Internet applications
Lecture 2

Common Gateway Interface

Bartosz Walter

Bartek.Walter@man.poznan.pl

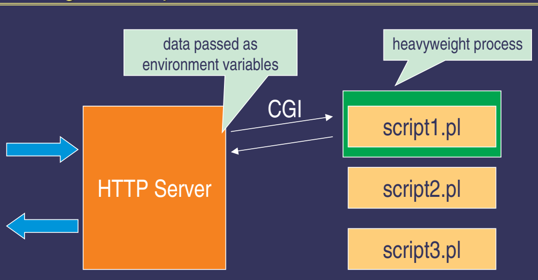
Agenda

- Web server to script communication
- Processing GET request
- Processing POST request
- CGI environment variables
- Configuring Apache HTTP server for CGI
- Security of CGI scripts
- Binary content

CGI in general

- A standard for running external programs on the WWW server side
- The standard includes passing HTTP params, input and output

Running CGI scripts



http://www.server.pl/cgi-bin/script1.cgi

CGI in General

A CGI script

- runs per request as a separate heavyweight process
- communicates to the server through the process environment variables
- the W3C CGI standard defines variables names
- reads the request from a preprocessed variable or stdin
- writes the response to stdout
- scripts DO NOT need to produce headers
- Content-type: application/x-www-form-urlencoded

HTTP POST vs. GET

GET

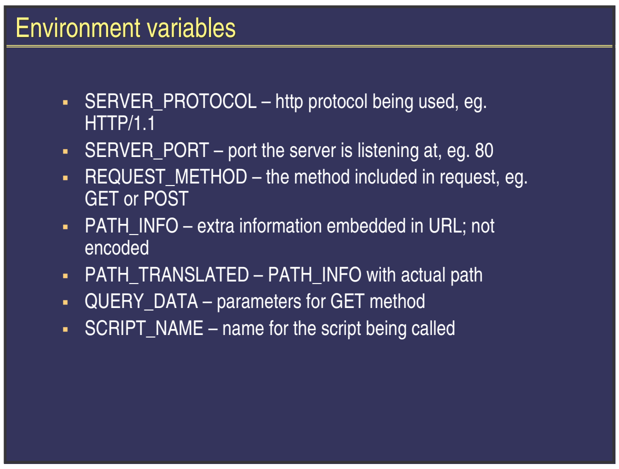
- limited parameters length
- no request body
- parameters encoded in URL only

POST

- parameters encoded in both URL & request body
- body read from stdin
- **content-length** header

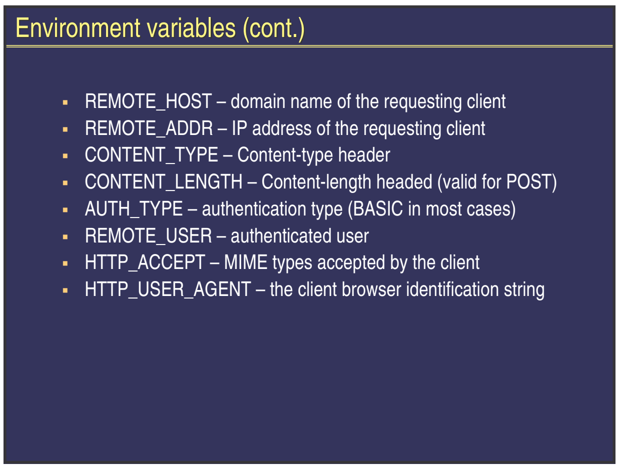
http://www.server.org:8888/students/grades/webapps.html?id=inf47890

http – schema (http, https)
www.server.org - server
8888 – port (80 by default for http, 443 for https)
webapps.html – path info, additional info provided to the servlet
id=inf47890 – query data



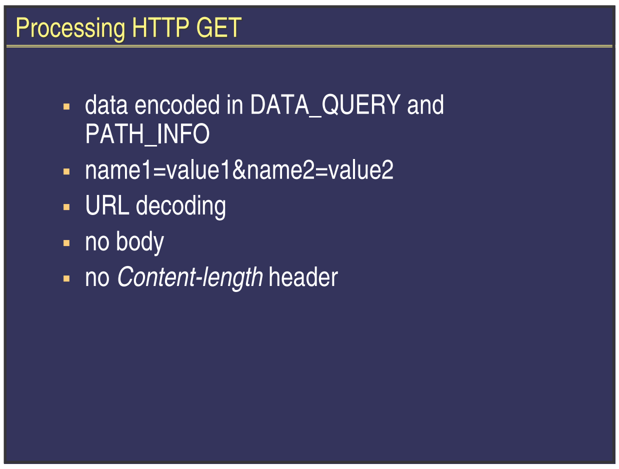
Environment variables

- SERVER_PROTOCOL – http protocol being used, eg. HTTP/1.1
- SERVER_PORT – port the server is listening at, eg. 80
- REQUEST_METHOD – the method included in request, eg. GET or POST
- PATH_INFO – extra information embedded in URL; not encoded
- PATH_TRANSLATED – PATH_INFO with actual path
- QUERY_DATA – parameters for GET method
- SCRIPT_NAME – name for the script being called



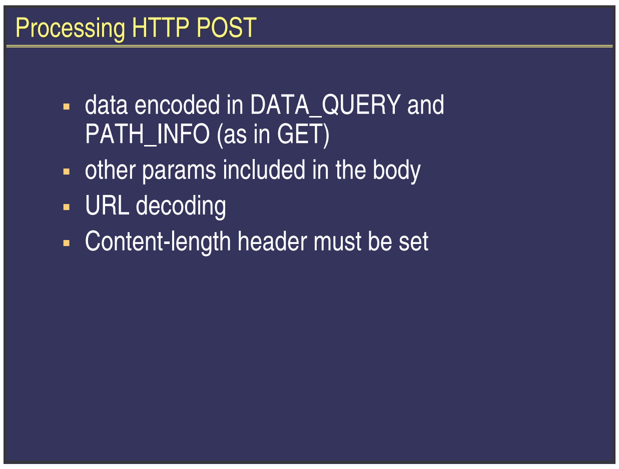
Environment variables (cont.)

- REMOTE_HOST – domain name of the requesting client
- REMOTE_ADDR – IP address of the requesting client
- CONTENT_TYPE – Content-type header
- CONTENT_LENGTH – Content-length headed (valid for POST)
- AUTH_TYPE – authentication type (BASIC in most cases)
- REMOTE_USER – authenticated user
- HTTP_ACCEPT – MIME types accepted by the client
- HTTP_USER_AGENT – the client browser identification string



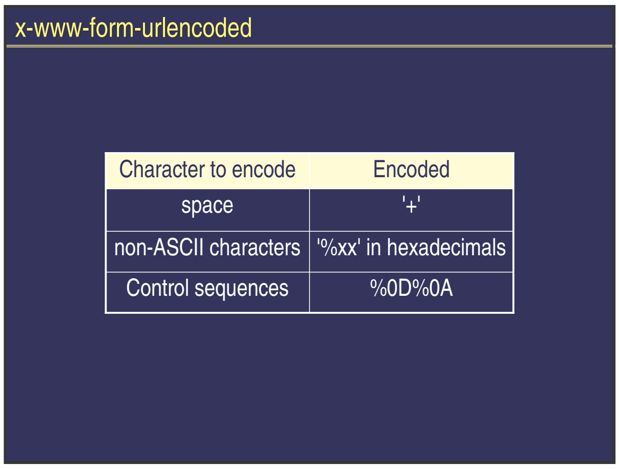
Processing HTTP GET

- data encoded in DATA_QUERY and PATH_INFO
- name1=value1&name2=value2
- URL decoding
- no body
- no *Content-length* header



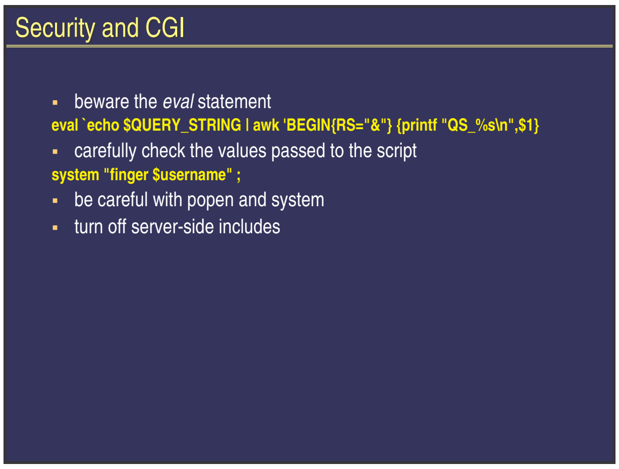
Processing HTTP POST

- data encoded in DATA_QUERY and PATH_INFO (as in GET)
- other params included in the body
- URL decoding
- Content-length header must be set



x-www-form-urlencoded

Character to encode	Encoded
space	'+'
non-ASCII characters	'%xx' in hexadecimals
Control sequences	%0D%0A



Security and CGI

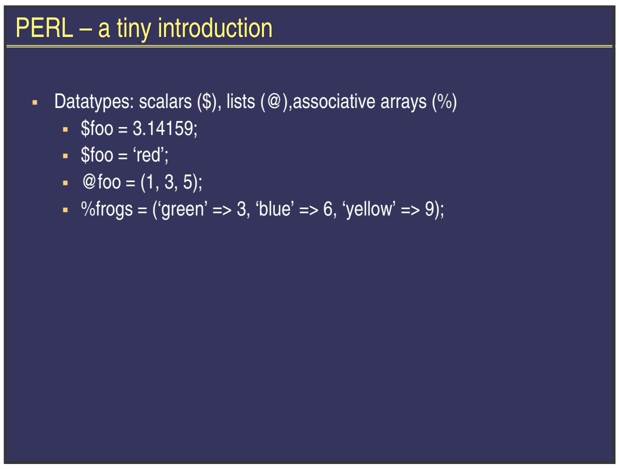
- beware the *eval* statement

```
eval `echo $QUERY_STRING | awk 'BEGIN{RS="&"} {printf "QS_%s\n",$1}
```

- carefully check the values passed to the script

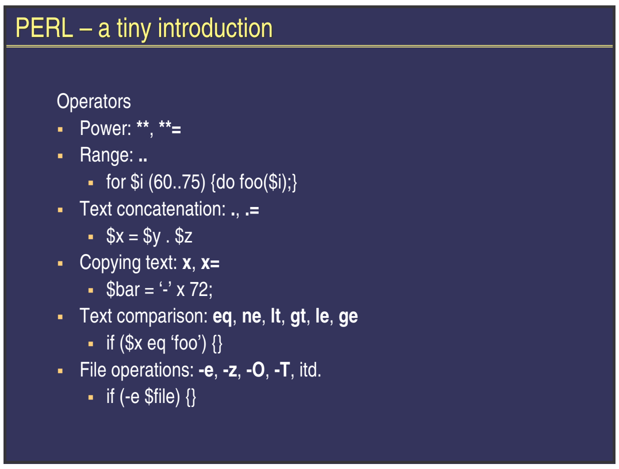
```
system "finger $username" ;
```

- be careful with popen and system
- turn off server-side includes



PERL – a tiny introduction

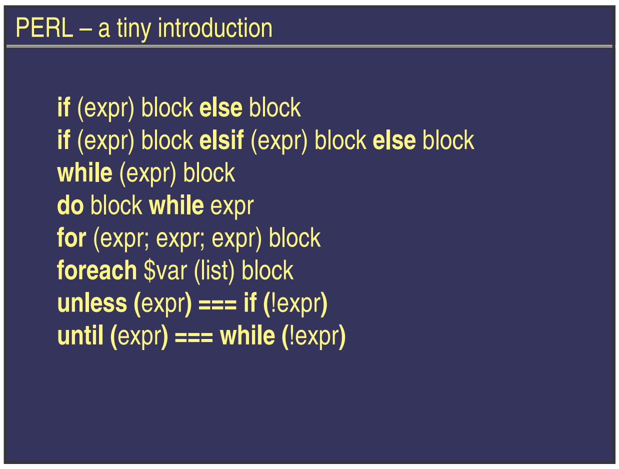
- Datatypes: scalars ($), lists (@),associative arrays (%)
  - $foo = 3.14159;
  - $foo = ‘red’;
  - @foo = (1, 3, 5);
  - %frogs = (‘green’ => 3, ‘blue’ => 6, ‘yellow’ => 9);



PERL – a tiny introduction

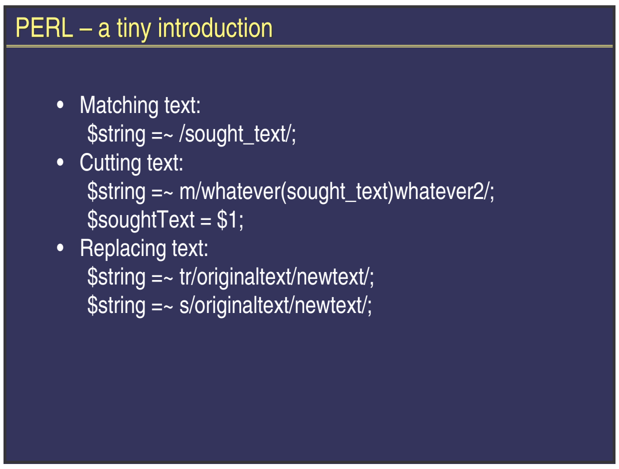
Operators

- Power: ******, ****=**
- Range: **..**
  - for $i (60..75) {do foo($i);}
- Text concatenation: **.**, **.=**
  - $x = $y . $z
- Copying text: **x**, **x=**
  - $bar = ‘-’ x 72;
- Text comparison: **eq**, **ne**, **lt**, **gt**, **le**, **ge**
  - if ($x eq ‘foo’) {}
- File operations: **-e**, **-z**, **-O**, **-T**, itd.
  - if (-e $file) {}



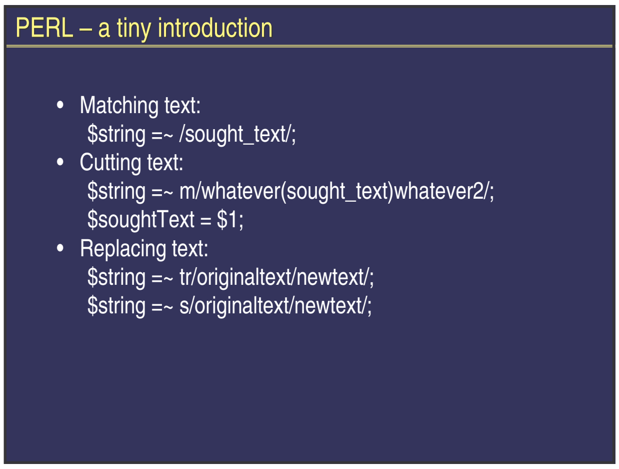
PERL – a tiny introduction

if (expr) block **else** block
if (expr) block **elsif** (expr) block **else** block
while (expr) block
do block **while** expr
for (expr; expr; expr) block
foreach $var (list) block
unless (expr**) === if (**!expr**)**
until (expr**) === while (**!expr**)**



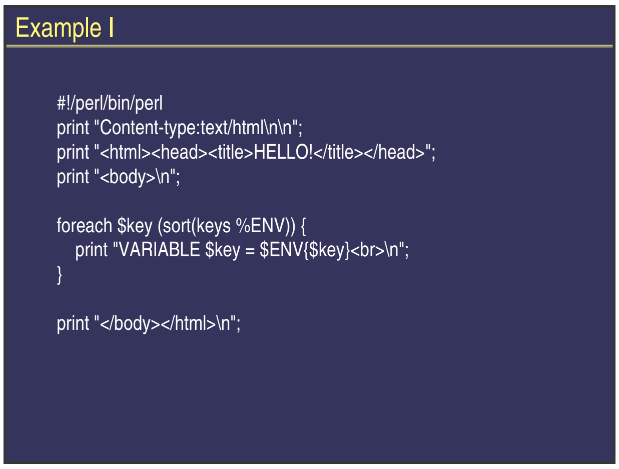
PERL – a tiny introduction

- Matching text:
  $string =~ /sought_text/;
- Cutting text:
  $string =~ m/whatever(sought_text)whatever2/;
  $soughtText = $1;
- Replacing text:
  $string =~ tr/originaltext/newtext/;
  $string =~ s/originaltext/newtext/;



PERL – a tiny introduction

- Matching text:
  $string =~ /sought_text/;
- Cutting text:
  $string =~ m/whatever(sought_text)whatever2/;
  $soughtText = $1;
- Replacing text:
  $string =~ tr/originaltext/newtext/;
  $string =~ s/originaltext/newtext/;

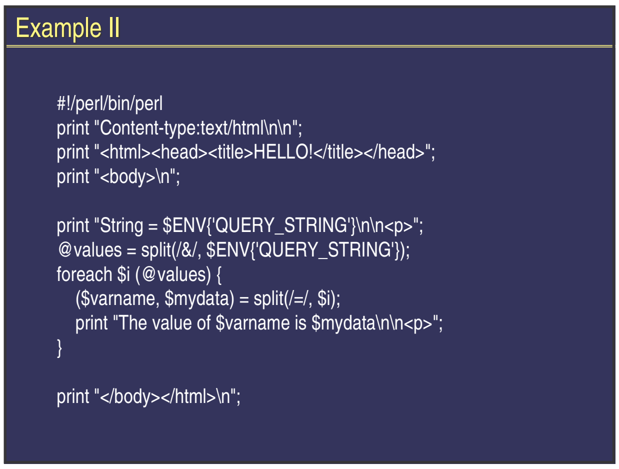


Example I

```
#!/perl/bin/perl
print "Content-type:text/html\n\n";
print "<html><head><title>HELLO!</title></head>";
print "<body>\n";

foreach $key (sort(keys %ENV)) {
    print "VARIABLE $key = $ENV{$key}<br>\n";
}

print "</body></html>\n";
```

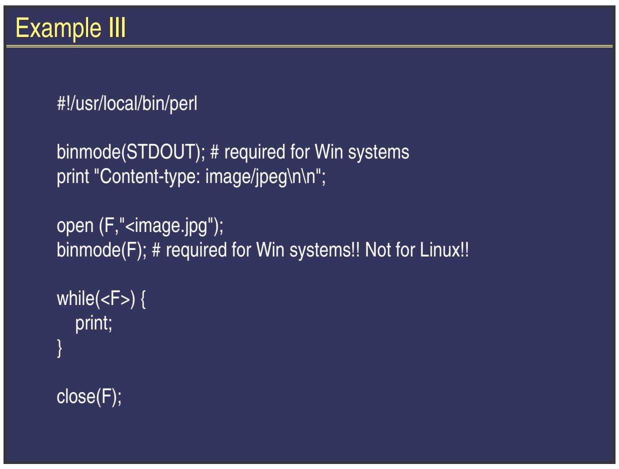


Example II

```
#!/perl/bin/perl
print "Content-type:text/html\n\n";
print "<html><head><title>HELLO!</title></head>";
print "<body>\n";

print "String = $ENV{'QUERY_STRING'}\n\n<p>";
@values = split(/&/, $ENV{'QUERY_STRING'});
foreach $i (@values) {
   ($varname, $mydata) = split(/=/, $i);
   print "The value of $varname is $mydata\n\n<p>";
}

print "</body></html>\n";
```



Example III

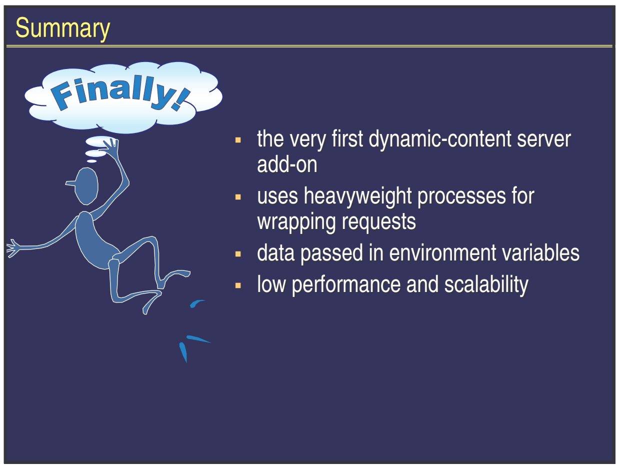
```
#!/usr/local/bin/perl

binmode(STDOUT); # required for Win systems
print "Content-type: image/jpeg\n\n";

open (F,"<image.jpg");
binmode(F); # required for Win systems!! Not for Linux!!

while(<F>) {
   print;
}

close(F);
```



Summary

- the very first dynamic-content server add-on
- uses heavyweight processes for wrapping requests
- data passed in environment variables
- low performance and scalability

Questions?