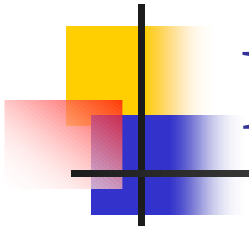
  - The foreach loop takes each element of the array and operates on it.

```
foreach $book_name (@book_list) { print "$book_name\n"; }
```



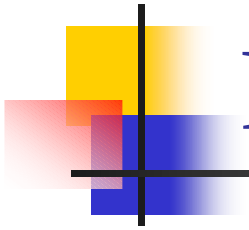
Typical Sequence of Steps

1. User invokes the CGI script
2. CGI script wakes up & Read the user's form input.
3. Do what you want with the data.
4. Write the HTML response to STDOUT.



Receiving User Input from Forms

- The <FORM> tag requires two arguments:
- ACTION
  - the URL representing the script which is to receive the form information
- METHOD
  - either GET or POST
  - represents the way in which the information will get passed to the script



Receiving User Input from Forms

- GET: **<form method="GET" action="…"> </form>**
  - Browser send QUERY_STRING as part of the URL
  - Limitation on the size of the QUERY_STRING
  - Security Problem
- POST: **<form method="POST" action="…"> </form>**
  - Browser send QUERY_STRING in the body of the HTTP request
  - No limitation on the size of the QUERY_STRING
  - Less of Security Problem

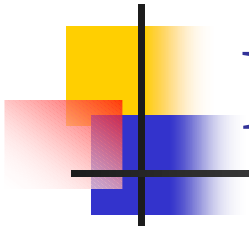
Receiving User Input from Forms

- Using METHOD="GET", we have
  - FORM elements' names are paired with their contents

    <input type="text" size="9" maxlength="9" name="zip"> User inputs 10003, then zip=10003
  - All such name/value pairs are joined together with an ampersand '&'
  - The entire string is URL encoded

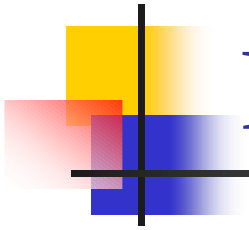
    name=Jane+Doe&address=277+Montrose+Ave&zip=M4G2Z6.

    This string is then passed to QUERY_STRING



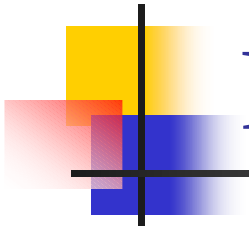
Receiving User Input from Forms

- Using METHOD="POST", we have
  - Basically the same as METHOD="GET"
  - string -> STDIN
  - length of string -> CONTENT_LENGTH



Receiving User Input from Forms

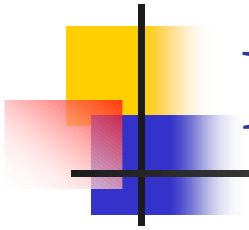
- **Environment Variables:**
  - Text strings (name and value pairs)
  - Can be accessed by other running programs
  - Global, and accessible to all running programs.



Receiving User Input from Forms

- **Environment Variables for** “Get”

  SERVER_SOFTWARE = Apache/1.2b4
  SERVER_NAME = www.scs.leeds.ac.uk
  GATEWAY_INTERFACE = CGI/1.1
  SERVER_PROTOCOL = HTTP/1.1
  SERVER_PORT = 80
  REQUEST_METHOD = GET
  PATH_INFO =
  PATH_TRANSLATED =
  SCRIPT_NAME = /Perl-cgi/environment-example



Receiving User Input from Forms

- **Environment Variables for “Post”**

HTTP_USER_AGENT= Mozilla/1.1N (Macintosh;I;68K)

PATH= /bin:/usr/bin:/usr/ucb:/usr/bsd:/usr/local/bin

SERVER_SOFTWARE = NCSA/1.3

SERVER_NAME = www.homer.doh

SERVER_PORT = 80

GATEWAY_INTERFACE = CGI/1.1

Receiving User Input from Forms

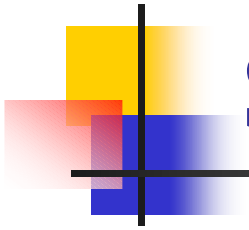
- "Post" Examples

```
#!/usr/local/bin/perl
read(STDIN,$buffer,$ENV{'CONTENT_LENGTH'});
@name_value_pairs=split('&', $buffer');
foreach $name_value_pair (@name_value_pairs) {
    ($name,$value)=split('=',$name_value_pair);
            … …
            %form{$name}=$value;
    }
print "My name is %form{'name'}\n";
print"I live at %form{'address'}\n;
print "My ZIP code is %form{'zip'}\n;
```

Receiving User Input from Forms

- "Get" Examples

```
#!/usr/local/bin/perl
@name_value_pairs=split('&', $ENV{'QUERY_STRING'});
foreach $name_value_pair (@name_value_pairs) {
    ($name,$value)=split('=',$name_value_pair);
    … …
    %form{$name}=$value;
    }
print "My name is %form{'name'}\n";
print"I live at %form{'address'}\n;
print "My ZIP code is %form{'zip'}\n;
```



Sending Info Back to User

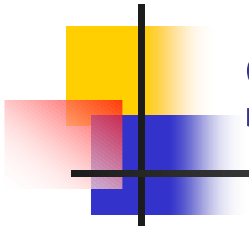
- Print Commands

  - Send information to STDOUT

- Header Information

  - Tells browser what to do with the info

    1. Content-type
    2. Location
    3. Status



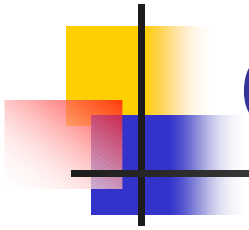
Sending Info Back to User

- Write the line:

  - Content-type: text/html

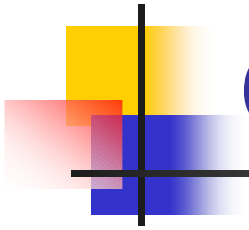
  - plus another blank line, to STDOUT.

- After that, write your HTML response page to STDOUT, and it will be sent to the user when your script is done.



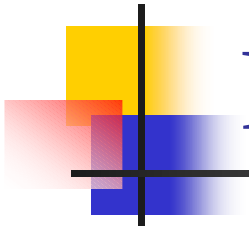
CGI Benefits & Drawbacks

- Benefits
  - Simple
  - Language independence
  - Process isolation
  - Open standard
  - Architecture independence
- Drawbacks
  - Poor efficiency
  - Limited functionality



CGI Security

- A CGI script is a program that anyone in the world can run on your machine.
  - Avoid creating security holes
  - Don't trust the user input
  - Don't put user data in a shell command without verifying the data carefully



References

- Webmaster in a Nutshell
  - O'Reilly & Associates
  - ISBN: 1-56592-325-1

- Official Guide to Programming with CGI.pm
  - Wiley Computer Publishing
  - ISBN: 0-471-24744-8