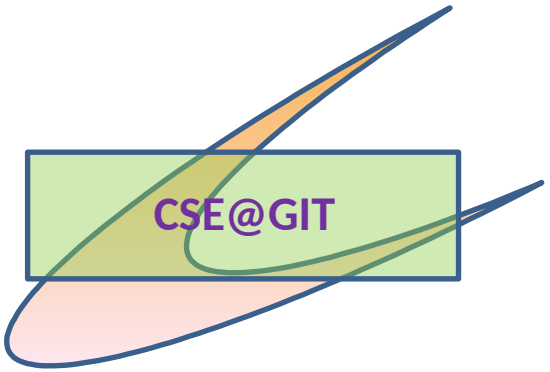
Experiment No. 5

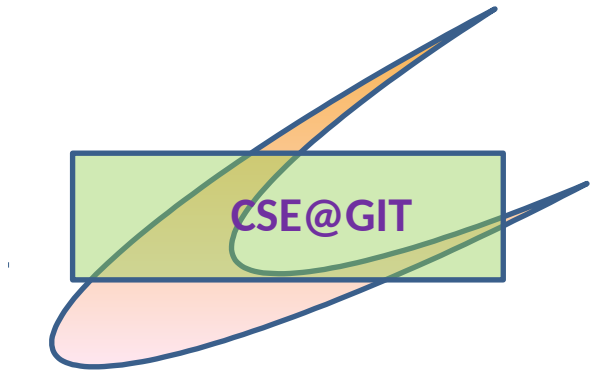
Problem Definition: 5. a) Write a Perl program to display various Server Information like Server, Name, Server Software, Server protocol, CGI Revision etc.

b) Write a Perl program to accept UNIX command from a HTML form and to display the output of the command executed.



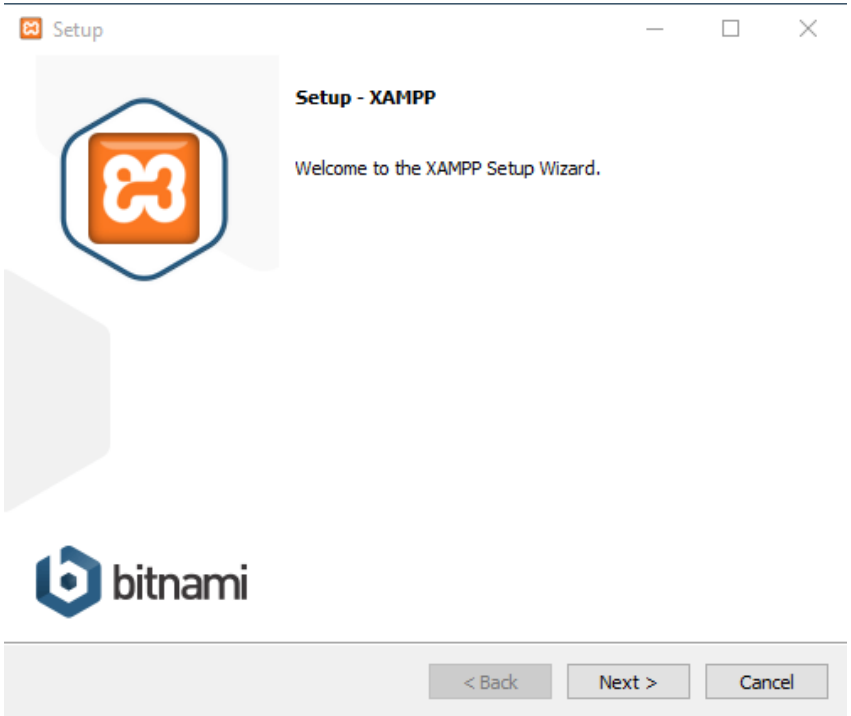
Objectives of the Experiment:

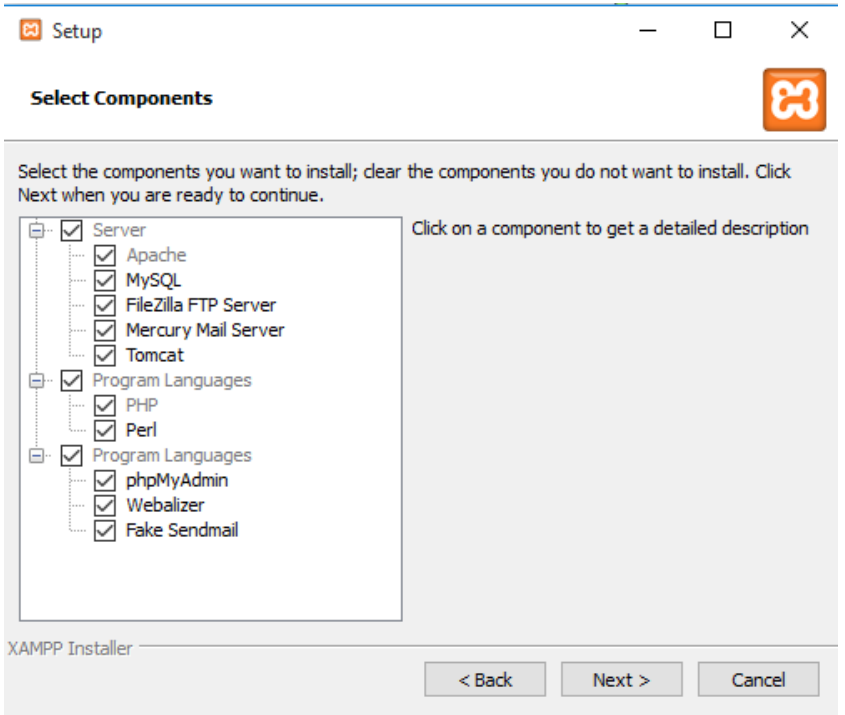
- To demonstrate the use of Common Gateway Interface(CGI)
- Client , server interaction
- To develop an understanding of Perl script



XAMPP for Windows

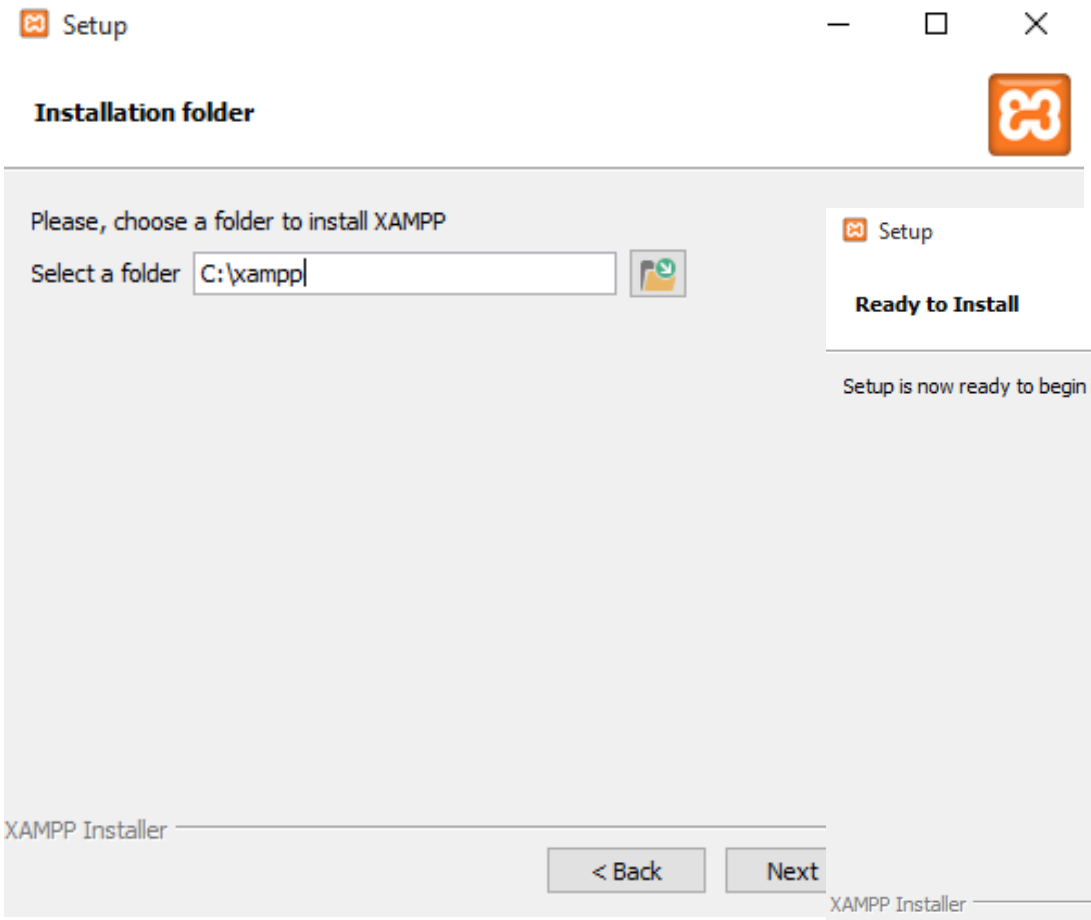
- **XAMPP** is Apache distribution containing MySQL, PHP, and Perl
- To install XAMPP : Download installer from https://www.apachefriends.org/download.html

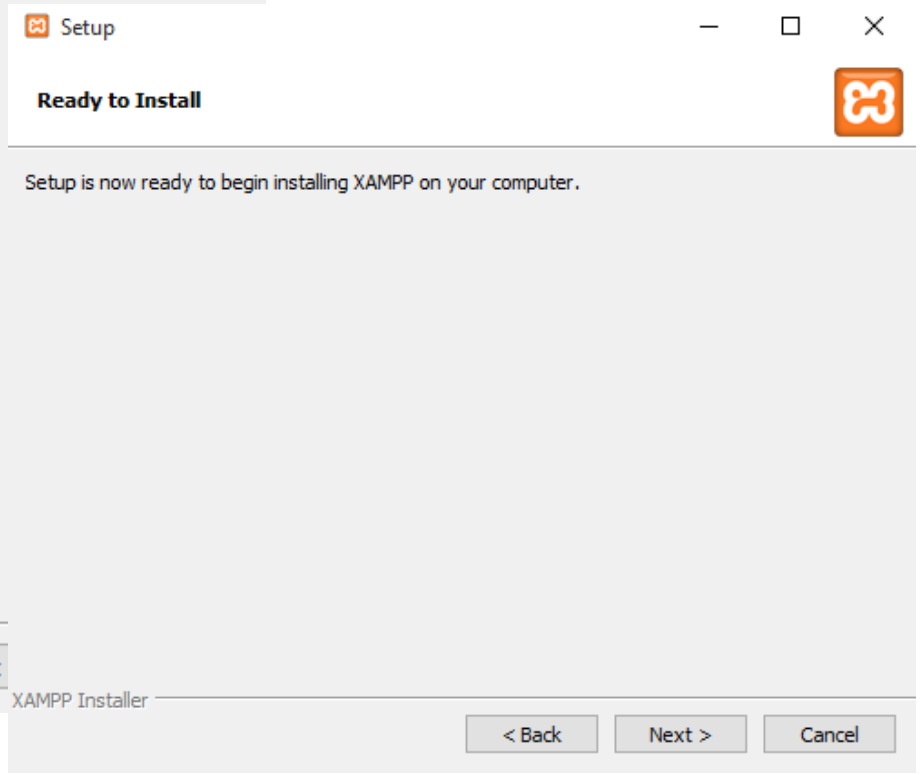




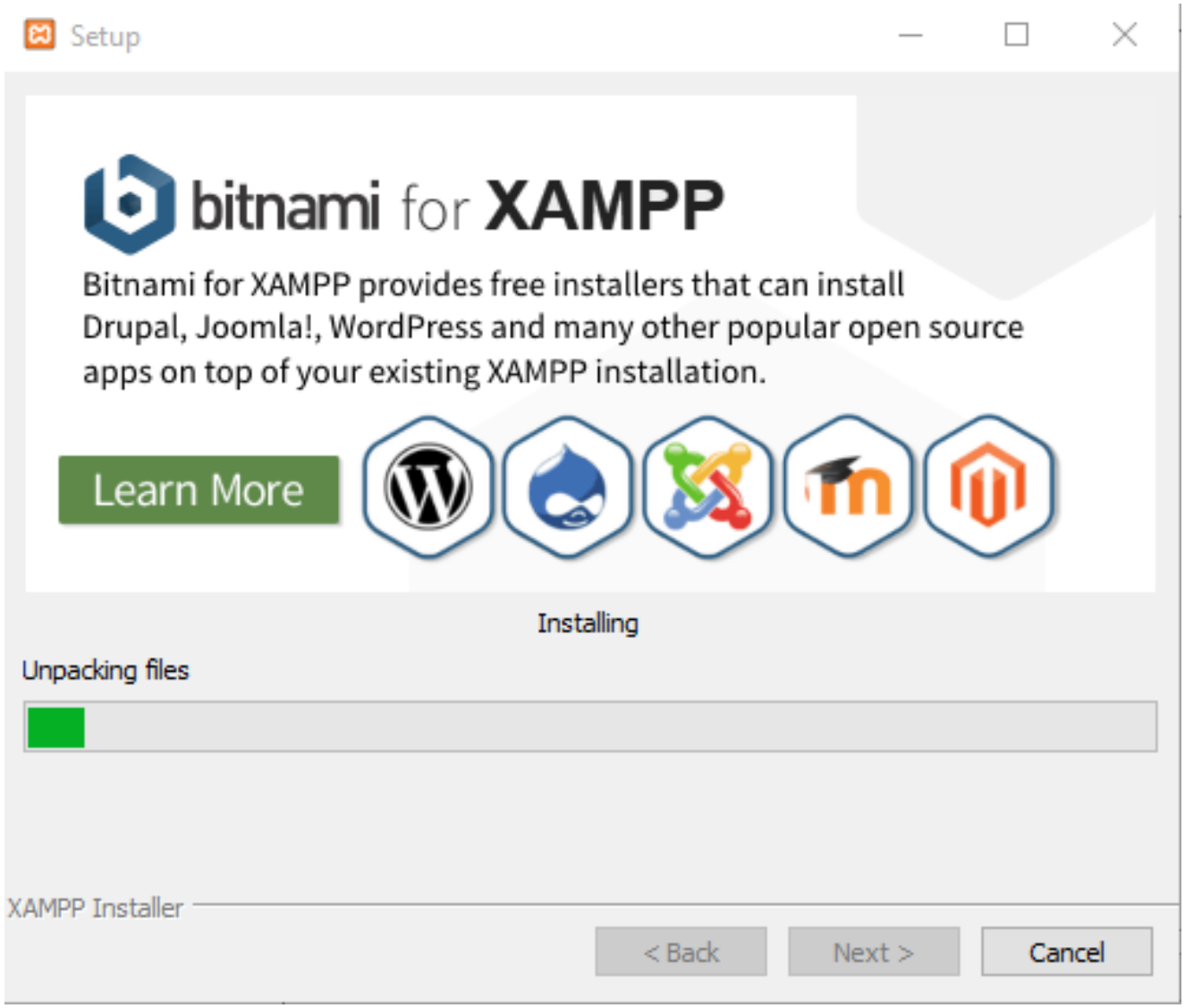
Install XAMPP

- Install in C:\XAMPP



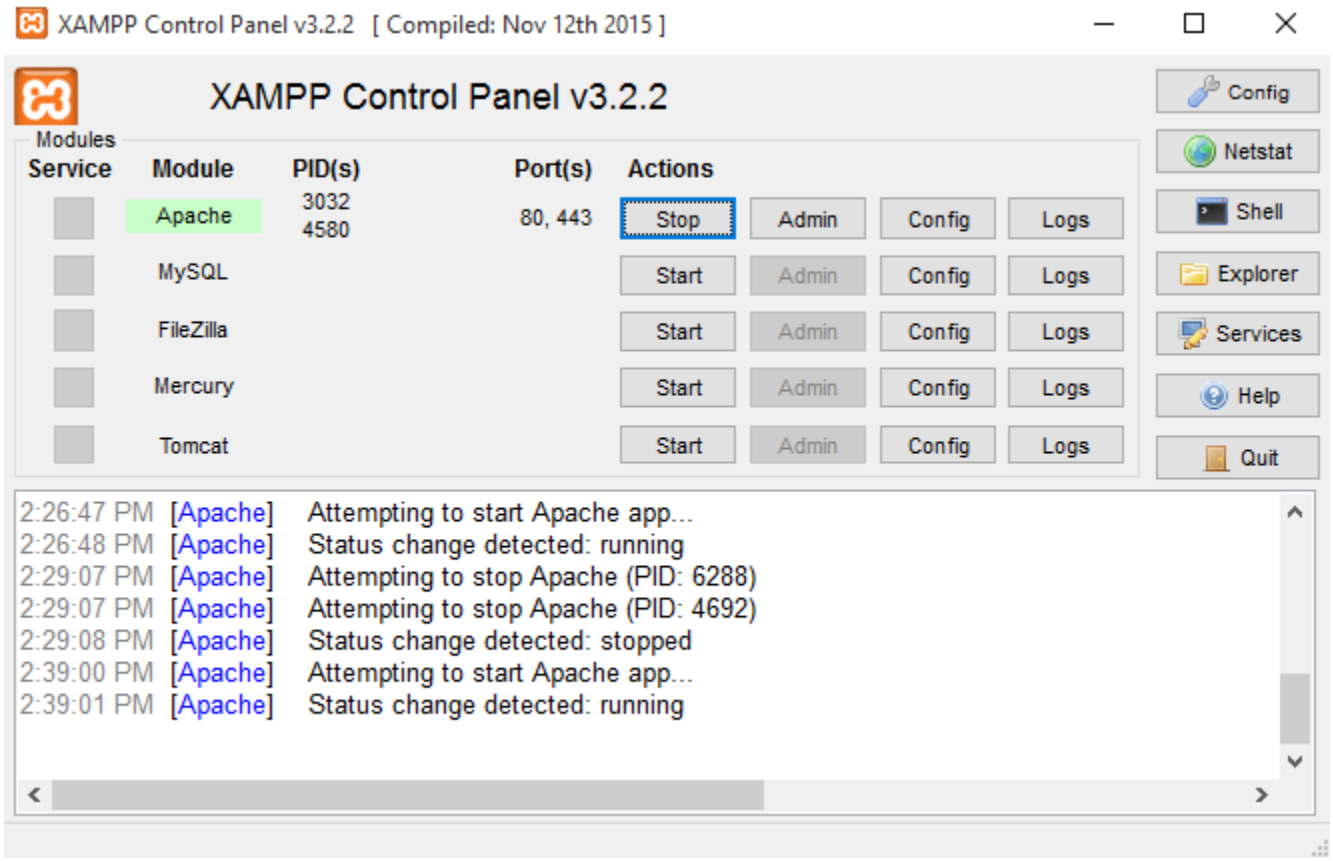


Install XAMPP



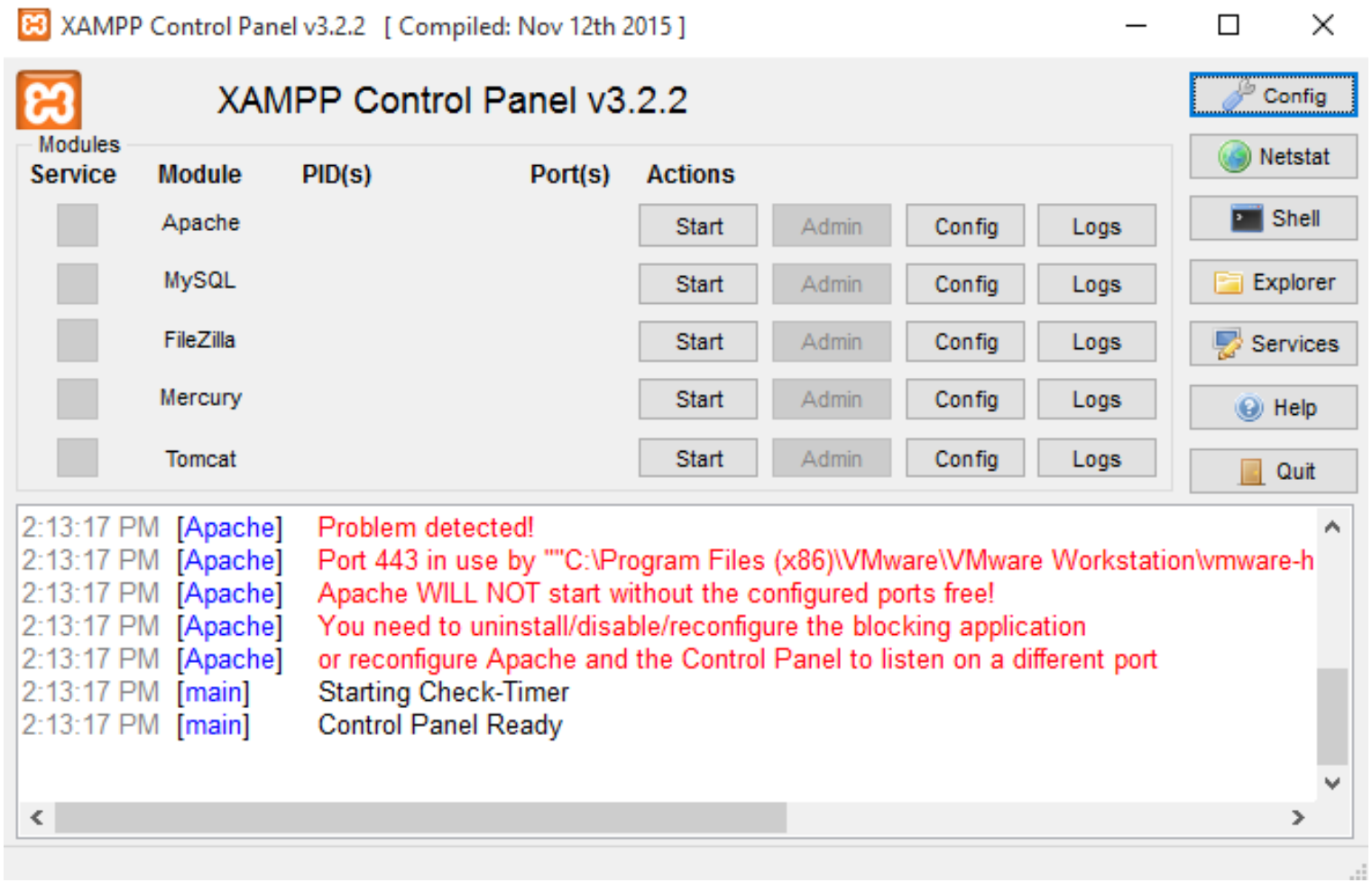
Starting XAMPP

- Run **XAMPP** Control Panel as administrator
- If no error then **Apache** will start successfully



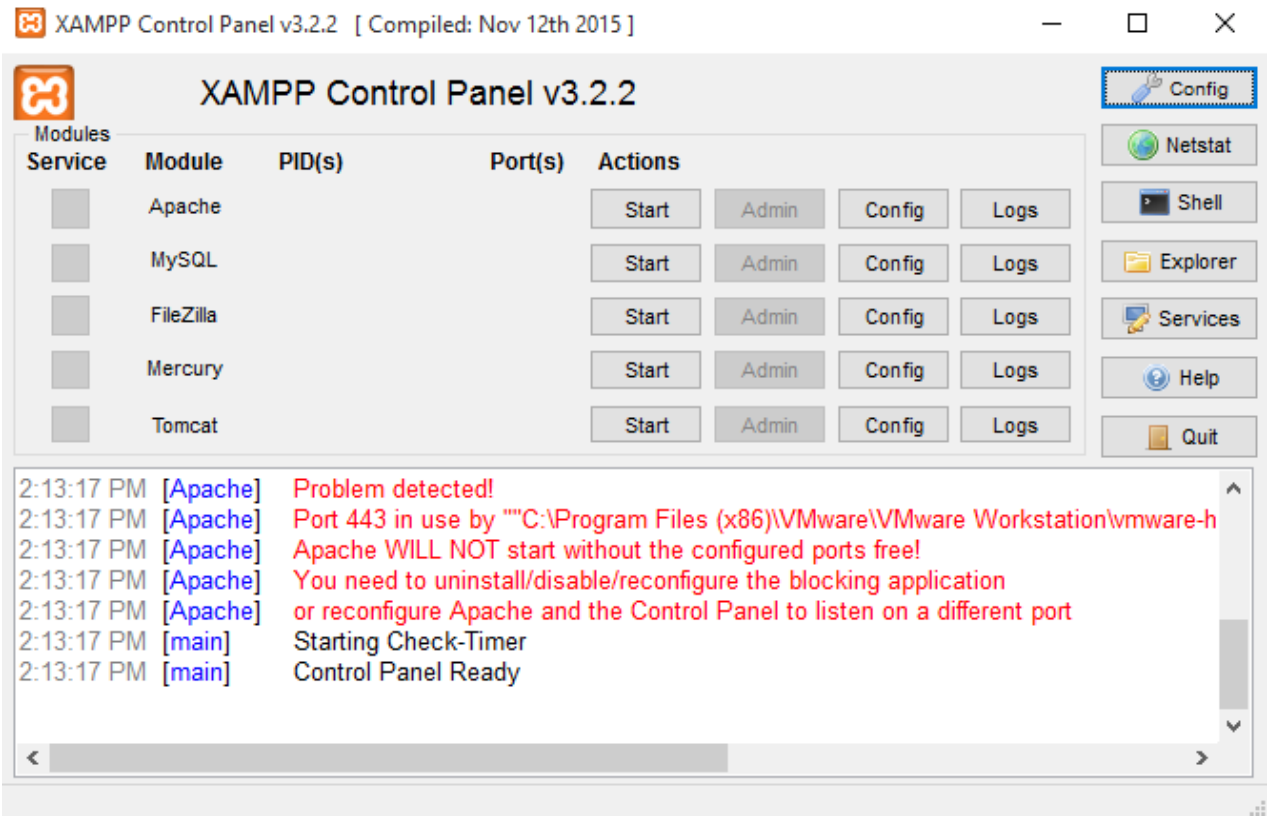
Starting XAMPP

- But if **error** -



Starting XAMPP

- Run XAMPP Control Panel as administrator



- **Error** : **Port** is being **used** Vmware (or may be Skype) , so . . .

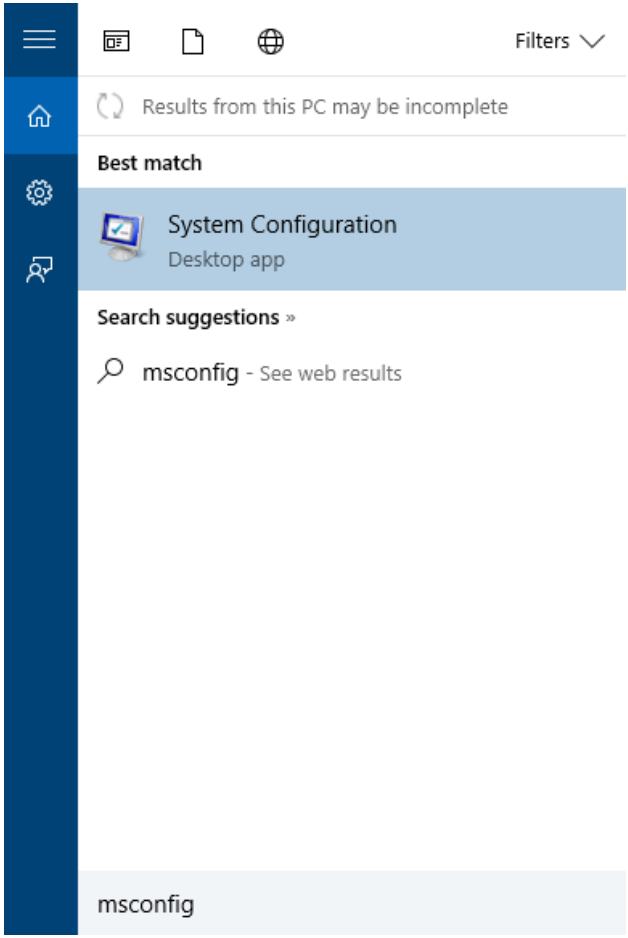
Starting XAMPP

- Run XAMPP Control Panel as administrator
- Error : Port is being used Vmware (or may be Skype) , so . . .
- **Alt** + **Ctrl** + **Delete** to start **Task Manager**
- Stop (Kill) all VMware (or Skype . .) **Processes**
- Stop all VMware **Services**
- Start XAMPP again

Starting XAMPP

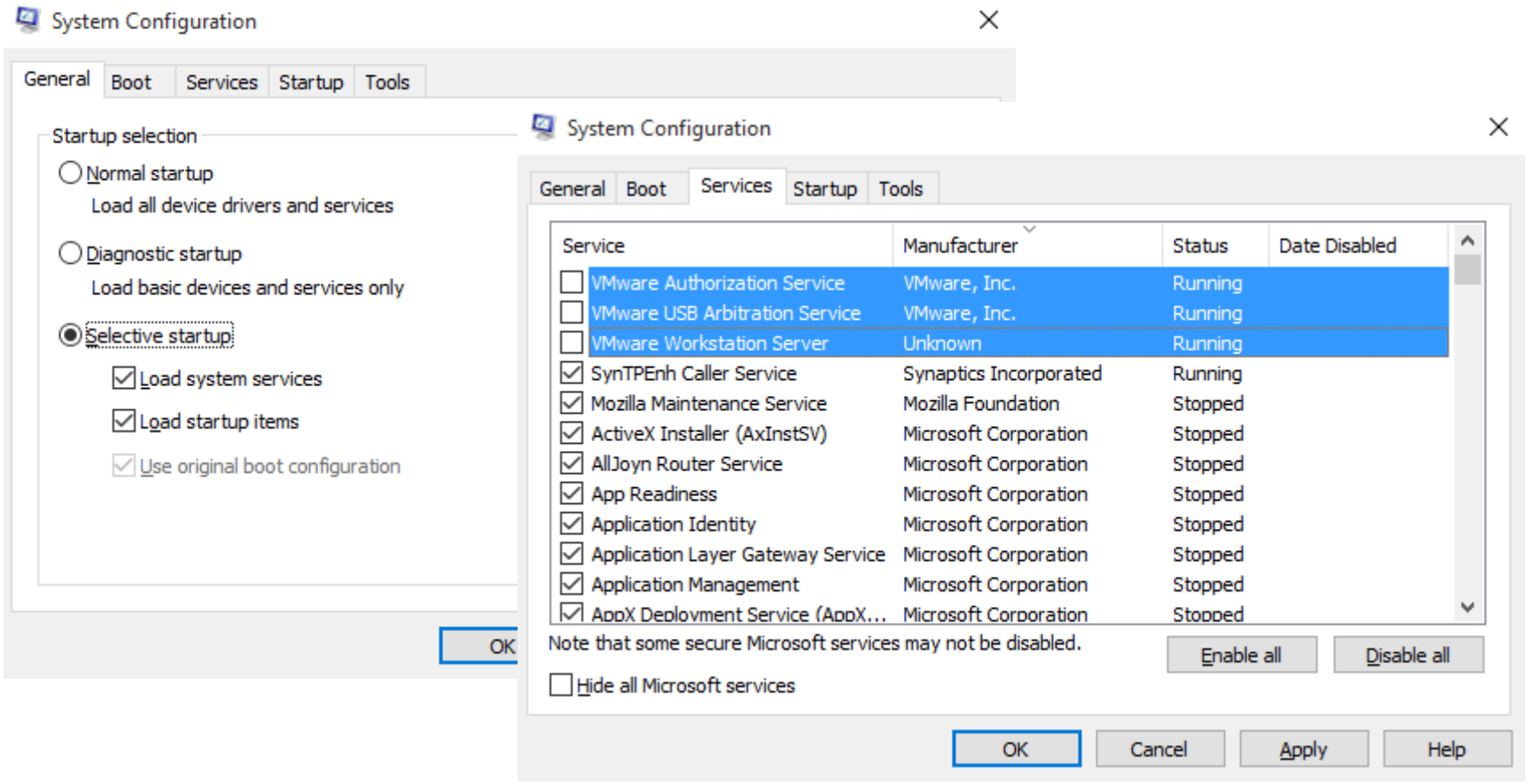
- Run XAMPP Control Panel as administrator
- Error : Port is being used Vmware (or may be Skype) , so . . .
- Alt + Ctrl + Delete to run Task Manager
- Stop (Kill) all VMware Processes
- And Stop all VMware Services
- Start XAMPP again

- If it does not start then :
- Click Windows button , type **msconfig**
- Run as administrator



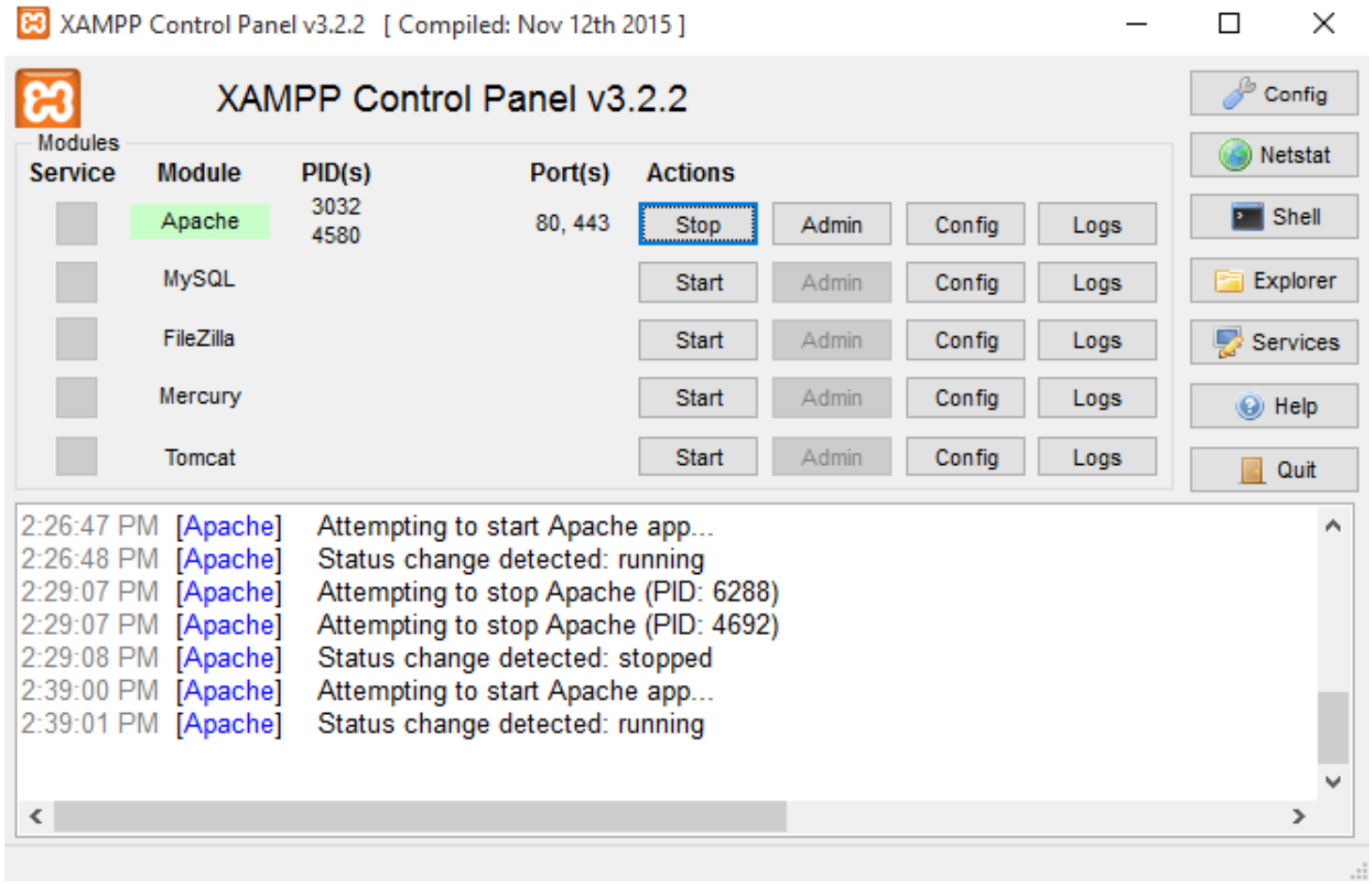
msconfig

- General -> Select Selective startup
- Service -> Unselect all VMware services , Apply and restart



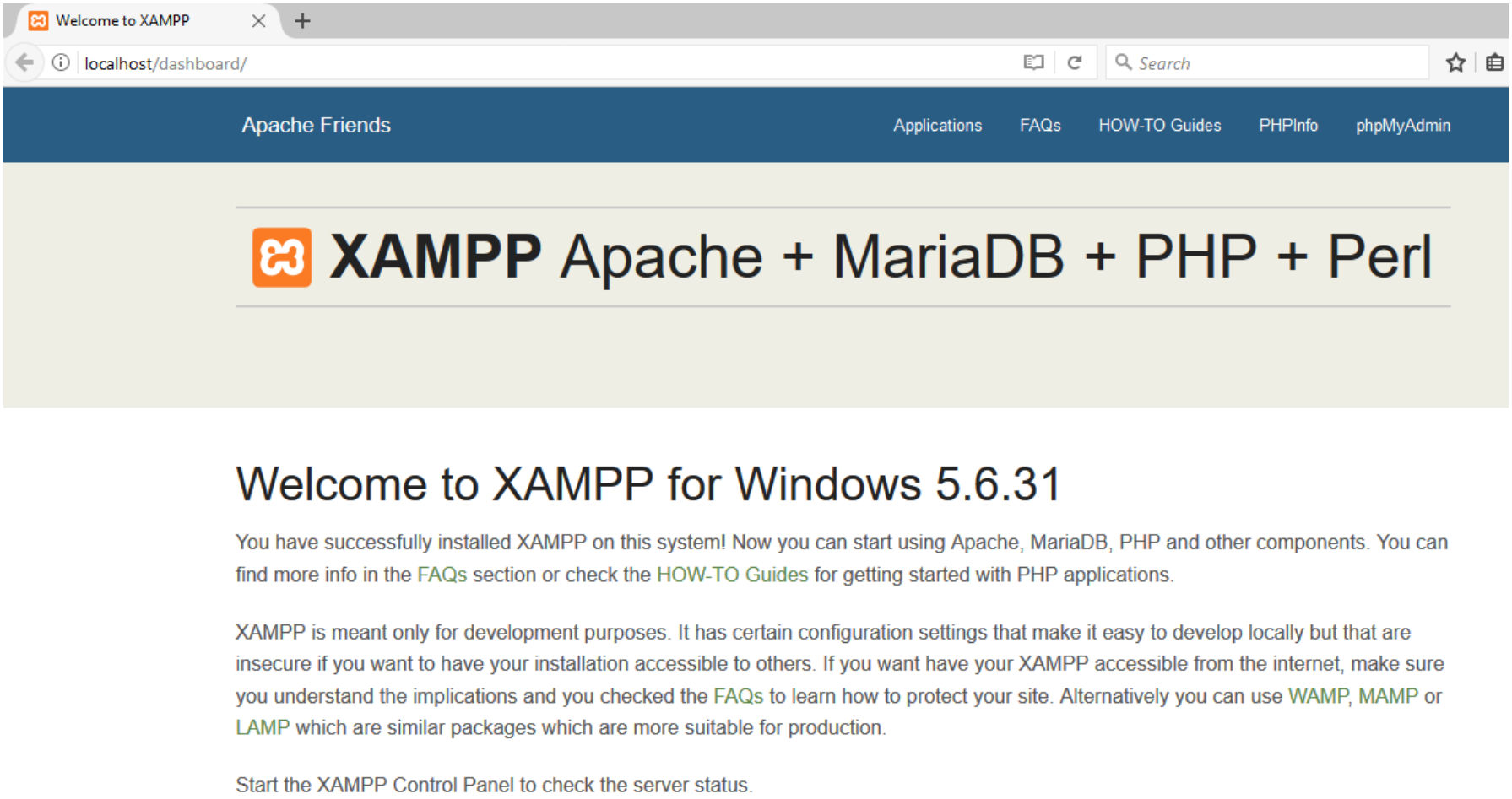
XAMPP

- Run XAMPP Control Panel as administrator
- Click **Admin**



Default home page : http://localhost

- A defult page opens with link - http://localhost



CGI

- Common Gateway Interface (CGI) programming
- Static vs Dynamic content
- CGI defines a way for a web server to interact with external content-generating programs
- It is a simple way to put dynamic content web site, using any programming language
- Widely used : Java Servlet , ASP.NET , Python , Ruby , Perl
- In order to get your CGI programs to work properly, you'll need to have Apache configured to permit CGI execution
- .htaccess , httpd.conf

[http://httpd.apache.org/docs/current/howto/cgi.html]

CGI

- Differences between ''regular'' programming, and CGI programming :

  All output from CGI program must be preceded by a **MIME-type** header
  This is HTTP header that tells the client what sort of content it is receiving.

  Most of the time, this will look like:

  Content-type: text/html

CGI

- Differences between ''regular'' programming, and CGI programming :

  Output needs to be in HTML, or some other format that a browser will be able to display

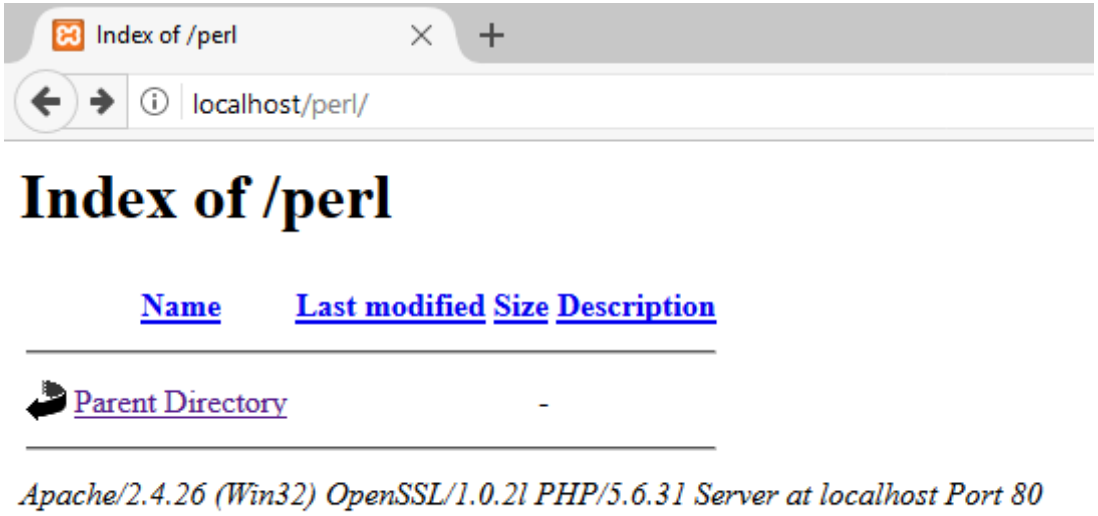
  Most of the time, this will be HTML, but occasionally you might write a CGI program that outputs a gif image, or other non-HTML content

  CGI program will look a lot like any other program

Webpages, Perl, PHP pages

- Default page is rendered from location C:\xampp**htdocs**
- Custom HTML webpages, Perl, PHP pages can be saved in new folders created under C:\xampp**htdocs**

- Create new folder called as "**perl**" in **htdocs**
- Can be accessed as :



Webpages, Perl, PHP pages

- Default page is rendered from location C:\xampp**htdocs**
- Custom HTML webpages, Perl, PHP pages can be saved in new folders created under C:\xampp**htdocs**

- Create new folder called as "**perl**"
- Suppose a new file called as "**someFile.pl**" was created in **perl** folder. What will be the URL to access someFile.pl ?

Webpages, Perl, PHP pages

- Default page is rendered from location C:\xampp**htdocs**
- Custom HTML webpages, Perl, PHP pages can be saved in new folders created under C:\xampp**htdocs**

- Create new folder called as "**perl**"
- Suppose a new file called as "**someFile.pl**" was created in **perl** folder. What will be the URL to access someFile.pl ?

  http://localhost/perl/someFile.pl

Webpages, Perl, PHP pages

- Default page is rendered from location C:\xampp**htdocs**
- Custom HTML webpages, Perl, PHP pages can be saved in new folders created under C:\xampp**htdocs**

- Create new folder called as "**perl**"
- Suppose a new file called as "**someFile.pl**" was created in **perl** folder. What will be the URL to access someFile.pl ?

  http://localhost/perl/someFile.pl

- Generally on UNIX like systems : custom HTML are saved under /**var/www**/ or **/opt/lampp**/htdocs

[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

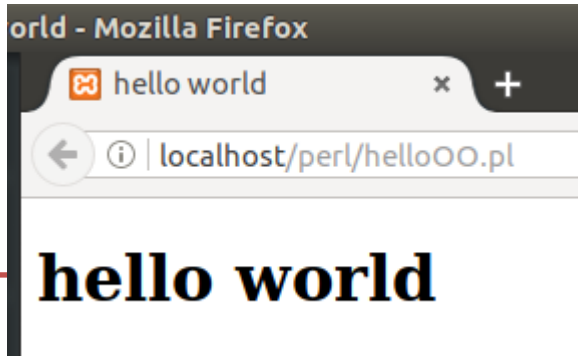
```
#!C:\xampp\perl\bin\perl
use CGI qw/:standard/;   # load standard CGI routines
```

[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

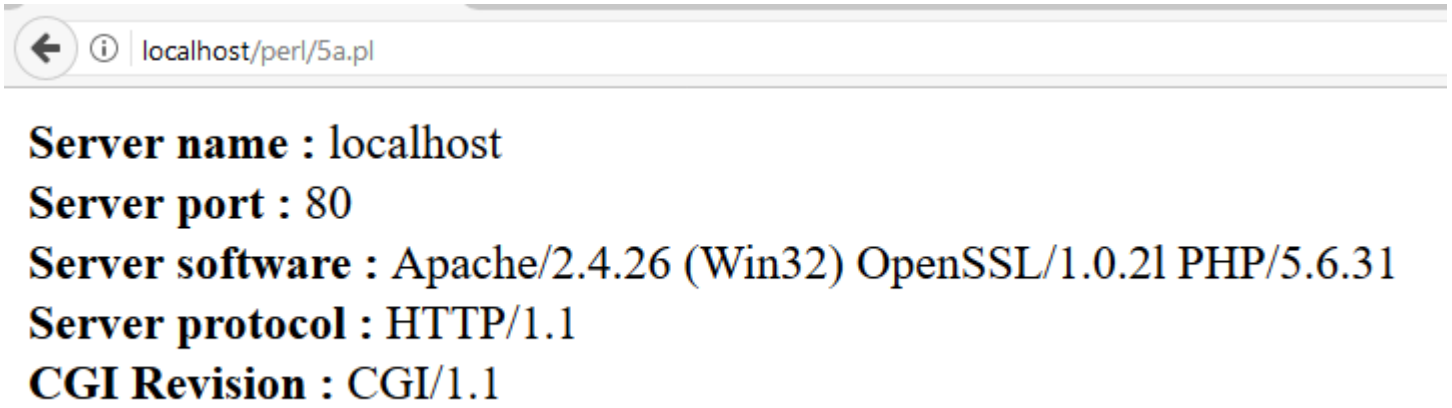
```
print " ENV Variable <strong> Name = Value </strong> <br/>";
foreach my $key ( keys %ENV )
 {
   print " $key = $ENV{$key} <br/>";
 }
```

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

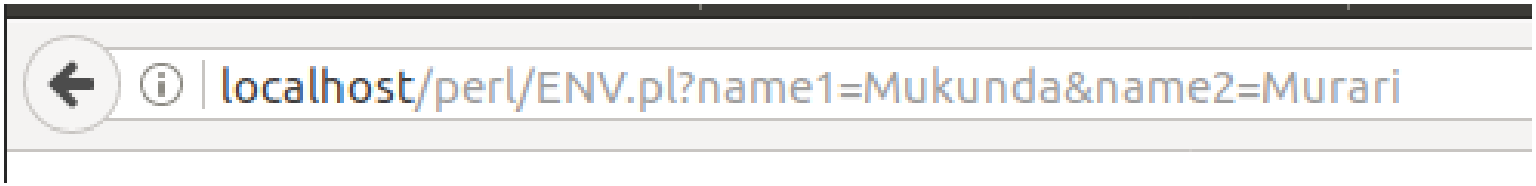

localhost/perl/5a.pl
Server name : localhost
Server port : 80
Server software : Apache/2.4.26 (Win32) OpenSSL/1.0.2l PHP/5.6.31
Server protocol : HTTP/1.1
CGI Revision : CGI/1.1

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/>";
```

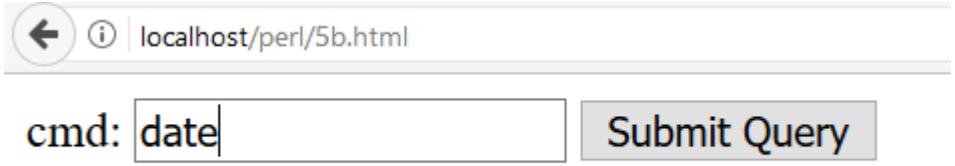
5b. Pseudo Code / Outline of the Algorithm

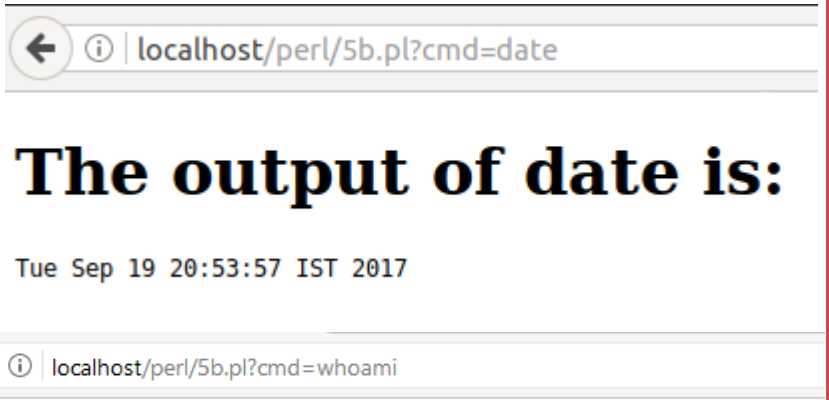
```
<html>
 <body>
   <form action="5b.pl">
     cmd: <input type="text" name="cmd"/>
          <input type="submit"/>
   </form>
 </body>
</html>
```

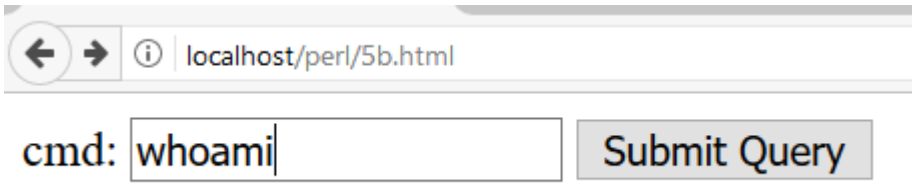
5b. Pseudo Code / Outline of the Algorithm

```
#!C:\xampp\perl\bin\perl
use CGI qw(:standard);
print header();
print start_html();
$cmd = param("cmd");
print "<h1>The output of $cmd is:</h1>";
print "<pre>".`$cmd`."</pre>";
print end_html();
```

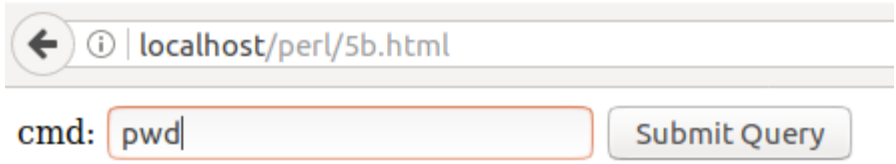
Sample Run


localhost/perl/5b.html
cmd: date
Submit Query


localhost/perl/5b.pl?cmd=date
The output of date is:
Tue Sep 19 20:53:57 IST 2017


localhost/perl/5b.html
cmd: whoami
Submit Query


localhost/perl/5b.pl?cmd=whoami
The output of whoami is:
desktop-65mqqvr\sagar

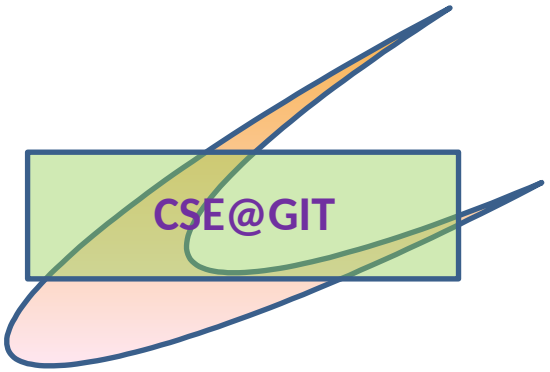

localhost/perl/5b.html
cmd: pwd
Submit Query


localhost/perl/5b.pl?cmd=pwd
The output of pwd is:
/opt/lampp/htdocs/perl

Experiment No. 6

Problem Definition: 6. a) Write a Perl program to accept the User Name and display a greeting message randomly chosen from a list of 4 greeting messages.

b) Write a Perl program to keep track of the number of visitors visiting the web page and to display this count of visitors, with proper headings.



Random number generation

- **int (rand (EXPR))**
- returns a random integer between 0 and EXPR , inclusive

```
print rand(10);
```

- returns a random integer between 0 and 9 , inclusive

- rand is not cryptographically secure
- Should not rely on it in security-sensitive situations

[https://perldoc.perl.org/functions/rand.html]

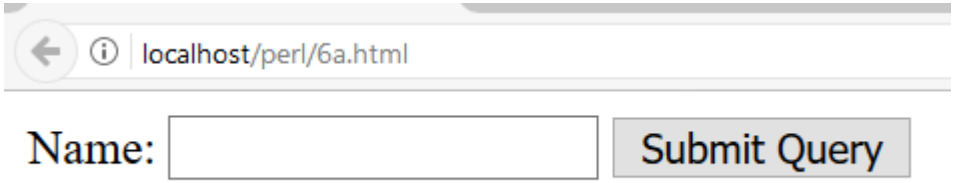
6a. Pseudo Code / Outline of the Algorithm

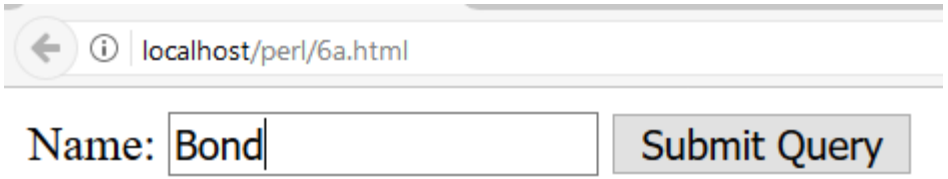
```
<html>
 <body>
  <form action="6a.pl">
   Name: <input type="text" name="name"/>
   <input type="submit"/>
  </form>
 </body>
</html>
```

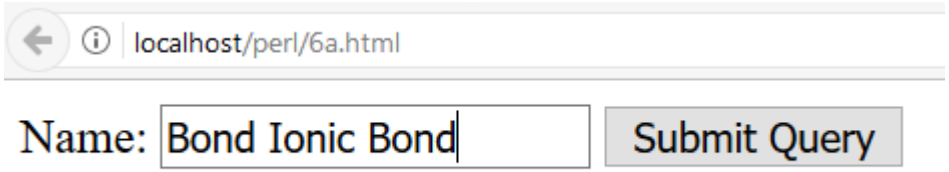
6a. Pseudo Code / Outline of the Algorithm

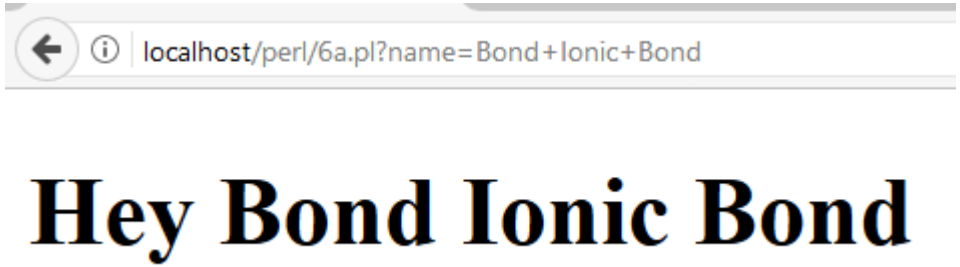
```
#!C:\xampp\perl\bin\perl
use CGI qw(:standard);
print header();
print start_html();

$name=param("name");
@arr = ("Hi","Hey","Hello","Welcome");
print "<h1> $arr[rand(4)] $name </h1>";

print end_html();
```

Sample Run

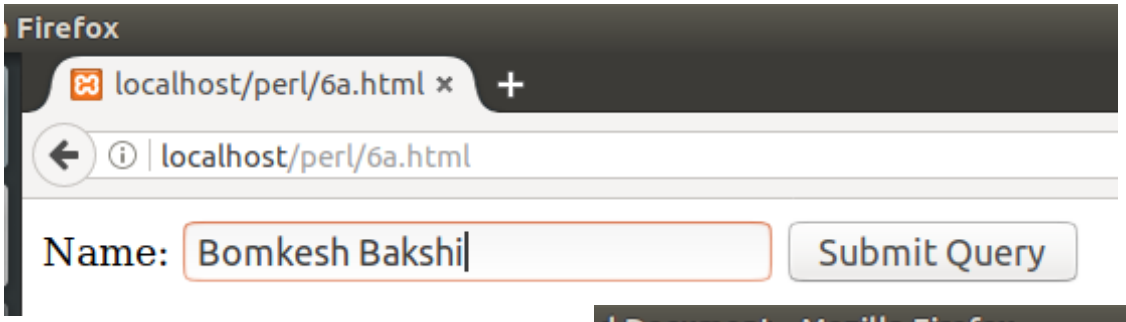

localhost/perl/6a.html
Name:
Submit Query

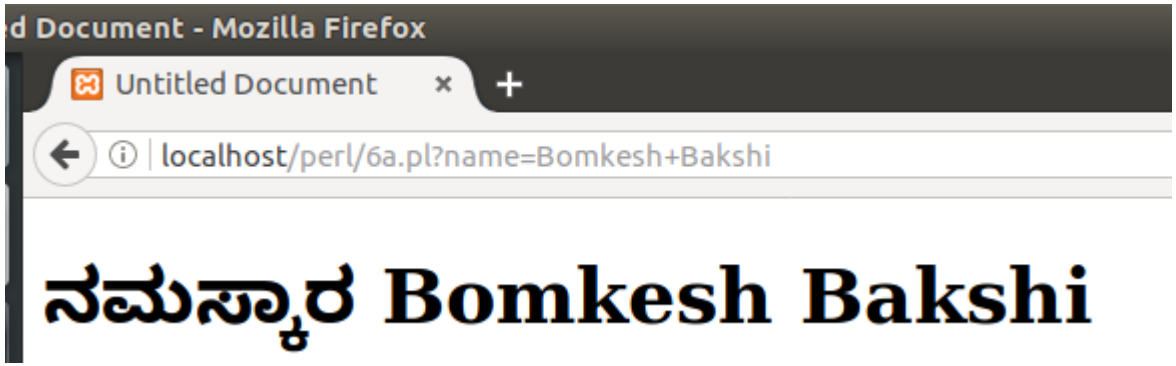

localhost/perl/6a.html
Name: Bond
Submit Query


localhost/perl/6a.html
Name: Bond Ionic Bond
Submit Query

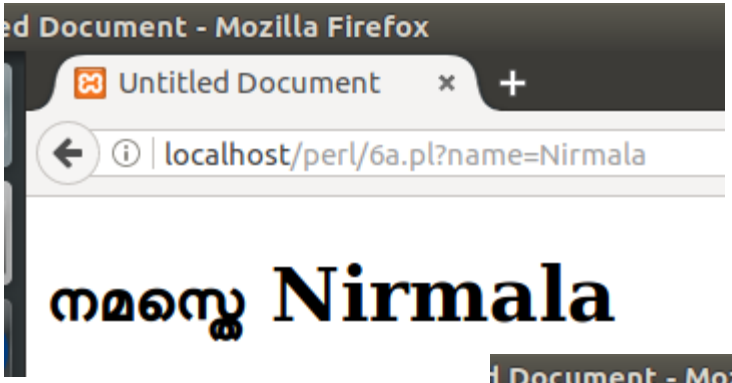

localhost/perl/6a.pl?name=Bond+Ionic+Bond
Hey Bond Ionic Bond

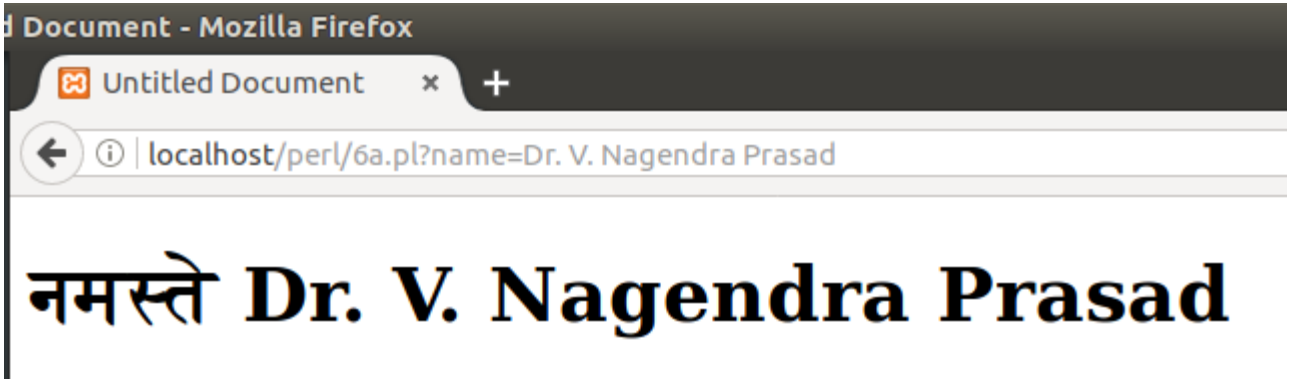
What changes have to be done for :

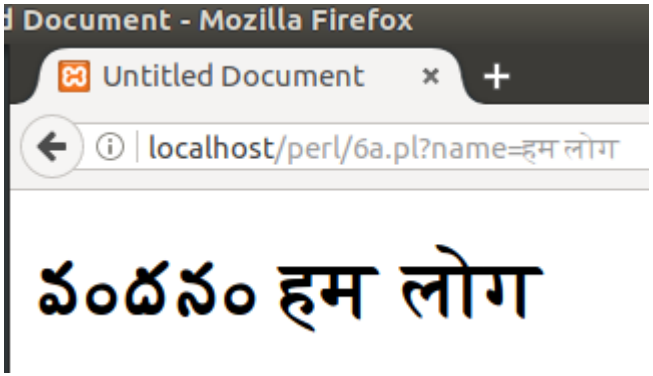




What changes have to be done for :







Changes:

```
print header(-charset=>'utf-8');
```

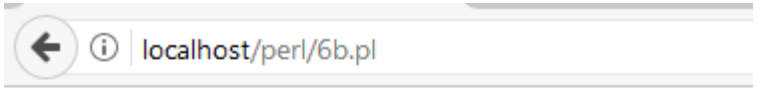
6b. Pseudo Code / Outline of the Algorithm

```
# When script 6b.pl is called

#   Open a file called as count.txt

#   Read from file count.txt
#     Read from file returns 0 if first run
#       as file does not have any content
#     Esle read returns number saved in count.txt

#   Save content read from file in variable count

#   Increment count

#   Write count to same file , But
#   Not append mode, over write existing content

#   Every time the script is called ,
#     remembers how many times the script invoked
```

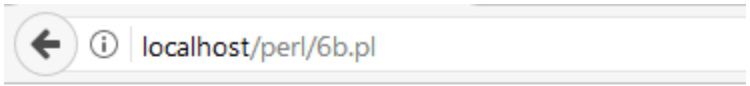
6b. Pseudo Code / Outline of the Algorithm

```
#!C:\xampp\perl\bin\perl
use CGI qw(:standard);
print header();
print start_html();

open(FILE,"<count.txt");
$count=<FILE>;
close(FILE);

$count++;

open(FILE,">count.txt");
print FILE $count;
close(FILE);

print "<h1>Accessed $count times</h1>";

print end_html();
```

Sample Run



Accessed 1 times



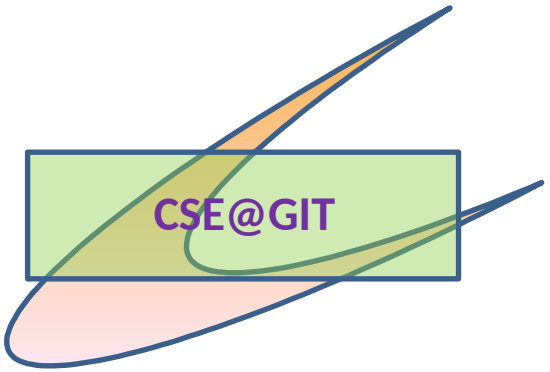
Accessed 2 times

Sample Run

```
</html>rahul@WhySoSerious:/opt/lampp/htdocs/perl$ perl 6b.pl
Content-Type: text/html; charset=ISO-8859-1

<!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>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<h1>Accessed 31 times</h1>
</body>
</html>rahul@WhySoSerious:/opt/lampp/htdocs/perl$ perl 6b.pl
Content-Type: text/html; charset=ISO-8859-1

<!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>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<h1>Accessed 32 times</h1>
</body>
```

Experiment No. 7

Problem Definition: 7. Write a Perl program to display a digital clock which displays the current time of the server.



time

- **time()** : returns the number of non-leap seconds since whatever time the system considers to be the epoch, suitable for feeding to "gmtime" and "**localtime**".
- On most systems the epoch is 00:00:00 UTC, January 1, 1970

[perldoc -f time]

localtime

- **localtime** : Converts a time as returned by the time function to a 9-element list with the time analyzed for the local time zone

```
#     0    1    2     3     4    5     6     7     8
my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
                                        localtime(time);
```

- All list elements are numeric and come straight out of the C `struct tm'
- **$sec** , **$min** , and **$hour** are the seconds, minutes, and hours of the specified time
- **$mday** is the day of the month
- **$mon** the month in the range 0..11 , with 0 indicating January and 11 indicating December
- **$year** contains the number of years since 1900

localtime

- **localtime** : Converts a time as returned by the time function to a 9-element list with the time analyzed for the local time zone

```
#     0    1    2     3     4    5     6     7     8
my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
                                        localtime(time);
```

- **$wday** is the day of the week, with 0 indicating Sunday and 3 indicating Wednesday
- **$yday** is the day of the year, in the range 0..364 (or 0..365 in leap years)
- **$isdst** is true if the specified time occurs during Daylight Saving Time, false otherwise

[https://perldoc.perl.org/functions/localtime.html]

Refresh

- To refresh page ever second :
  Include refresh: 1 before create header function call is made

```
use CGI qw(:standard);
print "refresh: 1\n";
print header();
```

  or

- Include REFRESH property of header function

```
use CGI qw(:standard);
print header(-REFRESH=>"1");
print start_html();
```

  or

- Meta tag of start_html

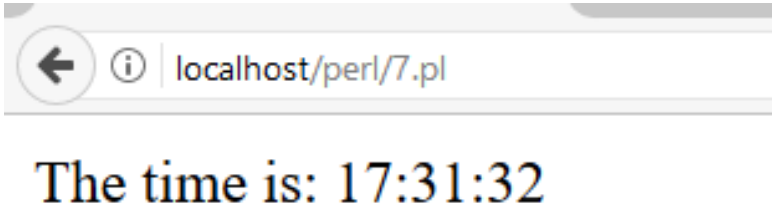
```
print header();
print start_html(-head=>meta({-http_equiv => 'Refresh',
      -content=> '1;URL=http://localhost/perl/7.pl'}));
```

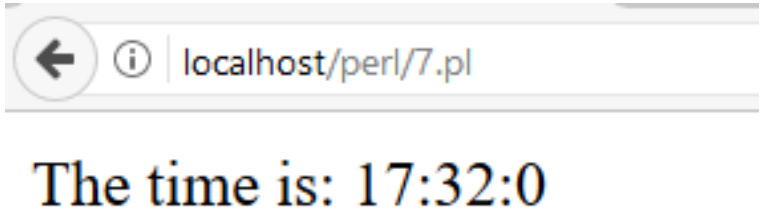
[https://perldoc.perl.org/5.8.8/CGI/Push.html]

7. Pseudo Code / Outline of the Algorithm

```
#!C:\xampp\perl\bin\perl
use CGI qw(:standard);
print "refresh: 1\n";
print header();
print start_html();

($s,$m,$h)=localtime(time());
print "The time is: $h:$m:$s";

print end_html();
```

Sample Run


localhost/perl/7.pl
The time is: 17:31:32


localhost/perl/7.pl
The time is: 17:32:0

Learning Outcomes of the Experiment

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

1) Explain the features of Perl. [L2]

2) Experiment with the usage of basic programming concepts like variables, data types and conditional and looping statements [L3]

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