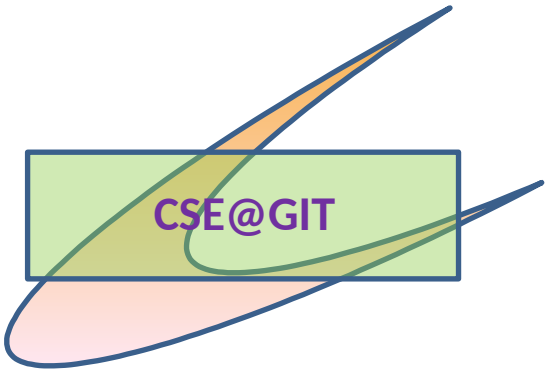
# Experiment No. 8

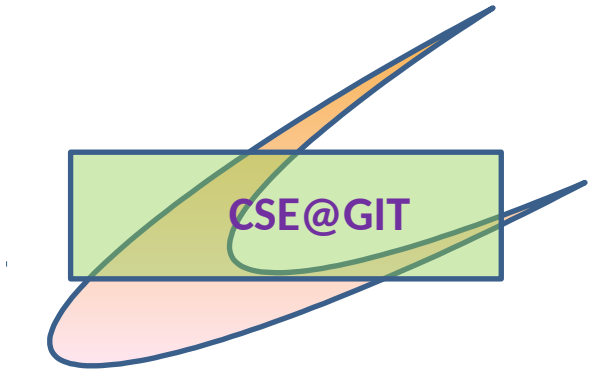
**Problem Definition:** 8. Write a Perl program to insert name and age information entered by the user into a table created using MySQL and to display the current contents of this table.



# Objectives of the Experiment:

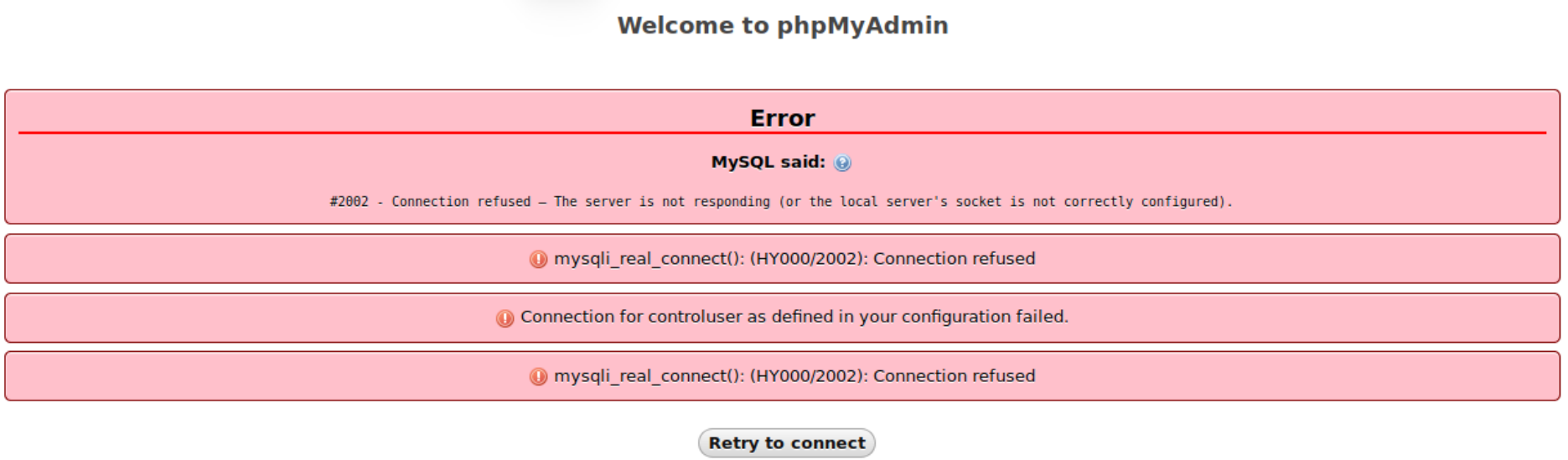
- To demonstrate the use MySQL
- Use PERL to access tables in MySQL

- How To Guide in http://localhost
- Compile and run PHP code



# XAMPP and MySQL

- http://localhost
- http://localhost/phpmyadmin/




- #2002 - The server is not responding (or the local MySQL server's socket is not correctly configured)

# XAMPP and MySQL

- Check on terminal / command prompt
- mysql -u root -p

- Configuration files - **config.inc.php**
- Same setting should not be used for setting up server
- Conditions are relaxed for development / testing environment

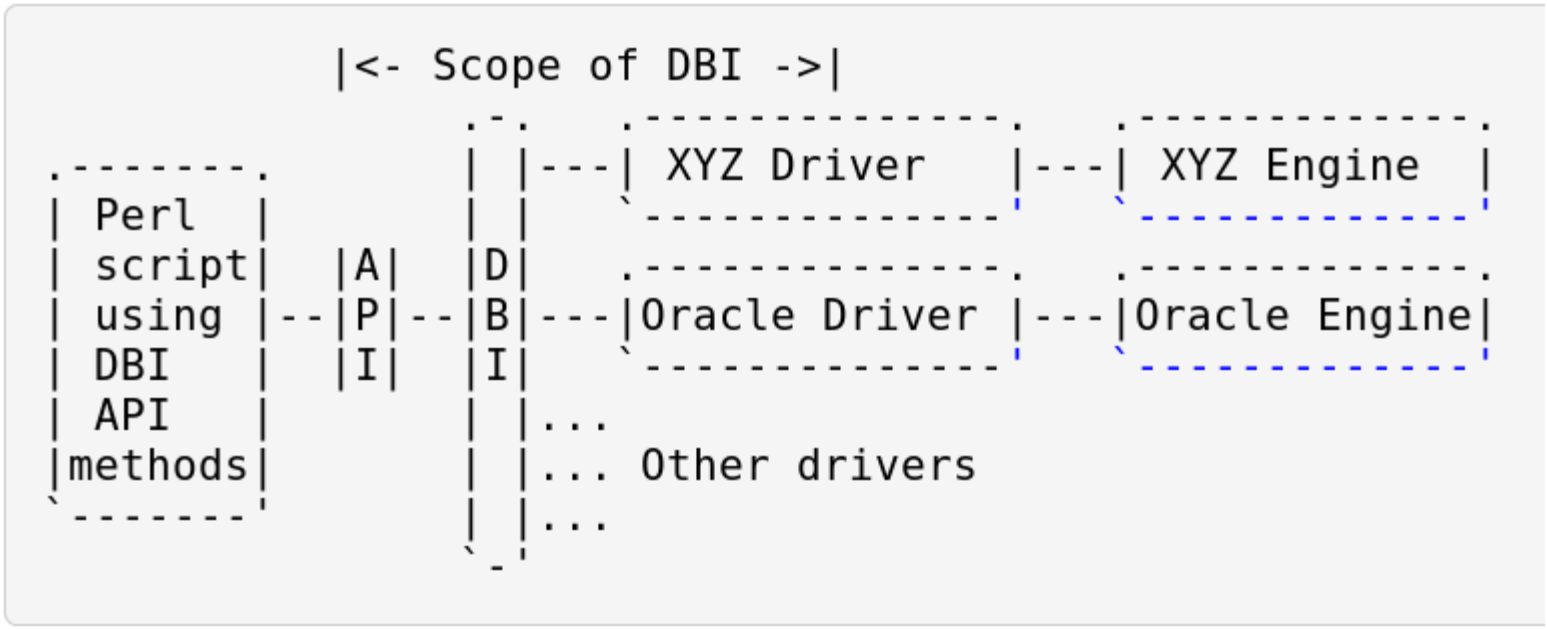
# XAMPP and MySQL

- Since XAMPP 5.5.30 and 5.6.14, XAMPP ships MariaDB instead of **MySQL**
- But commands and tools **are the same for both**
- MariaDB - created by the original developers of MySQL

[ https://dbi.perl.org/ ]

# Architecture of a DBI Application

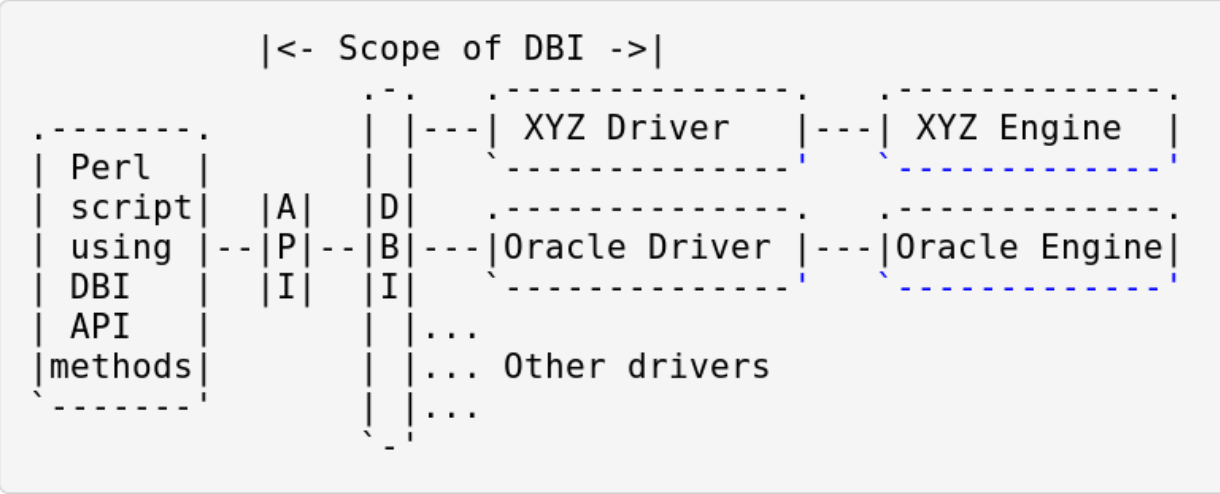
- DBI - Database independent interface
- Defines a set of methods, variables, and conventions
- Provide a consistent database interface, independent of the actual database being used




[ Tim Bunce, https://metacpan.org/pod/DBI ]

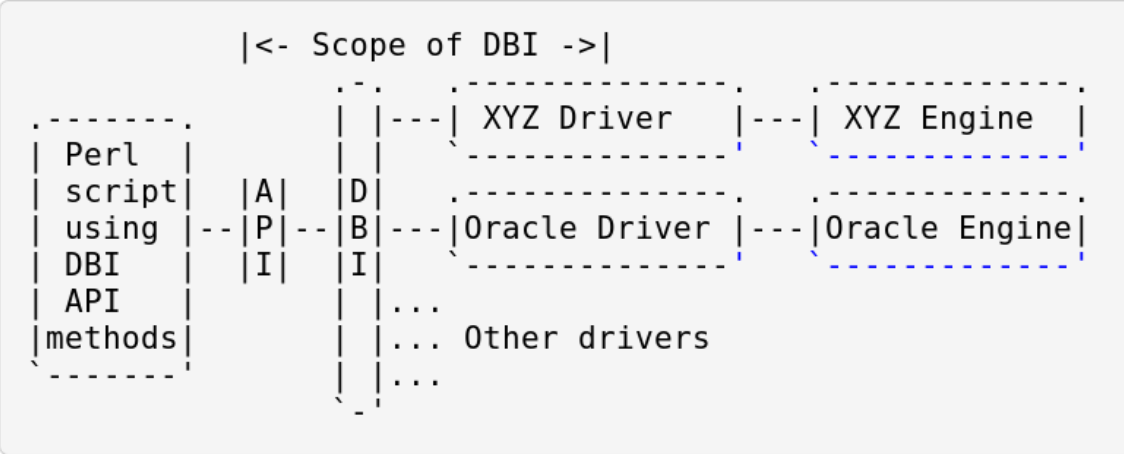
# Architecture of a DBI Application

- DBI - Database independent interface
- Defines a set of methods, variables, and conventions
- Provide a consistent database interface, independent of the actual database being used




- DBI is interface, a layer of "glue" between an application and one or more database driver modules
- Driver modules which do most of the real work

# Architecture of a DBI Application




- **API**, or Application Programming Interface, defines call interface and variables for Perl scripts to use
- API is implemented by the Perl **DBI** extension
- DBI "dispatches" the method calls to the appropriate driver for actual execution
- Driver contains implementations of the DBI methods using the private interface functions of the corresponding database engine

# Notation and Conventions

- 

```
$dbh     Database handle object
$sth     Statement handle object
$drh     Driver handle object (rarely seen or used in applications)
$h       Any of the handle types above ($dbh, $sth, or $drh)
$rc      General Return Code  (boolean: true=ok, false=error)
$rv      General Return Value (typically an integer)
@ary     List of values returned from the database, typically a row
           of data
$rows    Number of rows processed (if available, else -1)
$fh      A filehandle
undef    NULL values are represented by undefined values in Perl
\%attr   Reference to a hash of attribute values passed to methods
```

# Usage

- To use DBI, first you need to load the DBI module:

**use DBI;**

```
#!"C:\xampp\perl\bin\perl.exe"
use DBI;
```

# Usage

- To use DBI, first you need to load the DBI module:

**use DBI;**

```
#!"C:\xampp\perl\bin\perl.exe"
use DBI;
```

- To "**connect**" to your data source and get a handle for that connection:

```
$dbh = DBI->connect($dsn, $user, $password);
```

# Usage

- To use DBI, first you need to load the DBI module:

**use DBI;**

```
#!"C:\xampp\perl\bin\perl.exe"
use DBI;
```

- To "**connect**" to your data source and get a handle for that connection:

```
$dbh = DBI->connect($dsn, $user, $password);
```

| | |
|---|---|
| **$dsn** | Data Source Name (DSN) |
| **$user** | Username, owner of database |
| **$password** | Authentication, password |

# Usage

- To use DBI, first you need to load the DBI module:

**use DBI;**

```
#!"C:\xampp\perl\bin\perl.exe"
use DBI;
```

- To "**connect**" to your data source and get a handle for that connection:

```
$dbh = DBI->connect($dsn, $user, $password);
```

| | |
|---|---|
| `$dsn` | Data Source Name (DSN) |
| `$user` | Username, owner of database |
| `$password` | Authentication, password |

# Usage

- To get to know dsn, user and password : http://127.0.0.1/phpmyadmin/ or http://localhost/phpmyadmin/

  or
- config file

  or
- On terminal :

# Usage

- To get to know dsn, user and password :

```
$dsn="DBI:mysql:git";
$user="root";
$password="root";
$dbh = DBI->connect($dsn, $user, $password);
```

# Connect, query using PERL

- Depending on operating system : Driver name case sensitive
- Works fine on Windows but not on UNIX like operating system

```
$con=DBI->connect("DBI:Mysql:database=git","root","root");
```

# Connect, query using PERL

- Depending on operating system : Driver name case sensitive
- Works fine on Windows but not on UNIX like operating system

```
$con=DBI->connect("DBI:Mysql:database=git","root","root");
```

- Gives error :

```
install_driver(Mysql) failed: Can't locate DBD/Mysql.pm in @INC (you may need to
 install the DBD::Mysql module) (@INC contains: /etc/perl /usr/local/lib/x86_64-
linux-gnu/perl/5.24.1 /usr/local/share/perl/5.24.1 /usr/lib/x86_64-linux-gnu/per
l5/5.24 /usr/share/perl5 /usr/lib/x86_64-linux-gnu/perl/5.24 /usr/share/perl/5.2
4 /usr/local/lib/site_perl /usr/lib/x86_64-linux-gnu/perl-base) at (eval 6) line
 3.
Perhaps the DBD::Mysql perl module hasn't been fully installed,
or perhaps the capitalisation of 'Mysql' isn't right.
Available drivers: DBM, ExampleP, File, Gofer, Proxy, Sponge, mysql.
```

# Connect, query using PERL

- Depending on operating system : Driver name case sensitive
- Works fine on Windows but not on UNIX like operating system

```
$con=DBI->connect("DBI:Mysql:database=git","root","root");
```

- Instead use :

```
$con=DBI->connect("DBI:mysql:database=git","root","root");
```

# Connect, query using PERL

- D

  [ https://www.perl.com/pub/1999/10/DBI.html ]

[ Author : Randal Schwartz from Portland, OR, USA ]

# Perl

- Perl : Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development
- Larry Wall, major : chemistry
- Perl 5
- The Swiss Army chainsaw of scripting languages
- Official Perl documentation states that :

  1. Larry is always by definition right about how Perl should behave. This means he has final veto power on the core functionality.

  2. Larry is allowed to change his mind about any matter at a later date, regardless of whether he previously invoked Rule 1. Got that? Larry is always right, even when he was wrong.

[ https://www.perl.org/ ]

# Perl

- Supports both procedural and object-oriented (OO) programming
- Perl documentation : **perdoc**
- To solve a problem : There's More Than One Way To Do It
- Perl program genrally saved with extension **.pl**
- **hello.pl**

```
print " Hello World \n "
```

- To run a Perl program
- XAMP installation , Windows : Perl available in C:\xampp\perl\bin**perl.exe**
  ( Or in UNIX like systems , if Perl is installed , directly : )
- **perl hello.pl**

```
perl hello.pl
```

```
Hello World
```

# Perl

- To run directly ( like a Shell Script : )
- As **first** line in program : **#!Path**Of**perl.exe**

```
#!C:\xampp\perl\bin\perl
```

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

- Then to run : **./hello.pl**

- Safety net :

```
#!/usr/bin/perl
use strict;
use warnings;

 print " Hello World \n "
```

- **use strict;** will cause code to stop immediately when problem is encountered
- **use warnings;** will merely give a warning and let your code run

[ http://perldoc.perl.org/perlintro.html ]

# Perl Script / Program

- No need to have a main() function
- Perl statements **end** in a semi-colon **;**
- Comments start with a hash symbol and run to the end of the line

```
# This is a comment
```

- Whitespace is irrelevant , except inside quoted strings :

```
print " Hello World \n "
                        ;
print " Hello
        World \n ";
```

```
Hello World
 Hello
       World
```

- Double quotes or single quotes may be used around literal strings

```
print " Hello World \n "
                        ;
print ' Hello
        World \n ';
```

```
Hello World
 Hello
       World \n
```

# Perl Script / Program

- Only double quotes "interpolate" variables and special characters such as newlines \n
- Single quotes treats as string

```
my $name="What's in a name";

print " Hello $name \n "
                        ;
print ' Hello
        $name \n ';
```

```
Hello What's in a name
 Hello
       $name \n
```

- Parentheses can be used for function's arguments or omited
- Required to clarify issues of precedence

```
print("Hello, world\n");
```

# Perl variable types

- Scalars, Arrays and Hashes
- **Scalar** represents a single value

```
my $animal = "camel";
my $answer = 42;
```

- Scalar values can be strings, integers or floating point numbers, and Perl will automatically convert between them as required
- There is no need to pre-declare your variable types,
- But you have to declare them using the **my** keyword the first time you use them (One of the requirements of **use strict;** )

```
print $animal;
print "The animal is $animal\n";
print "The square of $answer is ", $answer * $answer, "\n";
```

```
camelThe animal is camel
The square of 42 is 1764
```

# Perl variable types

- Scalars, Arrays and Hashes

- **Array** represents a list of values

```
my @animals = ("camel", "llama", "owl");
my @numbers = (23, 42, 69);
my @mixed   = ("camel", 42, 1.23);
```

- Arrays are zero-indexed

```
print $animals[0];
print $animals[1];
```

- Variable **$#array** tells you the index of the last element of an array

```
print $mixed[$#mixed];
```

- Array slice : get multiple values

```
@animals[0,1]
@animals[0..2]
@animals[1..$#animals]
```

# Perl variable types

- Scalars, Arrays and Hashes

- **Hashes** : represent set of key/value pairs
- Use whitespace and the **=>** operator to lay them out

```
my %fruit_color = ("apple", "red", "banana", "yellow");

my %fruit_color = (
    apple  => "red",
    banana => "yellow",
);
```

- To get at hash elements : `$fruit_color{"apple"}`

# Conditional and looping constructs

- if
- unless
- while
- until
- for
- foreach

# Conditional and looping constructs

- if

```
if ( condition ) {
    ...
} elsif ( other condition ) {
    ...
} else {
    ...
}
```

- unless
  Negated version of if

```
unless ( condition ) {
    ...
}
```

# Conditional and looping constructs

- if and unless

```
my $zippy="Two and a half";
my $bananas="";
# the traditional way
if ($zippy) {
    print "Yow!";
}
# the Perlish post-condition way
print "Yow!" if $zippy;
print "We have no bananas" unless $bananas;
```

# Conditional and looping constructs

- while

```
while ( condition ) {
    ...
}
```

- until
  Negated version of while

```
until ( condition ) {
    ...
}
```

```
print "LA LA LA\n" while 1;
```

# Conditional and looping constructs

- for

```
for ($i = 0; $i <= $max; $i++) {
    ...
}
```

- C style for loop
- Perl provides the more friendly list scanning **foreach** loop

- Can we expect this soon ?

```
for (₹i = 0; ₹i <= ₹max; ₹i++) {
    ...
}
```

# Conditional and looping constructs

- foreach

```
my @animals = ("camel", "llama", "owl");
my @numbers = (23, 42, 69);
my %fruit_color = (
    apple  => "red",
    banana => "yellow",
);
```

```
foreach (@animals) {
    print "This element is $_\n";
}

print $numbers[$_] foreach 0 .. $#numbers;

# you don't have to use the default $_ either...
foreach my $key (keys %fruit_color) {
    print "The \$key is $key \n";
    print "The value of $key is $fruit_color{$key}\n";
}
```

# Builtin operators and functions

- Arithmetic
- Numeric comparison
- String comparison
- Boolean logic
- Miscellaneous

# Builtin operators and functions

- Arithmetic

```
+   addition
-   subtraction
*   multiplication
/   division
```

- Numeric comparison

```
==  equality
!=  inequality
<   less than
>   greater than
<=  less than OR equal
>=  greater than OR equal
```

# Builtin operators and functions

- String comparison

```
eq  equality
ne  inequality
lt  less than
gt  greater than
le  less than OR equal
ge  greater than OR equal
```

- Boolean logic

```
&&  AND
||  OR
!   NOT
```

- Miscellaneous

# Builtin operators and functions

- Miscellaneous

```
=    assignment
.    string concatenation
x    string multiplication
..   range operator (creates a list of numbers OR strings)
```

```
my $a=1;

$a += 1;          # same as $a = $a + 1
print " a = $a";

$a -= 1;          # same as $a = $a - 1
print " a = $a";

$a .= "\n";       # same as $a = $a . "\n";
print " a = $a";
```

# Files and I/O

- **open()** - open a file for input or output

```
open(my $in,  "<",  "input.txt")  or die "Can't open input.txt: $!";
open(my $out, ">",  "output.txt") or die "Can't open output.txt: $!";
open(my $log, ">>", "my.log")     or die "Can't open my.log: $!";
```

- Read from an open filehandle using the **<>** operator
- In scalar context it reads a single line from the filehandle

```
my $line  = <$in>;
my @lines = <$in>;
```

- In list context it reads the whole file in, assigning each line to an element of the list

[ Author : Kirrily "Skud" Robert <skud@cpan.org> ]

# Files and I/O

- **print()** can also take an optional first argument specifying which filehandle to print to :

```
my $message="Remember, Hope is a good thing, \n";
my $logmessage="maybe the best of things,
    and no good thing ever dies - Stephen King\n";

print STDERR "Program testing can be used to show
              the presence of bugs, but never to
              show their absence!. - Dijkstra\n";
print $out $message;
print $log $logmessage;
```

- When completed with read / write operation on files : **close()**

```
close $in  or die "$in: $!";
close $out or die "$out: $!";
close $log or die "$log: $!";
```

# Programming Style

- Object uriented or Function oriented
- use

```
#!C:\xampp\perl\bin\perl
use CGI;  # load CGI routines
```

- CGI has routines to :
  - Retrieve CGI parameters
  - Create HTML tags
  - Manage cookie

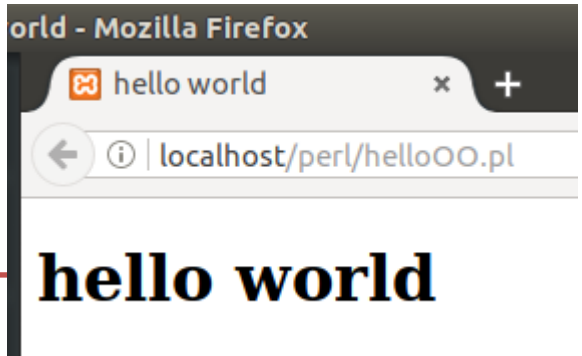
```
#!C:\xampp\perl\bin\perl
use CGI qw/:standard/;   # load standard CGI routines
```

[ http://perldoc.perl.org/CGI.html ]

# Programming Style

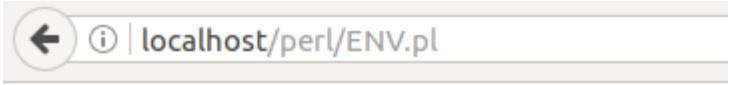
```
#!C:\xampp\perl\bin\perl
use CGI qw/:standard/;   # load standard CGI routines

print ( header( ) );                # create the HTTP header
print start_html('hello world'); # start the HTML
print h1('hello world');            # level 1 header
print end_html;                     # end the HTML
```

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head>
<title>hello world</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<h1>hello world</h1>
</body>
</html>
```



# Environment Variables

```
foreach my $key ( keys %ENV )
 {
   print " $key <br/>"
 }
```




## ENV Variables

ENV Variable Names =
SCRIPT_NAME
REQUEST_METHOD
HTTP_ACCEPT
SCRIPT_FILENAME
REQUEST_SCHEME
SERVER_SOFTWARE
QUERY_STRING
REMOTE_PORT
HTTP_USER_AGENT
SERVER_SIGNATURE
HTTP_ACCEPT_LANGUAGE
HTTP_UPGRADE_INSECURE_REQUESTS
MOD_PERL_API_VERSION
PATH
GATEWAY_INTERFACE
DOCUMENT_ROOT
UNIQUE_ID
SERVER_NAME
HTTP_REFERER
HTTP_ACCEPT_ENCODING
LD_LIBRARY_PATH
SERVER_ADMIN
HTTP_CONNECTION
CONTEXT_PREFIX
SERVER_PORT
REMOTE_ADDR
CONTEXT_DOCUMENT_ROOT
SERVER_PROTOCOL
REQUEST_URI
SERVER_ADDR
HTTP_HOST
MOD_PERL

# Environment Variables and Values

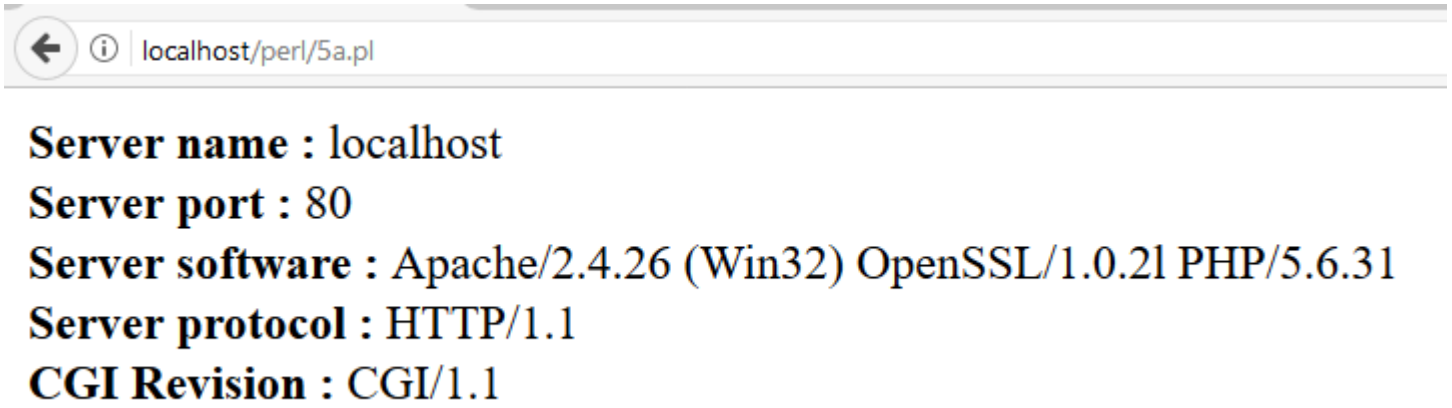
```
print " ENV Variable <strong> Name = Value </strong> <br/>";
foreach my $key ( keys %ENV )
 {
   print " $key = $ENV{$key} <br/>";
 }
```

ENV Variable **Name = Value**
SCRIPT_NAME = /perl/ENV.pl
REQUEST_METHOD = GET
HTTP_ACCEPT = text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
SCRIPT_FILENAME = /opt/lampp/htdocs/perl/ENV.pl
REQUEST_SCHEME = http
SERVER_SOFTWARE = Apache/2.4.25 (Unix) OpenSSL/1.0.2j PHP/7.1.1 mod_perl/2.0.8-dev Perl/v5.16.3
QUERY_STRING =
REMOTE_PORT = 46258
HTTP_USER_AGENT = Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:55.0) Gecko/20100101 Firefox/55.0
SERVER_SIGNATURE =
HTTP_CACHE_CONTROL = max-age=0
HTTP_ACCEPT_LANGUAGE = en-US,en;q=0.5
HTTP_UPGRADE_INSECURE_REQUESTS = 1
MOD_PERL_API_VERSION = 2
PATH = /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin

# 5a. Pseudo Code / Outline of the Algorithm

```
#!C:\xampp\perl\bin\perl
use CGI qw(:standard);
print header();
print start_html();
print "<b>Server name :</b> $ENV{'SERVER_NAME'}<br/>";
print "<b>Server port :</b> $ENV{'SERVER_PORT'}<br/>";
print "<b>Server software :</b> $ENV{'SERVER_SOFTWARE'}<br/>";
print "<b>Server protocol :</b> $ENV{'SERVER_PROTOCOL'}<br/>";
print "<b>CGI Revision :</b> $ENV{'GATEWAY_INTERFACE'}<br/>";
print end_html();
```

# Sample Run

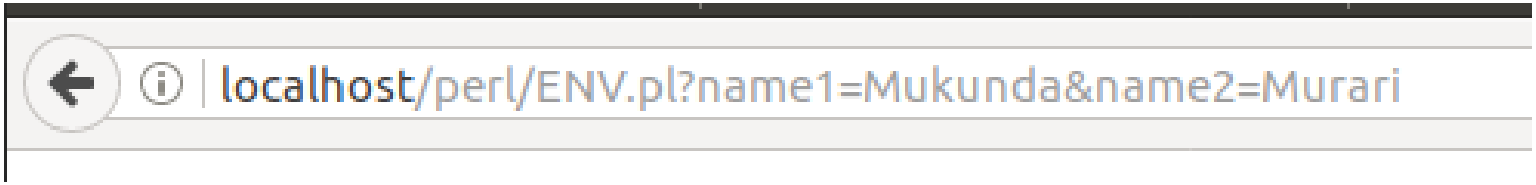

localhost/perl/5a.pl
Server name : localhost
Server port : 80
Server software : Apache/2.4.26 (Win32) OpenSSL/1.0.2l PHP/5.6.31
Server protocol : HTTP/1.1
CGI Revision : CGI/1.1

# 5a. Pseudo Code / Outline of the Algorithm

```
print "<br/> <b> Server name :</b> ", server_name() ,
      "<br/> <b> Server port :</b> ", server_port(),
      "<br/> <b> Server software :</b> ", server_software(),
      "<br/> <b> Server protocol :</b> ", server_protocol();
```

# FETCHING THE NAMES OF ALL THE PARAMETERS PASSED TO YOUR SCRIPT

- If the script was invoked with a parameter list
- http://localhost/perl/script.pl?name1=value1&name2=value2
- **param()** method will return the parameter names as a list




```
$value1 = param("name1");

$value2 = param("name2");

print " name1 = $value1 <br/> name2 = $value2 <br/>"
```

name1 = Mukunda
name2 = Murari

# Invoke UNIX commands in Perl Script

- **system()** call , back ticks , quote execute
- system(command) , `command` , qx/command/
- Differences is in the returning value
- system call returns the **return value** of that command execution
- `` ` `` and qx return command execution's output

```
$cmd = param("cmd");

print "<h1>The output of $cmd is:</h1>";

print system($cmd) , "<br/>";

print `$cmd` , "<br/>";

print qx/$cmd/ , "<br/>";

print qx{$cmd} , "<br/>";
```

# Learning Outcomes of the Experiment

At the end of the session, students should be able to :

**1) Experiment with the database connections, query using Perl [L3]**

Acknowledgement : Thank to Sagar for Laptop to test XAMPP installation and working of programs on Windows 10