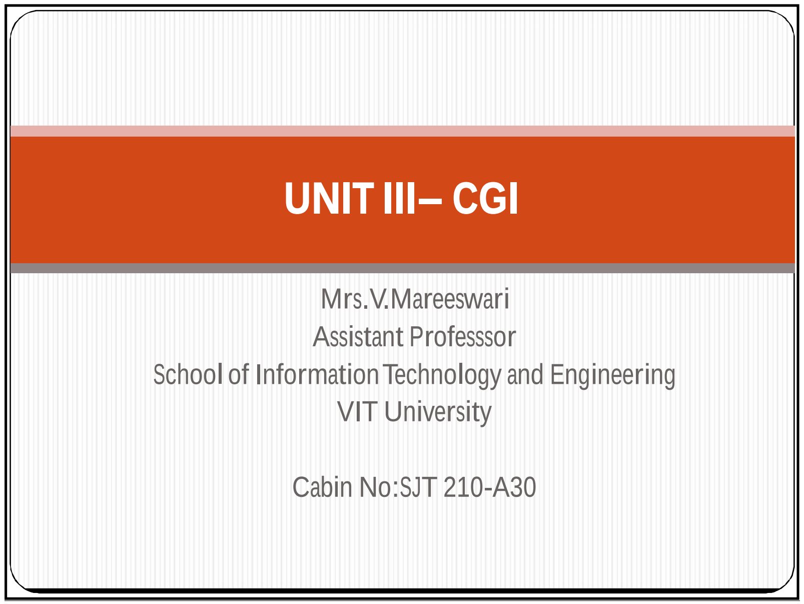
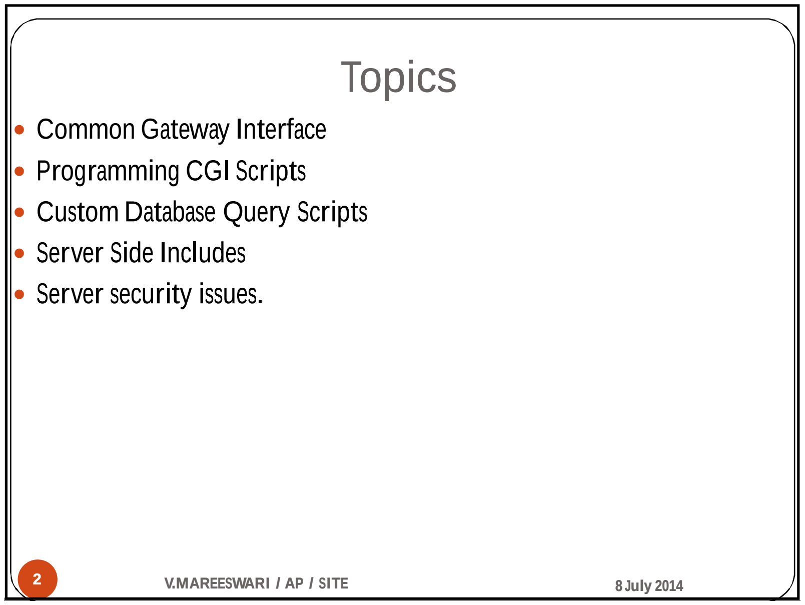


UNIT III– CGI

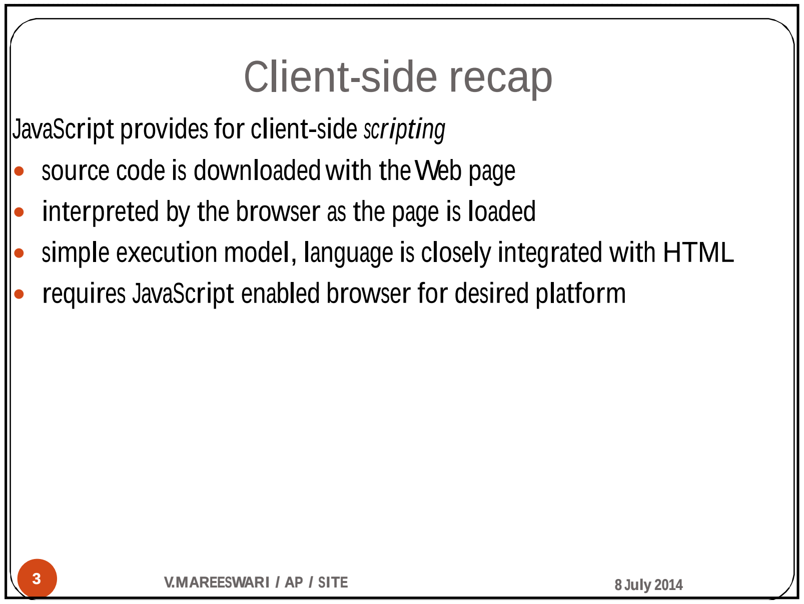
Mrs.V.Mareeswari
Assistant Professsor
School of Information Technology and Engineering
VIT University

Cabin No:SJT 210-A30



Topics

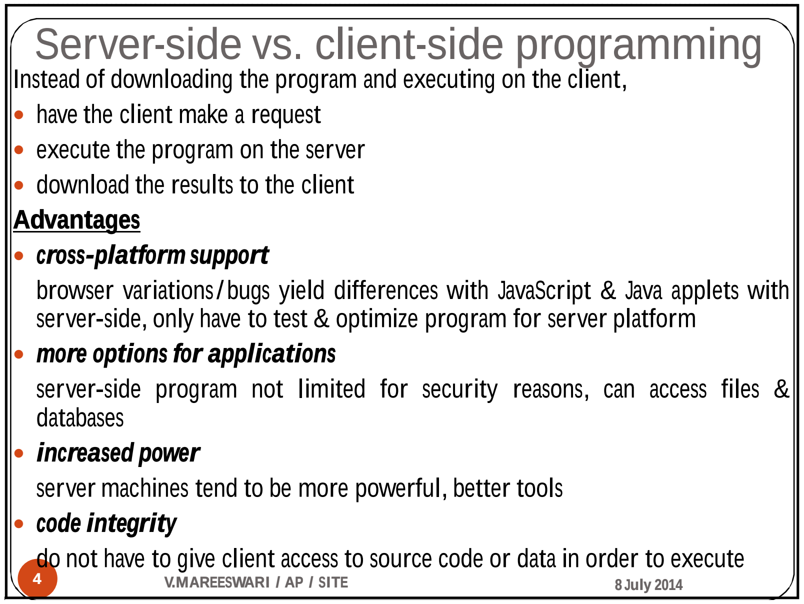
- Common Gateway Interface
- Programming CGI Scripts
- Custom Database Query Scripts
- Server Side Includes
- Server security issues.



Client-side recap

JavaScript provides for client-side *scripting*

- source code is downloaded with the Web page
- interpreted by the browser as the page is loaded
- simple execution model, language is closely integrated with HTML
- requires JavaScript enabled browser for desired platform



Server-side vs. client-side programming

Instead of downloading the program and executing on the client,

- have the client make a request
- execute the program on the server
- download the results to the client

Advantages

- ***cross-platform support***

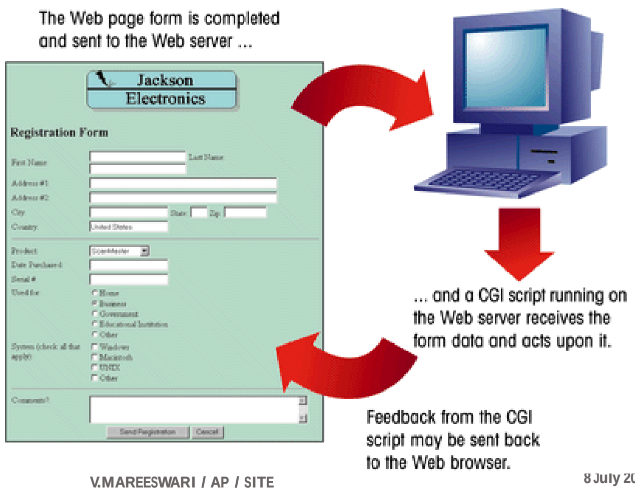
  browser variations/bugs yield differences with JavaScript & Java applets with server-side, only have to test & optimize program for server platform
- ***more options for applications***

  server-side program not limited for security reasons, can access files & databases
- ***increased power***

  server machines tend to be more powerful, better tools
- ***code integrity***

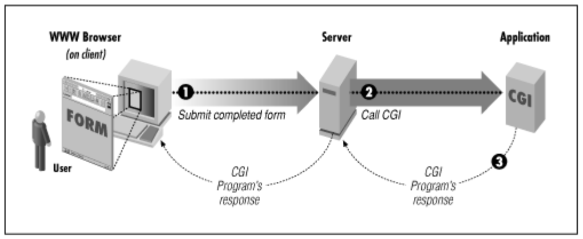
  do not have to give client access to source code or data in order to execute

The Interaction between a Web Page Form and a CGI Script


The Web page form is completed and sent to the Web server ...
Jackson Electronics
Registration Form
... and a CGI script running on the Web server receives the form data and acts upon it.
Feedback from the CGI script may be sent back to the Web browser.

Introduction to CGI

- **CGI (Common Gateway Interface)** script runs on the Web server and receives data from a form and uses it to perform a set of tasks.
- **CGI is the part of the Web server that can communicate with other programs running on the server.**
- CGI is not a programming language. It is just a set of standards.
- CGI solutions are often written in Perl, PHP, Basic, C, C++ and other languages.



CGI programs work as follows:

- STEP 1 (On the client side): Get Information from the user (using HTML forms, SSI, Java Applet, …,etc).
- STEP 2 (On the server side): Process the data, connect to DATABASE, search for PATTERNS, …,etc.
- STEP 3 (On the server side): Send the result of computation with adding appropriate HTTP headers back to the client.

Requirements

- Web server (NCSA, Apache, IIS, Microsoft Personal Web server etc.)
- Compiler (C/C++) or Interpreter (Perl), PHP, ASP
- Web browser (NN, IE etc.)

- CGI scripts are frequently used to generate special purpose web output, process input forms, handle user feedback, search web sites and types of other databases, run web chat rooms, add guestbook services, score online exams, and much more.
- There are several reasons to restrict direct access to CGI scripts:
  - CGI scripts are actually running a program directly on the server
  - **security risks** from computer hackers
  - Drain on system resources caused by large numbers of programs running simultaneously
- System administration are understandably careful to maintain strict control over their servers and systems.

Format of HTTP messages

Browser Request

For the simple hypertext link in an HTML document:

Browser will send a request of the following type:

```
GET /test.html HTTP/1.0
Accept: text/plain
Accept: text/html
Two blank lines
```

Server Response

```
HTTP /1.0 200 OK
Date: Monday, 24-Dec-2000
11:09:05 GMT
Server: NCSA/1.3
MIME-version 1.0
Content-type: text/html
Content-length: 231
<HTML><HEAD><TITLE>Test
  Page</TITLE></HEAD>
<BODY>
This is a simple HTML page.
</BODY>
</HTML>
```

9 V.MAREESWARI / AP / SITE 8 July 2014

Environment Variables

What are they used for?

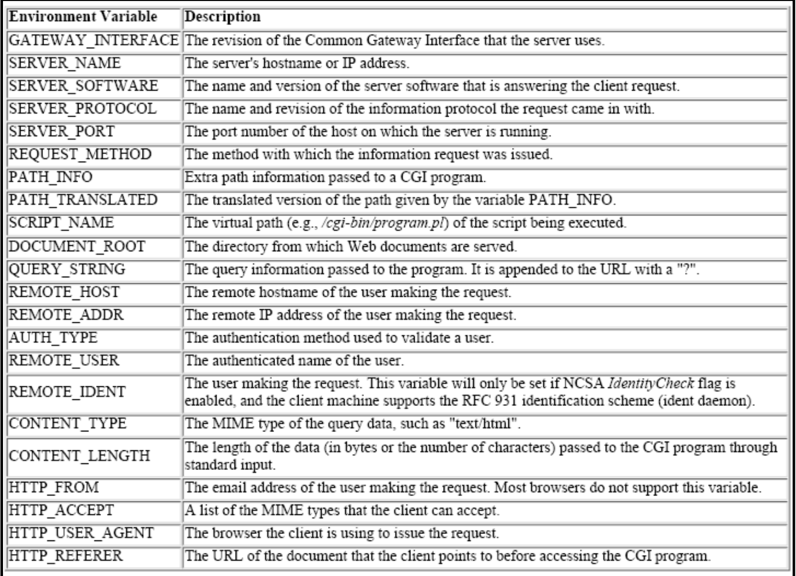
- In order to pass data from the server to the script, the server uses command line arguments along with environment variables.
- The Environment Variables are set when the server executes a CGI Script.
- Environment Variables allow the CGI Script to reference variables that might be wanted for the Script output.
- Environment variables are stored in a hash named %ENV
- There are two types of environment variables:
  - **Non-Request specific variables** - those set for every request
  - **Request specific variables** - those that are dependent on the request being fulfilled by the CGI Script

10 V.MAREESWARI / AP / SITE 8 July 2014

Environment Variables

- SERVER_NAME → The server's Host name or IP address
- SERVER_SOFTWARE → The name and version of the server-software that is answering the client requests
- SERVER_PROTOCOL → The name and revision of the information protocol the request came in with.
- REQUEST_METHOD → The method with which the information request was issued.
- QUERY_STRING → The query information passed to the program. It is appended to the URL with a "?"
- CONTENT_TYPE → The MIME type of the query data, such as "text/html"
- CONTENT_LENGTH → The length of the data in bytes, passed to the CGI program through standard input.
- HTTP_USER_AGENT → The browser the clients is using to issue the request.
- DOCUMENT_ROOT → It displays the server document root directory

11 V.MAREESWARI / AP / SITE 8 July 2014



Environment Variable	Description
GATEWAY_INTERFACE	The revision of the Common Gateway Interface that the server uses.
SERVER_NAME	The server's hostname or IP address.
SERVER_SOFTWARE	The name and version of the server software that is answering the client request.
SERVER_PROTOCOL	The name and revision of the information protocol the request came in with.
SERVER_PORT	The port number of the host on which the server is running.
REQUEST_METHOD	The method with which the information request was issued.
PATH_INFO	Extra path information passed to a CGI program.
PATH_TRANSLATED	The translated version of the path given by the variable PATH_INFO.
SCRIPT_NAME	The virtual path (e.g., */cgi-bin/program.pl*) of the script being executed.
DOCUMENT_ROOT	The directory from which Web documents are served.
QUERY_STRING	The query information passed to the program. It is appended to the URL with a "?".
REMOTE_HOST	The remote hostname of the user making the request.
REMOTE_ADDR	The remote IP address of the user making the request.
AUTH_TYPE	The authentication method used to validate a user.
REMOTE_USER	The authenticated name of the user.
REMOTE_IDENT	The user making the request. This variable will only be set if NCSA *IdentityCheck* flag is enabled, and the client machine supports the RFC 931 identification scheme (ident daemon).
CONTENT_TYPE	The MIME type of the query data, such as "text/html".
CONTENT_LENGTH	The length of the data (in bytes or the number of characters) passed to the CGI program through standard input.
HTTP_FROM	The email address of the user making the request. Most browsers do not support this variable.
HTTP_ACCEPT	A list of the MIME types that the client can accept.
HTTP_USER_AGENT	The browser the client is using to issue the request.
HTTP_REFERER	The URL of the document that the client points to before accessing the CGI program.

V.MAREESWARI / AP / SITE 8 July 2014

Typical Environment Variables

- SERVER_SOFTWARE = Apache/1.3.14
- SERVER_NAME = www.ncsi.iisc.ernet.in
- GATEWAY_INTERFACE = CGI/1.1
- SERVER_PROTOCOL = HTTP/1.0
- SERVER_PORT = 80
- REQUEST_METHOD = GET
- HTTP_ACCEPT = 'image/gif, image/x-xbitmap, image/jpeg'
- SCRIPT_NAME = /cgi-bin/environment-example
- QUERY_STRING = /cgi-bin/environment.cgi?foo=1&bar=2
- REMOTE_HOST = ece.iisc.ernet.in
- REMOTE_ADDR = 144.16.64.3

Architecture of CGI Application

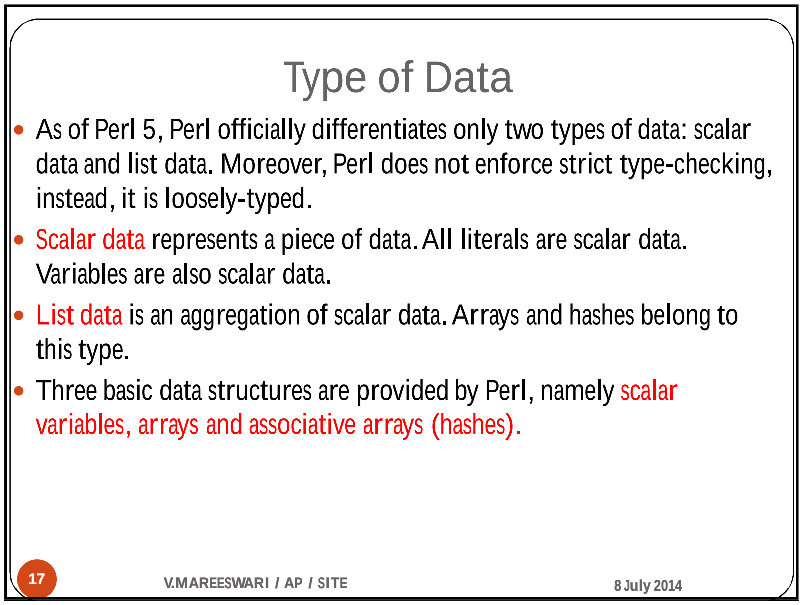
- In order to pass data about the information request from the server to the script, the server uses command-line arguments, as well as environment variables. These environment variables; set when the server executes the gateway program.
- The client connects to the Web server via the network. Several applications can reside on the server. The CGI interface acts as the gateway between the Web server and the server-side processes. One of the server-side processes could be a relational database. Although CGI acts as a gateway between any type of Web server and server-side applications, it has been more commonly implemented with the HTTP server.
- The client browser invokes the CGI script and waits until the CGI script completes its process. Thus, the CGI script must execute fast enough to have no perceived delay in the response time.
- The drawback of using CGI is that every time a CGI script is invoked, the Web server spawns a new process. This setup becomes a problem when a given Web site gets frequently accessed by several users.

PERL

- PERL stands for *Practical Extraction Report Language*.
- Is an interpreted language.
- Runs on multiple platforms:Windows, Unix, Mac, …, etc.
- Is a scripting language.
- Is a typeless language.
- Has some aspects similar to C language.
- Could be as procedural as you want it to be.
- Could be as object-oriented as you want it to be (PERL 5).
- Perl is a free, open source programming language created by Larry Wall.
- Perl is probably best known for text processing -- dealing with files, strings, and regular expressions.
- Perl is the most popular scripting language used to write scripts that utilize the Common Gateway Interface (CGI)

Syntax

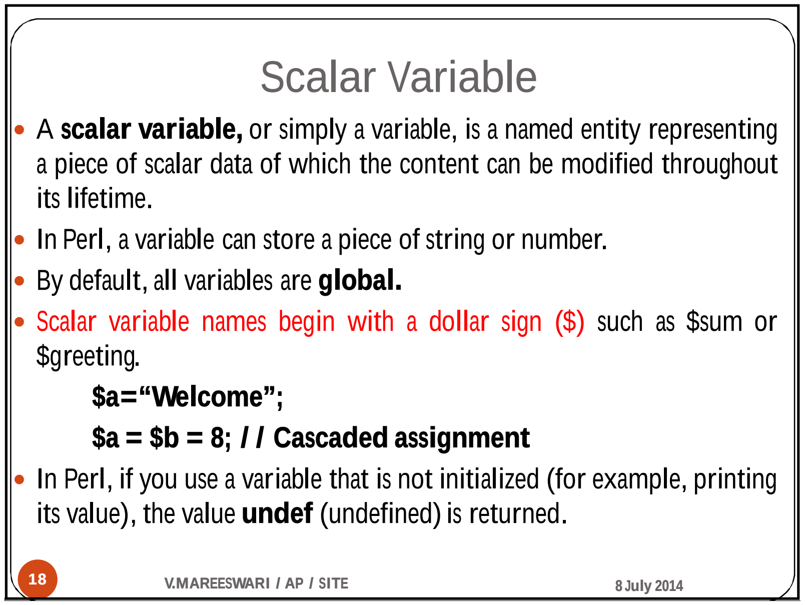
- A Perl script consists of statements, and each statement is terminated with a semicolon (;)
- print() is an example of a builtin function. A function usually accepts a number of parameters, or arguments.
- Comments begin with a "#" and extend to the end of the line.
- Variables are case sensitive.
- Identifiers cannot be longer than 255 characters



Type of Data

- As of Perl 5, Perl officially differentiates only two types of data: scalar data and list data. Moreover, Perl does not enforce strict type-checking, instead, it is loosely-typed.
- Scalar data represents a piece of data. All literals are scalar data. Variables are also scalar data.
- List data is an aggregation of scalar data. Arrays and hashes belong to this type.
- Three basic data structures are provided by Perl, namely scalar variables, arrays and associative arrays (hashes).

17 V.MAREESWARI / AP / SITE 8 July 2014



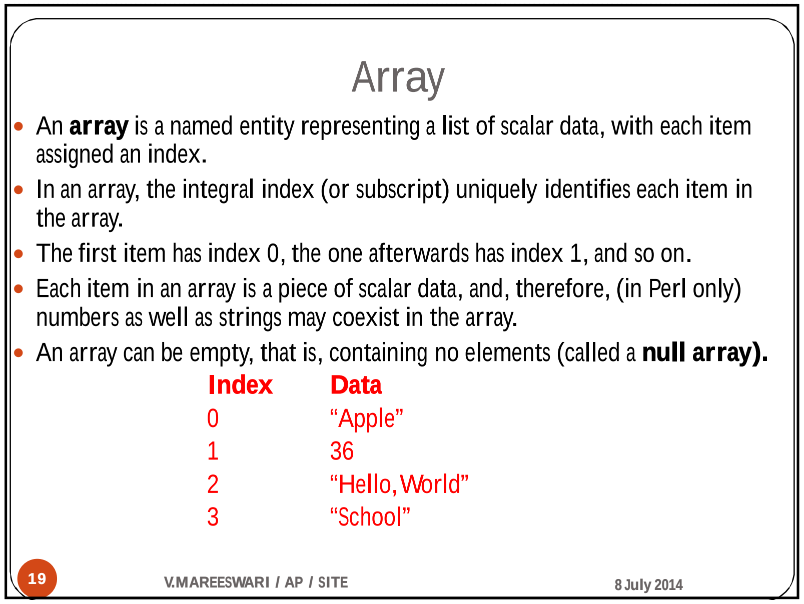
Scalar Variable

- A **scalar variable,** or simply a variable, is a named entity representing a piece of scalar data of which the content can be modified throughout its lifetime.
- In Perl, a variable can store a piece of string or number.
- By default, all variables are **global.**
- Scalar variable names begin with a dollar sign ($) such as $sum or $greeting.

  $a="Welcome";

  $a = $b = 8; / / Cascaded assignment
- In Perl, if you use a variable that is not initialized (for example, printing its value), the value **undef** (undefined) is returned.

18 V.MAREESWARI / AP / SITE 8 July 2014

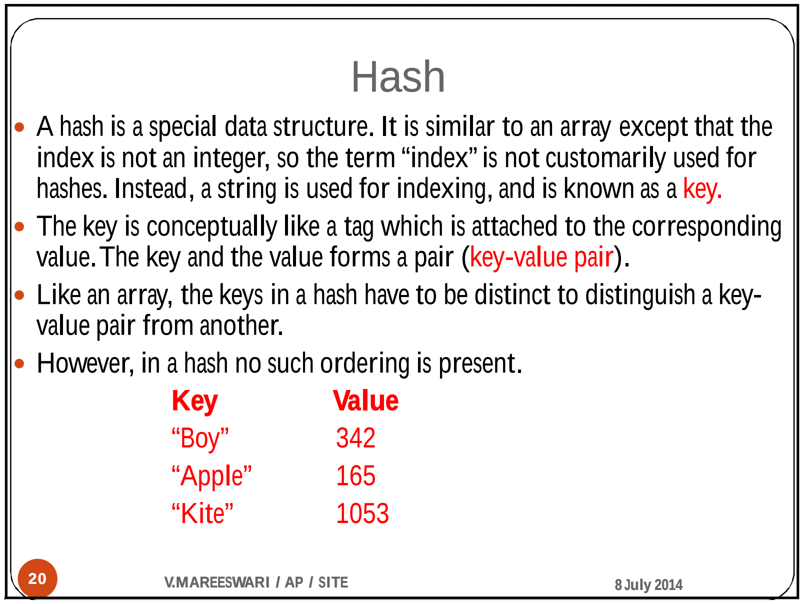


Array

- An **array** is a named entity representing a list of scalar data, with each item assigned an index.
- In an array, the integral index (or subscript) uniquely identifies each item in the array.
- The first item has index 0, the one afterwards has index 1, and so on.
- Each item in an array is a piece of scalar data, and, therefore, (in Perl only) numbers as well as strings may coexist in the array.
- An array can be empty, that is, containing no elements (called a **null array).**

Index	**Data**
0	"Apple"
1	36
2	"Hello, World"
3	"School"

19 V.MAREESWARI / AP / SITE 8 July 2014



Hash

- A hash is a special data structure. It is similar to an array except that the index is not an integer, so the term "index" is not customarily used for hashes. Instead, a string is used for indexing, and is known as a key.
- The key is conceptually like a tag which is attached to the corresponding value. The key and the value forms a pair (key-value pair).
- Like an array, the keys in a hash have to be distinct to distinguish a key-value pair from another.
- However, in a hash no such ordering is present.

Key	**Value**
"Boy"	342
"Apple"	165
"Kite"	1053

20 V.MAREESWARI / AP / SITE 8 July 2014

Strings

- Strings constants are enclosed within double quotes (") or in single quotes (').
- $x, inside a double quoted string is evaluated at run-time and the result is pasted into the string.
- Single quoted (') strings suppress all the special evaluation -- they do not evaluate \n or $x, and they may contain newlines.

```
$fname = "binky.txt";
$a = "Could not open the file $fname.\n";
## $fname evaluated and pasted in
$b = 'Could not open the file $fname.\n';
## single quotes (') do no special evaluation
print $a;
print $b;
```

Running Steps in command prompt

```
D:\IWP\Programs\Perl>set path=C:\Program Files\perl\bin;
D:\IWP\Programs\Perl>perl string.pl
  Could not open the file binky.txt.
  Could not open the file $fname.\n
D:\IWP\Programs\Perl>perl -w string.pl
```

-w is specied so that warning messages, if any, are displayed on screen.

- You may put a pair of curly brackets around the identifier name to separate it from the surrounding text, e.g.

```
print "${Num}th Edition";
```

```
$i="HII"; chop $i;
# The chop function is used to "chop off" the last character of a string variable
print $i;# HI
```

Getting Input from User

Program:

```
print "Please enter a Celsius degree:";
$cel=<STDIN>;
$fah = ($cel * 1.8) + 32;
print "The Fahrenheit equivalent of $cel degrees Celsius is $fah";
```

Output:

```
D:\IWP\Programs\Perl>perl celsius.pl
Please enter a Celsius degree: 23
The Fahrenheit equivalent of 23
degrees Celsius is 73.4
```

Getting Input from User

Program:

```
print "Please enter a Celsius degree:";
# Chop off the trailing newline character
chomp($cel = <STDIN>);
$fah = ($cel * 1.8) + 32;
print "The Fahrenheit equivalent of $cel degrees Celsius is $fah";
```

Output:

```
D:\IWP\Programs\Perl>perl celsius.pl
Please enter a Celsius degree: 23
The Fahrenheit equivalent of 23 degrees Celsius is 73.4
```

Exercise

1. Write a program that computes the circumference of a circle with a radius of 12.5. The circumference is 2π times the radius, or about 2 times 3.141592654.
2. Modify the program from the previous exercise to prompt for and accept a radius from the person running the program.
3. Write a program that prompts for and reads two numbers, and then prints out the result of the two numbers multiplied together.
4. Write a program that reads a string and a number, and then prints the string the number of times indicated by the number on separate lines.

substr() - Extraction of Substrings

The syntax of the substr() function is as follows:

- substr STRING, OFFSET
- substr STRING, OFFSET, LENGTH

```
$string = "This is test";
print substr($string, 5); # is test
print substr($string, 5, 2); # is
```

# length of the string using length()

```
print length($string); #12
```

Array

- An array can be created by assigning a list to an array variable. An array variable starts with the symbol @ (compare with the case of $ for scalar variables).

<u>**Eg 1:**</u> @colors = ("red", "orange", "green", "blue");

<u>**Eg 2:**</u> $colors[0] = "red";
$colors[1] = "orange";

<u>**Eg 3:**</u> @nullarray = (); #Empty Array

<u>**Eg 4:**</u>
```
@array = (1, 2, "hello");
$,=" ";
print @array;            # 1  2   hello
$len = @array;
print "\nLength of an array = $len“;  # 3
```

<u>**Eg 5:**</u>
```
$x = 1 ,$y = 2;
@nums = ($x + $y, $x - $y);
```

Extending an array

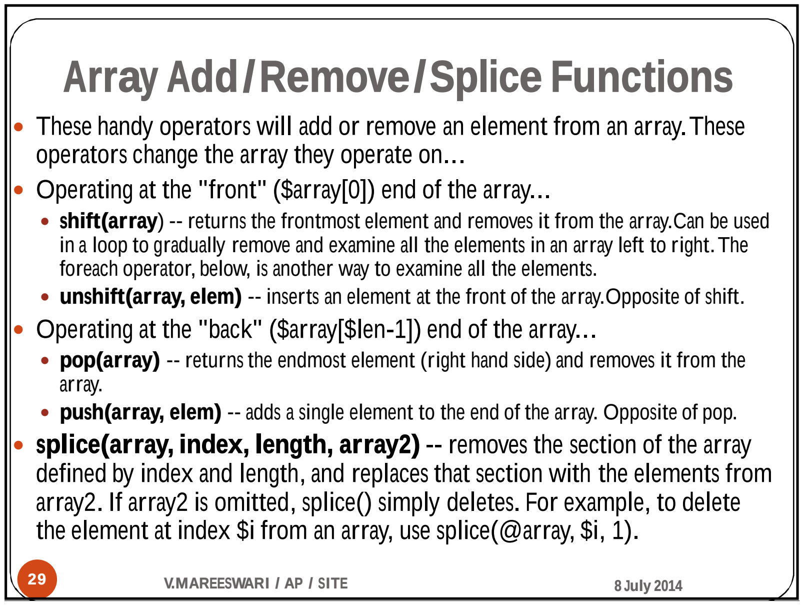
- @unix = ("FreeBSD", "Linux");
- @os = ("MacOS", "Windows NT", "Windows ME”, @unix);

<u>**@os is expanded into**</u>

- @os = ("MacOS", "Windows NT", "Windows ME", "FreeBSD", "Linux");
- print "OS= @os \n";
- @array[3,5]=(“Windows XP”,”Windows2007”); #add items in an array

→ ("MacOS", "Windows NT", "Windows ME", “WindowsXP", "Linux“,”Windows2007”)

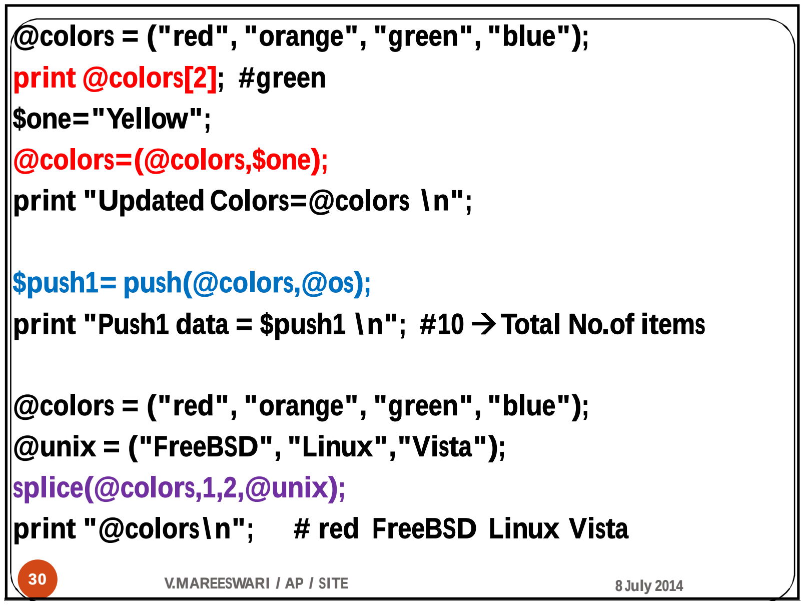
- @hundrednums = (101 .. 200); # **range operator (..)**
- @hundrednums = reverse (101 .. 200); **/ / reverse order**



Array Add/Remove/Splice Functions

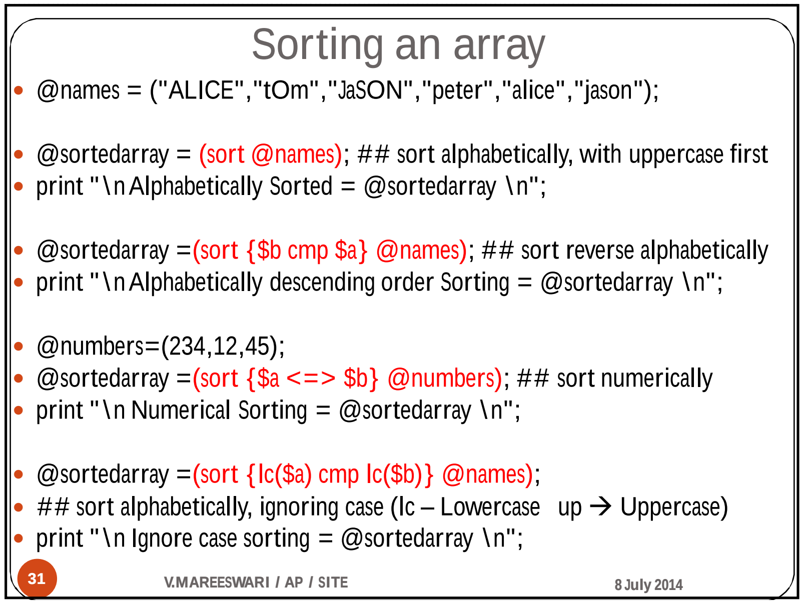
- These handy operators will add or remove an element from an array. These operators change the array they operate on...
- Operating at the "front" ($array[0]) end of the array...
  - **shift(array)** -- returns the frontmost element and removes it from the array. Can be used in a loop to gradually remove and examine all the elements in an array left to right. The foreach operator, below, is another way to examine all the elements.
  - **unshift(array, elem)** -- inserts an element at the front of the array. Opposite of shift.
- Operating at the "back" ($array[$len-1]) end of the array...
  - **pop(array)** -- returns the endmost element (right hand side) and removes it from the array.
  - **push(array, elem)** -- adds a single element to the end of the array. Opposite of pop.
- **splice(array, index, length, array2)** -- removes the section of the array defined by index and length, and replaces that section with the elements from array2. If array2 is omitted, splice() simply deletes. For example, to delete the element at index $i from an array, use splice(@array, $i, 1).

29 V.MAREESWARI / AP / SITE 8 July 2014



```
@colors = ("red", "orange", "green", "blue");
print @colors[2];  #green
$one="Yellow";
@colors=(@colors,$one);
print "Updated Colors=@colors \n";

$push1= push(@colors,@os);
print "Push1 data = $push1 \n";  #10 → Total No.of items

@colors = ("red", "orange", "green", "blue");
@unix = ("FreeBSD", "Linux","Vista");
splice(@colors,1,2,@unix);
print "@colors\n";     # red  FreeBSD  Linux  Vista
```

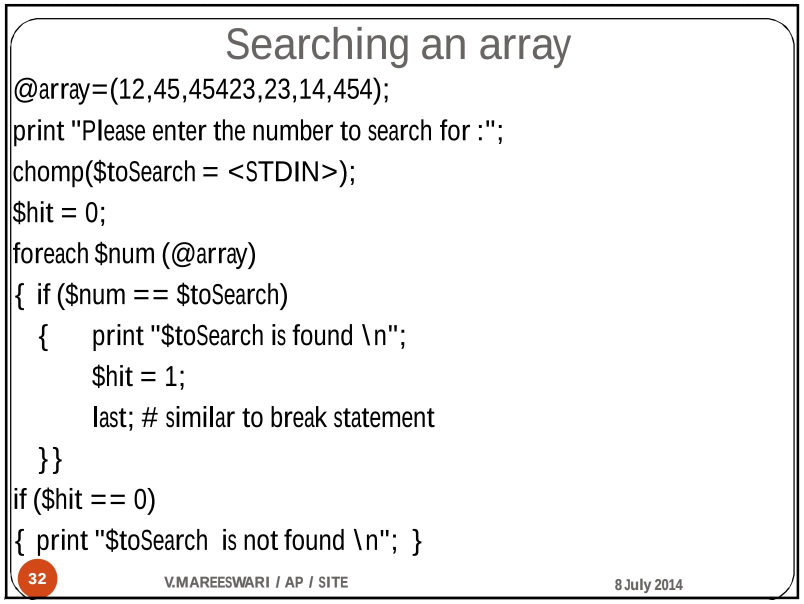
30 V.MAREESWARI / AP / SITE 8 July 2014



Sorting an array

```
@names = ("ALICE","tOm","JaSON","peter","alice","jason");

@sortedarray = (sort @names); ## sort alphabetically, with uppercase first
print "\n Alphabetically Sorted = @sortedarray \n";

@sortedarray =(sort {$b cmp $a} @names); ## sort reverse alphabetically
print "\n Alphabetically descending order Sorting = @sortedarray \n";

@numbers=(234,12,45);
@sortedarray =(sort {$a <=> $b} @numbers); ## sort numerically
print "\n Numerical Sorting = @sortedarray \n";

@sortedarray =(sort {lc($a) cmp lc($b)} @names);
## sort alphabetically, ignoring case (lc – Lowercase   up → Uppercase)
print "\n Ignore case sorting = @sortedarray \n";
```

31 V.MAREESWARI / AP / SITE 8 July 2014



Searching an array

```
@array=(12,45,45423,23,14,454);
print "Please enter the number to search for :";
chomp($toSearch = <STDIN>);
$hit = 0;
foreach $num (@array)
{  if ($num == $toSearch)
   {     print "$toSearch is found \n";
         $hit = 1;
         last; # similar to break statement
   }}
if ($hit == 0)
{  print "$toSearch  is not found \n";  }
```

32 V.MAREESWARI / AP / SITE 8 July 2014

Hashes

- Associative Arrays "Hashes"
  - Associative arrays are created with a set of key/value pairs. A key is a text string of your choice that will help you remember the value later.
  - A hash name begins with % sign.

```
%hashName = ("key1", "value1", "key2",
  "value2");
%ourFriends = ("best", "Don", "good",
  'Robert", "worst", "Joe");
```

  - To access an element, use $+hash name+{+key+}.

```
$hashName{"key1"} #This will return value1
$ourFriends{"good"} #This will return
                     #'Robert'
```

- if(exists($hashName{"key1"})

```
%Age = ("Tom", 26, "Peter", 51, "Jones", 23);
$,=" ";
print  %Age , "\n";  #   Peter 51 Tom 26 Jones 23
%Age = ("Tom" => 26, "Peter" => 51, "Jones" => 23);
#  => is equivalent symbol to the comma
$,=",";
print %Age, "\n";  #   Peter,51,Tom,26,Jones,23,
print "The value of Tom is $Age{Tom} \n";
```

To add a new value to a hash,

Syntax: $hashname{newkey} = newvalue;

Example: $Age{"John"}=33;

```
print %Age, "\n";  #   Peter,51,Tom,26,Jones,23,John,33
```

Hash Operations

- **`keys(%ARRAY)`**
  return a list of all the keys in the %ARRAY
- **`values(%ARRAY)`**
  return a list of all the values in the %ARRAY
- **`each(%ARRAY)`**
  iterates through the ***key-value*** pairs of the %ARRAY
- **`delete($ARRAY{KEY})`**
  removes the key-value pair associated with {KEY} from the ARRAY

```
delete $Age{Tom};
print "The Tom key-value pair is deleted \n";
print %Age, "\n";  #   Peter51Jones23
%Age=();   # Entire hash deleted or assigned by empty
```

Sorting in Hash

```
%array =
(
  3 => "apple",
  11 => "orange",
  5 => "banana",
);
@k = sort { $a <=> $b } keys  %array;
$,=" ";
print  @k;             # 3  5  11
@v = sort { $a cmp $b } values  %array;
$,=" ";
print  @v;             # apple  banana  orange
```

```
if(exists($array{11}))
{        print "EXISTS";
}
```

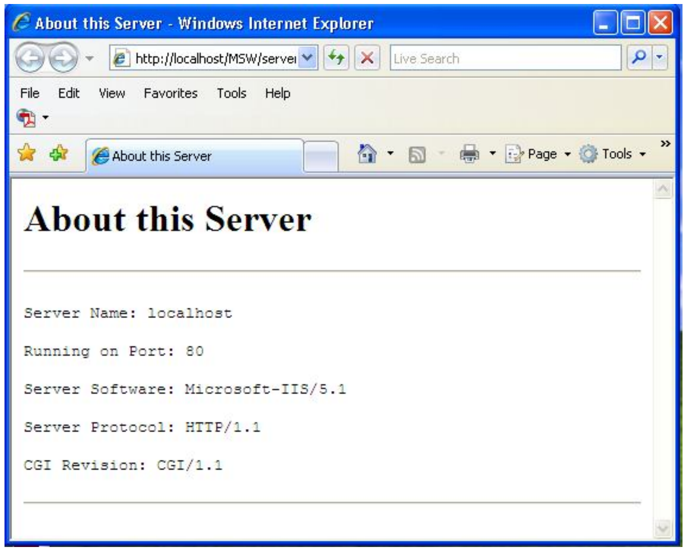
Running Steps in Browser

- Control Panel → Administrative Tools → Internet Information Services
- Local Computer → Default Web Site → Properties
- Home Directory → Configuration → Add →
- Executable = D:\IWP\Material\Perl\bin\perl.exe " %s" %s
- Extension=.pl
- Ok→Apply
- Execute Permissions=Scripts and Executables → ok
- Refresh or Start again IIS
- Perl Program path : c:\Inetpub\wwroot\first.pl
- print "content-type:text/html\n\n";
- print "welcome";
- Run in browser like http://localhost/first.pl

New → Virtual Directory → MSW → Program path → Except write option

Server Information

```
print "Content-type: text/html", "\n\n";
print "<HTML>", "\n";
print "<HEAD><TITLE>About this Server</TITLE></HEAD>", "\n";
print "<BODY><H1>About this Server</H1>", "\n";
print "<HR><PRE>";
print "Server Name: ", $ENV{'SERVER_NAME'}, "<BR>", "\n";
print "Running on Port: ", $ENV{'SERVER_PORT'}, "<BR>", "\n";
print "Server Software: ", $ENV{'SERVER_SOFTWARE'}, "<BR>", "\n";
print "Server Protocol: ", $ENV{'SERVER_PROTOCOL'}, "<BR>", "\n";
print "CGI Revision: ", $ENV{'GATEWAY_INTERFACE'}, "<BR>", "\n";
print "<HR></PRE>", "\n";
print "</BODY></HTML>", "\n";
```


About this Server - Windows Internet Explorer
http://localhost/MSW/server
Live Search
File Edit View Favorites Tools Help
About this Server
Page Tools

About this Server

```
Server Name: localhost
Running on Port: 80
Server Software: Microsoft-IIS/5.1
Server Protocol: HTTP/1.1
CGI Revision: CGI/1.1
```

Perl Operators

Operator	Description
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Modulus operator
+	Positive sign
-	Negative sign
++	Autoincrement operator
--	Autodecrement operator
**	Exponentiation operator

Operator	Description
|| or	Logical OR
&& and	Logical AND
! not	Logical NOT, i.e. negation
xor	Logical XOR — Exclusive OR

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<=>	general comparison

Operator	Description
<<	Binary shift left
>>	Binary shift right
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT

Operator	Description
lt	less than
gt	greater than
le	less than or equal to
ge	greater than or equal to
cmp	general comparison

Operator	Description
==	equal (numeric comparison)
!=	not equal (numeric comparison)
eq	equal (stringwise comparison)
ne	not equal (stringwise comparison)

Operator	Description
=	Assignment operator
+= -= *= /= %= **=	Arithmetic manipulation with assignment
.= x=	String manipulation with assignment
&&= ||=	Logical manipulation with assignment
&= |= ^= <<= >>=	Bitwise manipulation with assignment

41 V.MAREESWARI / AP / SITE 8 July 2014

Control Structures

If / else control structure looks like:

```
    if(condition){
       If body
    }
    else{
       Else body
    }
```

elsif

Example:

```
    if ($gas_money < 10) {
       print "You don't have enough
    money!";
    }
    else {
       print "You have enough money.";
    }
```

42 V.MAREESWARI / AP / SITE 8 July 2014

Control Structures

- For loop has the following C-based formula:

```
for (initialize;condition; increment){
   code to repeat
}
```

For Example:

```
for ($count=1; $count<11; $count++){
   print "cool\n";
}
```

- While loop has also the following C-based formula:

```
while (test condition) {
   code to repeat
}
```

43 V.MAREESWARI / AP / SITE 8 July 2014

Control Structures

While Example:

```
$count=1;
while ($count < 11){
   print "$count\n";   $count++;
}
```

- foreach has the following formula:

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

Foreach Example:

```
foreach $item(@inventory){
   print “$item\n”;
}
```

44 V.MAREESWARI / AP / SITE 8 July 2014

PERL Subroutines

- A subroutine is a named group of statements that does a specific job and can be called over and over. Subroutines are very important for software reuse.
- The formula of creating subroutines in PERL is

```
sub subName(){
   group of statements
}
```

- To call a subroutines as follows:

```
subName(arg1, arg2, arg3, …);
```

- PERL pass the arguments to subroutines in an array named @_

Factorial Calculation

```
sub factorial($n)
{
$f=1;
for($i=1;$i<=$n;$i++)
{
  $f=$f*$i;
}
print "factorial=",$f;
}
$n=3;
factorial($n);
```

Find Maximum among two scalar values

```
sub maximum(@)
{
  if(@_[0] < @_[1])
        return @_[1];
  else
        return @_[0];
}
$n1=34;
$n2=23;
print "Max =", maximum($n1,$n2);
```

Subroutines

```
sub summation(@)
{
  # This subroutine takes a list of numbers as input
  # and returns the sum
  $sum = 0;
  foreach $tmp (@_) #@_[0] , @_[1]
  {
       $sum += $tmp;
  }
  return $sum;
}
# calculates 0 + 1 + 2 + … + 100 and prints the value
print summation 0 .. 100; # must be 5050. No doubt.
```

Regular Expressions

- First, you construct the regular expression, which is essentially a sequence of characters describing the **pattern** you would like to match.
- For eg, in MS-DOS if you would like to list all files with the extension .txt

  dir *.txt
- The "*.txt" here can be described as a pattern as it is the specification used by the operating system
- After constructing the regular expression, you can then bind the data to be searched .
- In this process, you have provided the Perl regexp engine with both the data to search for and the data to be searched.
- The return value indicates if pattern matching is successful. If it is successful, you may want to store the data temporarily, or in a file, or export the results directly to the standard output.

Regular expressions are used in Perl in a number of ways:

- Search for a string that matches a specified pattern, and optionally replacing the pattern found with some other strings
- Counting the number of occurrences of a pattern in a string
- Split a formatted string (e.g. a date like 02/06/2001) into respective components (i.e. into day, month and year)
- Validation of fields received from submitted HTML forms by verifying if a piece of data conforms to a particular format
- … and much more

Three Types of Functions:

1. Matching 2.Subsititution 3.Translate

Building a Pattern

- To search for a pattern match, simply construct the pattern and put it in between the two slashes of the m// operator.

  m/able/
- Now, to see if this pattern occurs in the string "Capable", we bind the twos together by using the binding operator =~

```
if ("Capable" =~ m/able/)
{
        print "match!\n";
}
else
{
        print "no match!\n";
}
```

Output:

```
D:\IWP\Programs\Perl>perl RE_1.pl
match!
```

Examples

for example, to match a Unix path name /var/logs/httpd/error log in an expression you have to escape the forward slashes

```
$expression =~ m/\/var\/logs\/httpd\/error_log/
```

In the manual pages, this is described as the *leaning toothpick syndrome (LTS)* where a lot of forward and backward slashes are present, making the pattern itself difficult to recognize. If you change the symbol to, for example, j, then the entire pattern suddenly becomes clear:

```
$expression =~ m|/var/logs/httpd/error_log|
```

This is just one of the methods to remove the leaning toothpick syndrome.

Basic matching and substitution

- **Assign a value (string literal) to the variable.**
  $mystring = "Hello world!";
- **Does the string contains the word "World"?**
  if($mystring =~ m/World/) { print "Yes"; }
- **Does the string contains the word "World", ignoring case?**
  if($mystring =~ m/World/i) { print "Yes"; }
- **I want "Hello world!" to be changed to "Hello mom!" instead.**
  $mystring =~ s/world/mom/; print $mystring;
  Prints "Hello mom!". The substitution operator s/// replaces the pattern between the s/ and the middle /, with the pattern between the middle / and last /. In this case, "world" is replaced with the word "mom".
- **Okay, ignoring case, change "Hello mom!" to say "Goodby mom!".**
  $mystring =~ s/hello/Goodbye/i; print $mystring;
  Prints "Goodby mom!".

Examples

- The following two expressions are equivalent:
  !($expression =~ m/pattern/)
  $expression !~ m/pattern/
- you can generate patterns at runtime and apply them by, for example,
  $expression =~ m/$var/

- m/for|if|while/ # A match if either 'for', 'if' or 'while' found
- m/a(a|b|c)a/ # A match if either 'aaa', 'aba' or 'aca' found

Extracting substrings

- **I want to see if my string contains a digit.**
  $mystring = “On 2nd April ";
  if($mystring =~ m/\d/) { print "Yes"; }
  Prints "Yes". The pattern \d matches any single digit. In this case, the search will finish as soon as it reads the "2". Searching always goes left to right.
- **Huh? Why doesn't "\d" match the exact characters '\' and 'd'?**
  This is because Perl uses characters from the alphabet to also match things with special meaning, like digits. To differentiate between matching a regular character and something else, the character is immediately preceded by a backslash. Therefore, whenever you read '\' followed by any character, you treat the two together as one symbol. For example, '\d' means digit, '\w' means alphanumeric characters including '_', '\/' means forward slash, and '\\' means match a single backslash. Preceding a character with a '\' is called escaping, and the '\' together with its character is called an escape sequence.

- **Okay, how do I return the first matching digit from my string?**
  $mystring = “On 25th April 1989";
  if($mystring =~ m/**(\d)**/)
  { print "The first digit is **$1**."; }
  Prints "The first digit is 2." In order to designate a pattern for extraction, one places parenthesis around the pattern. If the pattern is matched, it is returned in the Perl special variable called $1. If there are multiple parenthesized expressions, then they will be in variables $1, $2, $3, etc.

```
$mystring="1234567890 hai 345 welcome 6789";
@array=($mystring=~m/(\d+)/g);
print "@array \n";
```

- This introduces another pattern modifier g, which tells Perl to do a global search on the string. In other words, search the whole string from left to write.
- **Output:**

```
1234567890 345 6789
```

\d means single digit in a number
\d+ means all digits in a number
g means all numbers in a string

Common tasks

- **How do I extract everything between a the words "start" and "end"?**

```
$mystring = "The start text always precedes the end of the end
   text.The start text always precedes the end of the end text.";
if($mystring =~ m/start(.*)end/) {
   print $1;
}
```

The pattern .* is two different metacharacters that tell Perl to match everything between the start and end. Specifically, the metacharacter means match any symbol except new line. The pattern quantifier * means match *zero or more* of the preceding symbol.

That isn't exactly what I expected. How do I extract everything between "start" and the first "end" encountered?

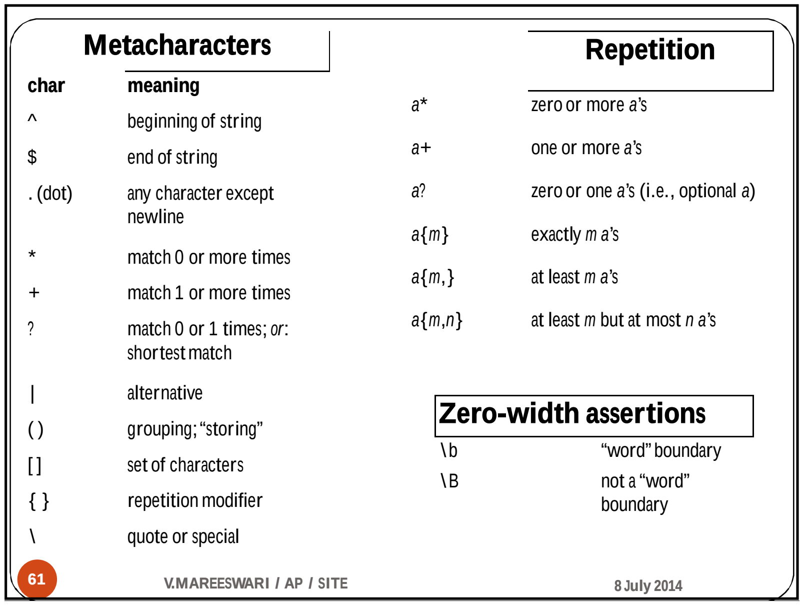
```
$mystring = "The start text always precedes the end of the end text.
   The start text always precedes the end of the end text.";
if($mystring =~ m/start(.*?)end/) {
   print $1;
}
```

This means that when you say .*, Perl matches every character (except new line) all the way to the end of the string, and then works backward until it finds end. To make the pattern quantifier miserly, you use the pattern quantifier limiter ?. This tells Perl to match as *few* as possible of the preceding symbol before continuing to the next part of the pattern.

Translate

- Translations are like substitutions, except they happen on a letter by letter basis instead of substituting a single phrase for another single phrase.

```
$mystring="HAI welcome to VIT University";
$mystring=~tr /[a-z]/[A-Z]/;
print $mystring;
```



Metacharacters

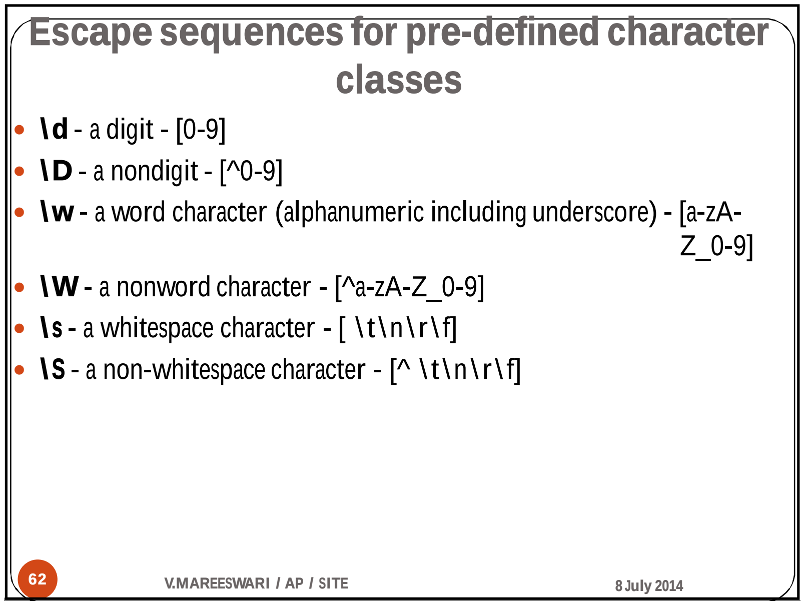
char	meaning
^	beginning of string
$	end of string
. (dot)	any character except newline
*	match 0 or more times
+	match 1 or more times
?	match 0 or 1 times; *or*: shortest match
|	alternative
( )	grouping; "storing"
[ ]	set of characters
{ }	repetition modifier
\	quote or special

Repetition

*a**	zero or more *a*'s
a+	one or more *a*'s
a?	zero or one *a*'s (i.e., optional *a*)
a{*m*}	exactly *m a*'s
a{*m*,}	at least *m a*'s
a{*m*,*n*}	at least *m* but at most *n a*'s

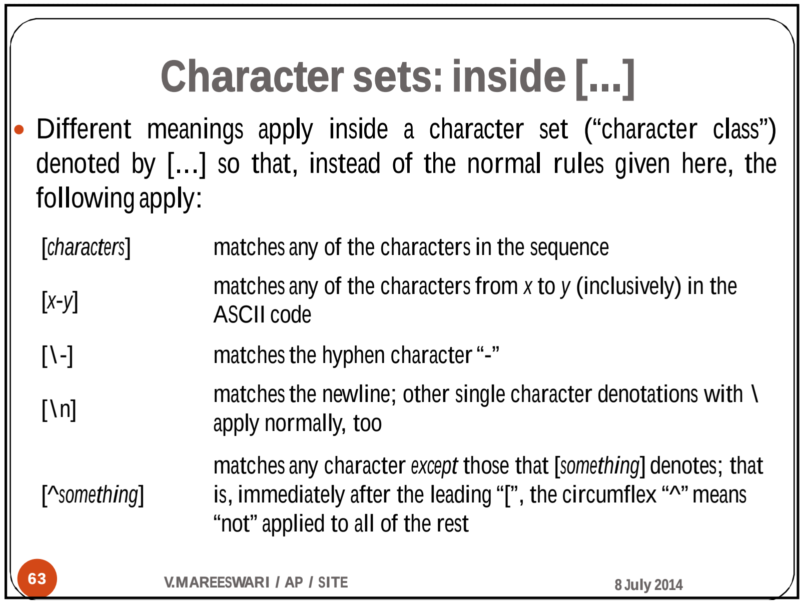
Zero-width assertions

\b	"word" boundary
\B	not a "word" boundary



Escape sequences for pre-defined character classes

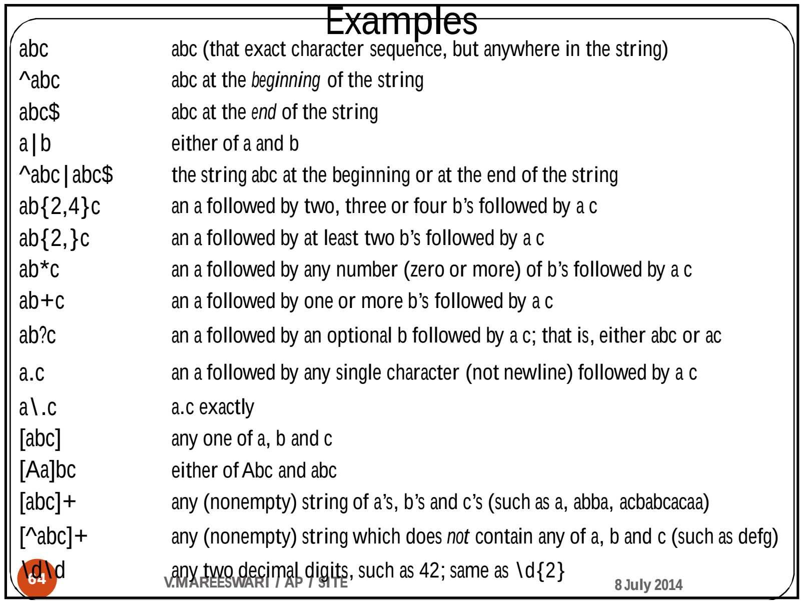
- **\d** - a digit - [0-9]
- **\D** - a nondigit - [^0-9]
- **\w** - a word character (alphanumeric including underscore) - [a-zA-Z_0-9]
- **\W** - a nonword character - [^a-zA-Z_0-9]
- **\s** - a whitespace character - [\t\n\r\f]
- **\S** - a non-whitespace character - [^ \t\n\r\f]



Character sets: inside [...]

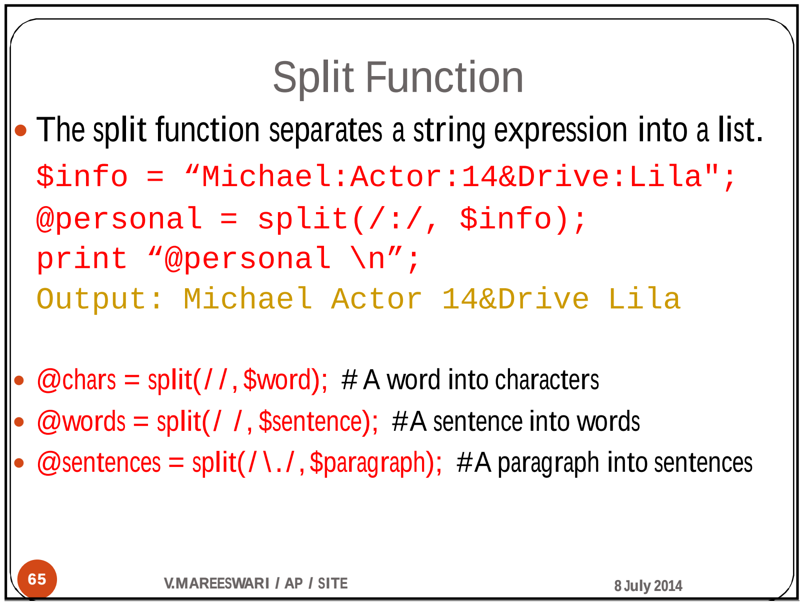
- Different meanings apply inside a character set ("character class") denoted by [...] so that, instead of the normal rules given here, the following apply:

[*characters*]	matches any of the characters in the sequence
[*x*-*y*]	matches any of the characters from *x* to *y* (inclusively) in the ASCII code
[\-]	matches the hyphen character "-"
[\n]	matches the newline; other single character denotations with \ apply normally, too
[^*something*]	matches any character *except* those that [*something*] denotes; that is, immediately after the leading "[", the circumflex "^" means "not" applied to all of the rest



Examples

abc	abc (that exact character sequence, but anywhere in the string)
^abc	abc at the *beginning* of the string
abc$	abc at the *end* of the string
a|b	either of a and b
^abc|abc$	the string abc at the beginning or at the end of the string
ab{2,4}c	an a followed by two, three or four b's followed by a c
ab{2,}c	an a followed by at least two b's followed by a c
ab*c	an a followed by any number (zero or more) of b's followed by a c
ab+c	an a followed by one or more b's followed by a c
ab?c	an a followed by an optional b followed by a c; that is, either abc or ac
a.c	an a followed by any single character (not newline) followed by a c
a\.c	a.c exactly
[abc]	any one of a, b and c
[Aa]bc	either of Abc and abc
[abc]+	any (nonempty) string of a's, b's and c's (such as a, abba, acbabcacaa)
[^abc]+	any (nonempty) string which does *not* contain any of a, b and c (such as defg)
\d\d	any two decimal digits, such as 42; same as \d{2}



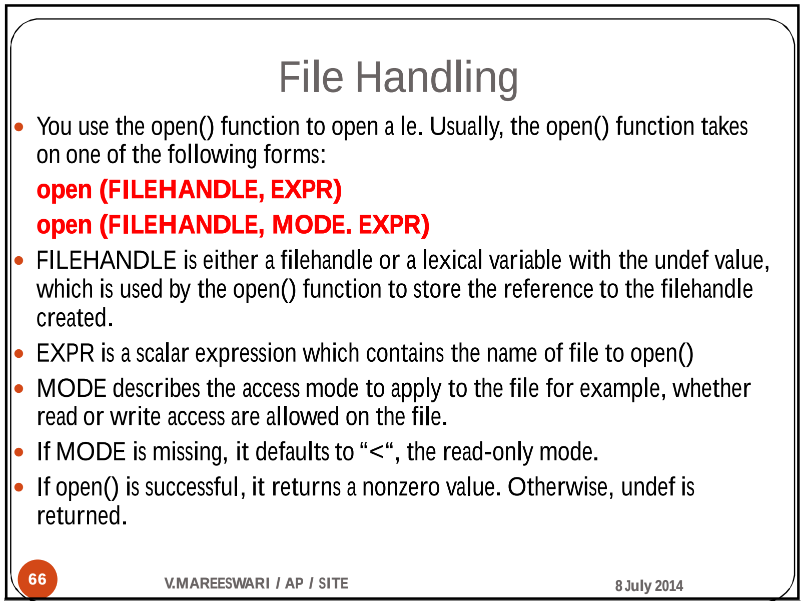
Split Function

- The split function separates a string expression into a list.

```
$info = “Michael:Actor:14&Drive:Lila";
@personal = split(/:/, $info);
print “@personal \n”;
Output: Michael Actor 14&Drive Lila
```

- @chars = split(/ /, $word); # A word into characters
- @words = split(/ /, $sentence); #A sentence into words
- @sentences = split(/\./, $paragraph); #A paragraph into sentences

65 V.MAREESWARI / AP / SITE 8 July 2014



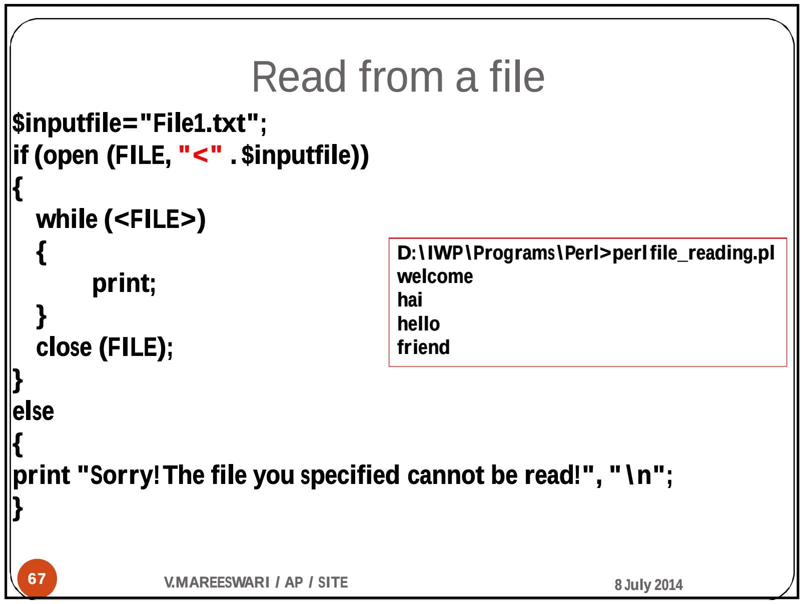
File Handling

- You use the open() function to open a le. Usually, the open() function takes on one of the following forms:

  open (FILEHANDLE, EXPR)

  open (FILEHANDLE, MODE. EXPR)
- FILEHANDLE is either a filehandle or a lexical variable with the undef value, which is used by the open() function to store the reference to the filehandle created.
- EXPR is a scalar expression which contains the name of file to open()
- MODE describes the access mode to apply to the file for example, whether read or write access are allowed on the file.
- If MODE is missing, it defaults to “<“, the read-only mode.
- If open() is successful, it returns a nonzero value. Otherwise, undef is returned.

66 V.MAREESWARI / AP / SITE 8 July 2014

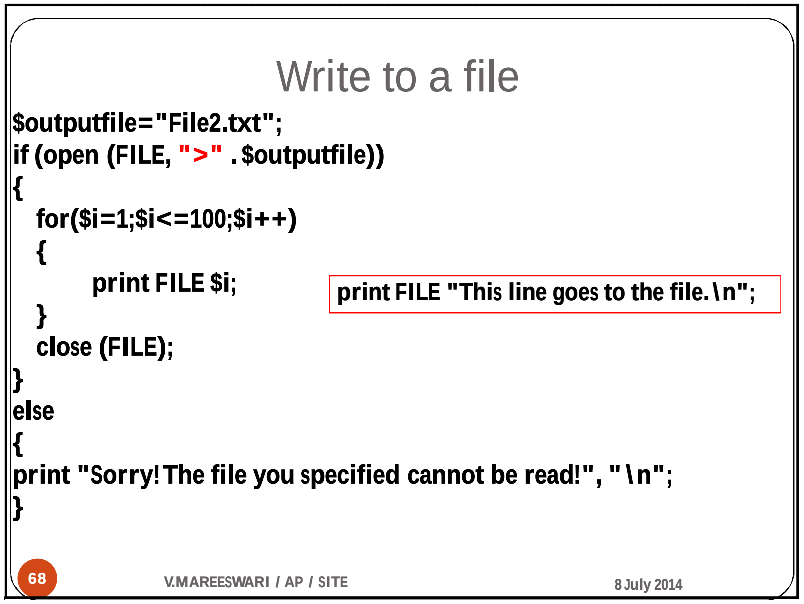


Read from a file

```
$inputfile="File1.txt";
if (open (FILE, "<" . $inputfile))
{
   while (<FILE>)
   {
         print;
   }
   close (FILE);
}
else
{
print "Sorry! The file you specified cannot be read!", "\n";
}
```

```
D:\IWP\Programs\Perl>perl file_reading.pl
welcome
hai
hello
friend
```

67 V.MAREESWARI / AP / SITE 8 July 2014



Write to a file

```
$outputfile="File2.txt";
if (open (FILE, ">" . $outputfile))
{
   for($i=1;$i<=100;$i++)
   {
         print FILE $i;
   }
   close (FILE);
}
else
{
print "Sorry! The file you specified cannot be read!", "\n";
}
```

```
print FILE "This line goes to the file.\n";
```

68 V.MAREESWARI / AP / SITE 8 July 2014

Read & Write

```
$inputfile="File1.txt";
$outputfile="File2.txt";
if (open (FILE1, "<". $inputfile) && open (FILE2, ">" . $outputfile))
{
   while (<FILE1>)
   {
          print  FILE2  $_;
   }
   close (FILE1);
   close (FILE2);
}
else
{
print "Sorry! The file you specified cannot be read!", "\n";
}
```

Append

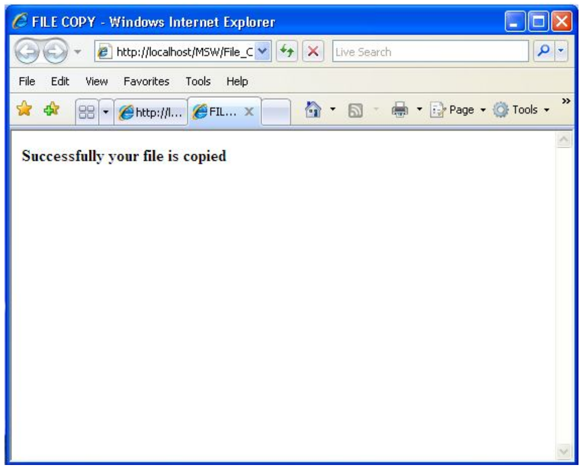
```
$inputfile="File1.txt";
$toappendfile="File2.txt";
if (open (FILE1, ">>". $inputfile) && open (FILE2, "<". $toappendfile))
{
   while (<FILE2>)
   {
          print FILE1  $_;
   }
   close (FILE1);
   close (FILE1);
}
else
{
print "Sorry! The file you specified cannot be read!", "\n";
}
```

Copy , Move and Delete

- We can duplicate a file using the *copy* function. Copy takes two values the URL of the file to be copied and the URL of the new file.
- copy($filetobecopied, $newfile);
- move($oldlocation, $newlocation); # 'cutting' the file and sending it to a new location
- unlink($file); # Use the *unlink* function to delete specific files from your web server.

Copy

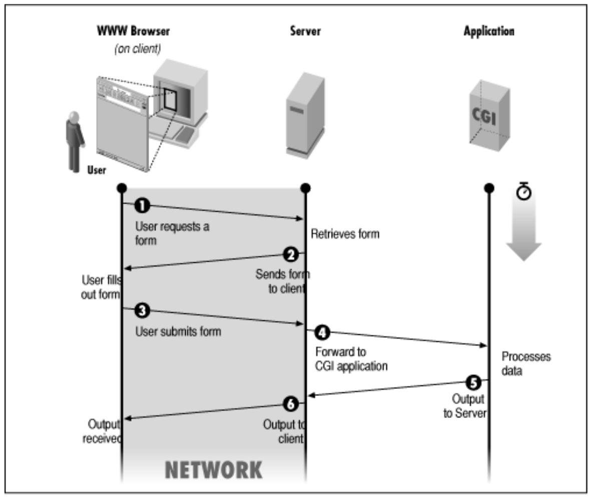
```
use File::Copy;
print "content-type: text/html \n\n"; #The header
$filetobecopied = "File1.txt";
$newfile = "File4.txt";
if(copy($filetobecopied, $newfile))
{
print "<html><head><title> FILE COPY </title></head>";
print "<body><b>Successfully your file is
   copied</b></body></html>\n\n";
}
else
{
print "<html><body><b>File copy Error</b></body></html>";
}
```


FILE COPY - Windows Internet Explorer
http://localhost/MSW/File_C
Live Search
File Edit View Favorites Tools Help
http://l...
FIL...
Page
Tools

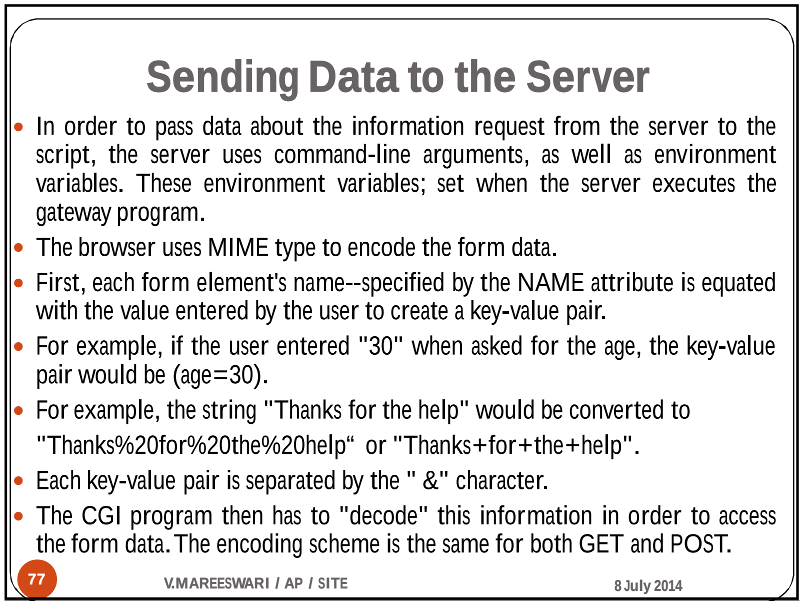
Successfully your file is copied

- **seek() → Set File Pointer Position**
  - seek(FILE, 0, SEEK_SET); # Jump to beginning of file
  - seek(FILE, 5, SEEK_CUR); # Jump 5 bytes forward
  - seek(FILE, -1, SEEK_END); # Jump to last byte of file
- **tell() → Return File Pointer Position**
- **opendir() → Open A Directory**
  - opendir DIRHANDLE, PATH
- **readdir() → Read Directory Content**
  - readdir DIRHANDLE
- **closedir() → Close A Directory**

Form interaction with CGI



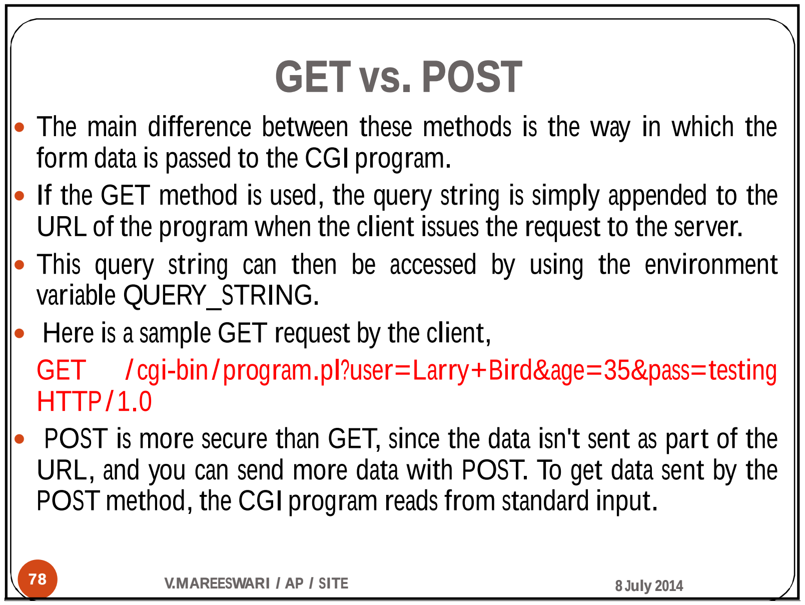
Form Tag	Description
<FORM ACTION="/cgi-bin/prog.pl" METHOD="POST">	Form Start
<INPUT TYPE="text" NAME="name" VALUE="value" SIZE="size">	Text Field
<INPUT TYPE="password" NAME="name" VALUE="value" SIZE="size">	Password Field
<INPUT TYPE="hidden" NAME="name" VALUE="value">	Hidden Field
<INPUT TYPE="checkbox" NAME="name" VALUE="value">	Checkbox
<INPUT TYPE="radio" NAME="name" VALUE="value">	Radio Button
<SELECT NAME="name" SIZE=1> <OPTION SELECTED> One <OPTION>Two … </SELECT>	Dropdown List
<INPUT TYPE="submit" VALUE="Message!" >	Submit Button
<INPUT TYPE="reset" VALUE="Message!">	Reset Button
</FORM>	Form Ends



Sending Data to the Server

- In order to pass data about the information request from the server to the script, the server uses command-line arguments, as well as environment variables. These environment variables; set when the server executes the gateway program.
- The browser uses MIME type to encode the form data.
- First, each form element's name--specified by the NAME attribute is equated with the value entered by the user to create a key-value pair.
- For example, if the user entered "30" when asked for the age, the key-value pair would be (age=30).
- For example, the string "Thanks for the help" would be converted to "Thanks%20for%20the%20help“ or "Thanks+for+the+help".
- Each key-value pair is separated by the " &" character.
- The CGI program then has to "decode" this information in order to access the form data. The encoding scheme is the same for both GET and POST.

77 V.MAREESWARI / AP / SITE 8 July 2014

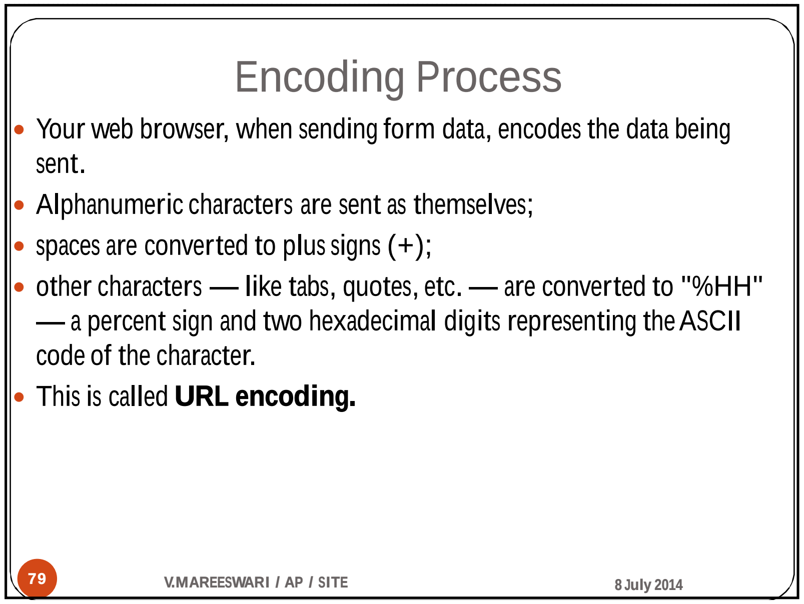


GET vs. POST

- The main difference between these methods is the way in which the form data is passed to the CGI program.
- If the GET method is used, the query string is simply appended to the URL of the program when the client issues the request to the server.
- This query string can then be accessed by using the environment variable QUERY_STRING.
- Here is a sample GET request by the client,

  GET /cgi-bin/program.pl?user=Larry+Bird&age=35&pass=testing HTTP/1.0
- POST is more secure than GET, since the data isn't sent as part of the URL, and you can send more data with POST. To get data sent by the POST method, the CGI program reads from standard input.

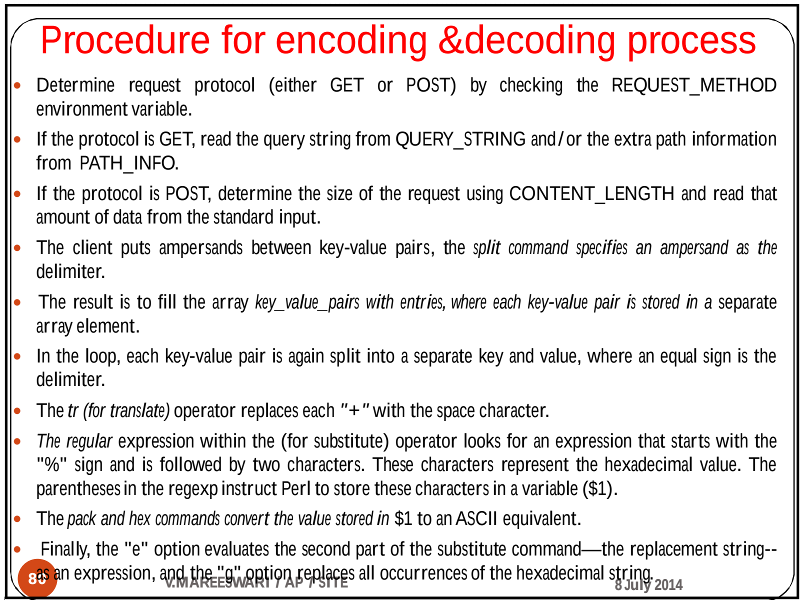
78 V.MAREESWARI / AP / SITE 8 July 2014



Encoding Process

- Your web browser, when sending form data, encodes the data being sent.
- Alphanumeric characters are sent as themselves;
- spaces are converted to plus signs (+);
- other characters — like tabs, quotes, etc. — are converted to "%HH" — a percent sign and two hexadecimal digits representing the ASCII code of the character.
- This is called **URL encoding.**

79 V.MAREESWARI / AP / SITE 8 July 2014



Procedure for encoding &decoding process

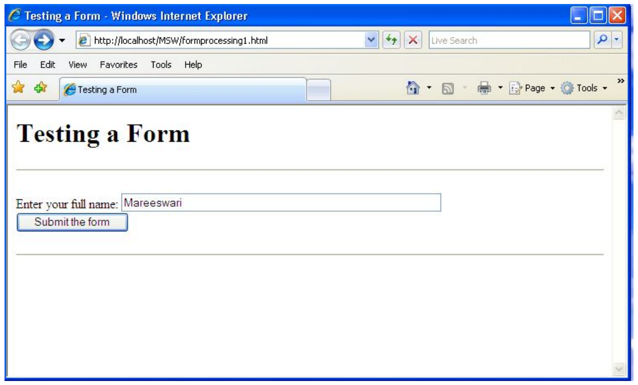
- Determine request protocol (either GET or POST) by checking the REQUEST_METHOD environment variable.
- If the protocol is GET, read the query string from QUERY_STRING and/or the extra path information from PATH_INFO.
- If the protocol is POST, determine the size of the request using CONTENT_LENGTH and read that amount of data from the standard input.
- The client puts ampersands between key-value pairs, the *split command specifies an ampersand as the* delimiter.
- The result is to fill the array *key_value_pairs with entries, where each key-value pair is stored in a* separate array element.
- In the loop, each key-value pair is again split into a separate key and value, where an equal sign is the delimiter.
- The *tr (for translate)* operator replaces each "+" with the space character.
- *The regular* expression within the (for substitute) operator looks for an expression that starts with the "%" sign and is followed by two characters. These characters represent the hexadecimal value. The parentheses in the regexp instruct Perl to store these characters in a variable ($1).
- The *pack and hex commands convert the value stored in* $1 to an ASCII equivalent.
- Finally, the "e" option evaluates the second part of the substitute command—the replacement string--as an expression, and the "g" option replaces all occurrences of the hexadecimal string.

80 V.MAREESWARI / AP / SITE 8 July 2014

formprocessing1.html

```
<HTML>
<HEAD><TITLE>Testing a Form</TITLE></HEAD>
<BODY>
<H1>Testing a Form</H1>
<HR>
<FORM ACTION="formprocessing1_decode.pl" METHOD="POST">
Enter your full name: <INPUT TYPE="text" NAME="user" SIZE=60><BR>
<INPUT TYPE="submit" VALUE="Submit the form">
</FORM>
<HR>
</BODY>
</HTML>
```

81 V.MAREESWARI / AP / SITE 8 July 2014


Testing a Form - Windows Internet Explorer
http://localhost/MSW/formprocessing1.html
File Edit View Favorites Tools Help
Testing a Form

Testing a Form

Enter your full name: Mareeswari

Submit the form

82 V.MAREESWARI / AP / SITE 8 July 2014

Formprocessing1_decode.pl

```
local(@ key_value_pairs,$request_method,$query_string,$key_value,$key,
   $value, %FORM);
$request_method = $ENV{'REQUEST_METHOD'};
   if ($request_method eq "GET") {
      $query_string = $ENV{'QUERY_STRING'};
   } elsif ($request_method eq "POST") {
      read (STDIN, $query_string,$ENV{'CONTENT_LENGTH'});
   }
@key_value_pairs = split (/&/, $query_string);
foreach $key_value (@key_value_pairs)
   {  ($key, $value) = split (/=/, $key_value);
      $value =~ tr/+/ /;
      $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack ("C", hex ($1))/eg;
      $FORM{$key} = $value;    }
```

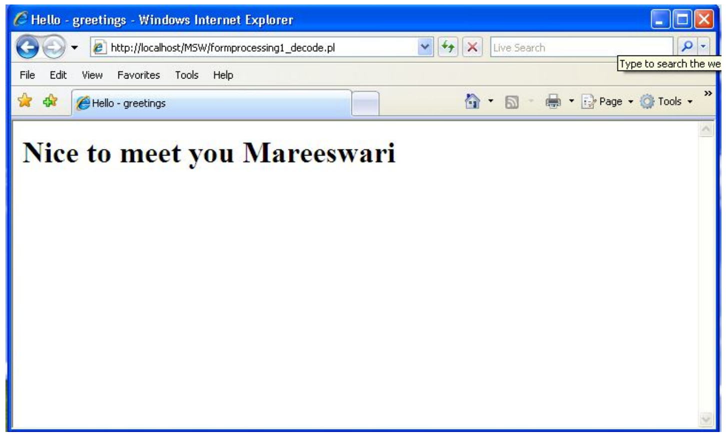
83 V.MAREESWARI / AP / SITE 8 July 2014

Cont...

```
print "Content-type:text/html","\n\n";
print "<html>";
print "<head>";
print "<title>Hello - greetings</title>";
print "</head>";
print "<body>";
print "<h1>Nice to meet you ", $FORM{"user"},"</h1> ";
print "</body>";
print "</html>";
```

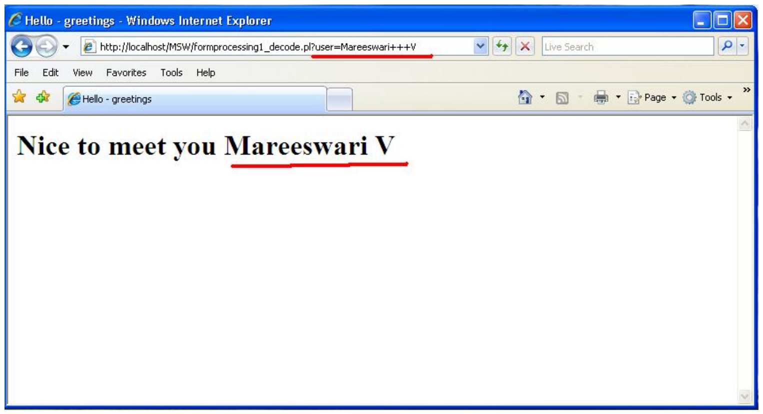
84 V.MAREESWARI / AP / SITE 8 July 2014

Post Method


Hello - greetings - Windows Internet Explorer
http://localhost/MSW/formprocessing1_decode.pl
File Edit View Favorites Tools Help
Hello - greetings

Nice to meet you Mareeswari

Get Method

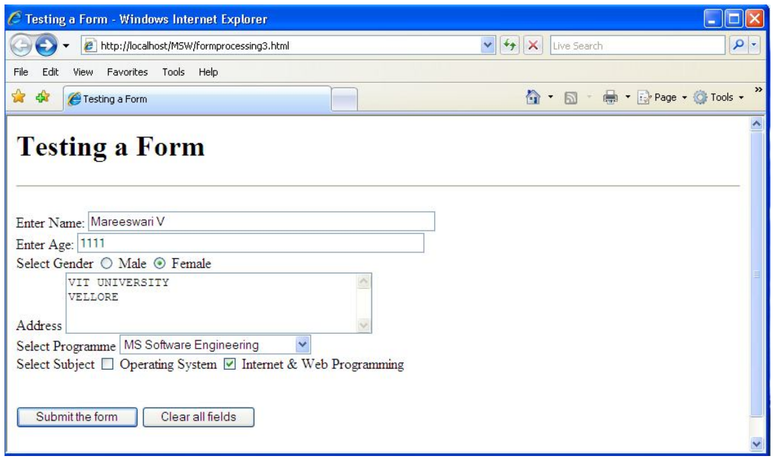

Hello - greetings - Windows Internet Explorer
http://localhost/MSW/formprocessing1_decode.pl?user=Mareeswari+++V
File Edit View Favorites Tools Help
Hello - greetings

Nice to meet you Mareeswari V

```
<HTML><HEAD><TITLE>Testing a Form</TITLE></HEAD>
<BODY><H1>Testing a Form</H1><HR>
<FORM ACTION="formprocessing3_decode.pl" METHOD="GET">
Enter Name: <INPUT TYPE="text" NAME="name" SIZE=60><BR>
Enter Age: <INPUT TYPE="text" NAME="age" SIZE=60><BR>
Select Gender <input type="radio" name="gender" value="male"> Male
<input type="radio" name="gender" value="female"> Female<br>
Address<textarea name="address" cols=40 rows=4>Type your
   Address</textarea><br>
Select Programme<select name="programme“><option value="B.Tech"
   selected>Bachelor Technology</option>
<option value="Ms(SE)">MS Software Engineering</option>
<option value="MCA">Master of Computer Application</option></select><br>
Select Subject<input type="checkbox" name="subject" value="os"> Operating
   System
<input type="checkbox" name="subject" value="iwp"> Internet & Web
   Programming<br><br><br>
<INPUT TYPE="submit" VALUE="Submit the form">
<INPUT TYPE="reset" VALUE="Clear all fields">
</FORM><HR></BODY></HTML>
```

SAME DECODE CODE

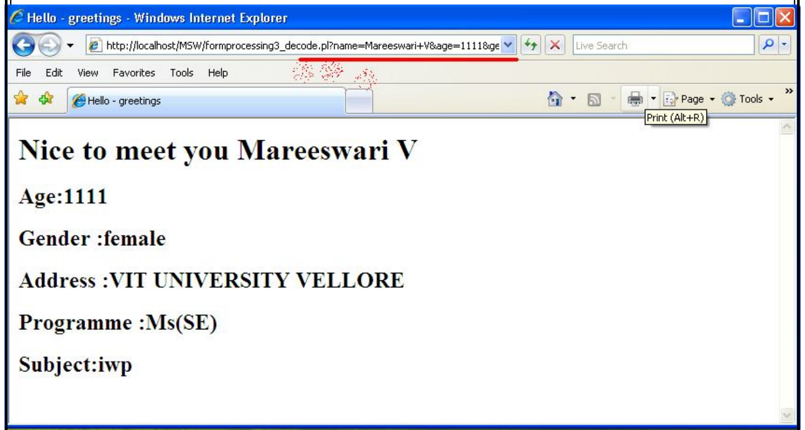
```
print "Content-type:text/html","\n\n";
print "<html>";
print "<head>";
print "<title>Hello - greetings</title>";
print "</head>";
print "<body>";
print "<h1> Nice to meet you ", $FORM{'name'},"</h1>";
print "<h2> Age:",$FORM{'age'},"</h2> ";
print "<h2> Gender :", $FORM{'gender'},"</h2>";
print "<h2> Address :",$FORM{'address'},"</h2>";
print "<h2> Programme :",$FORM{'programme'},"</h2>";
print "<h2> Subject:",$FORM{'subject'},"</h2>";
print "</body>";
print "</html>";
```


Testing a Form - Windows Internet Explorer

http://localhost/MSW/formprocessing3.html

Testing a Form

Enter Name: Mareeswari V
Enter Age: 1111
Select Gender ( ) Male [✓ Female]
Address: VIT UNIVERSITY VELLORE
Select Programme: MS Software Engineering
Select Subject ☐ Operating System ☑ Internet & Web Programming


Hello - greetings - Windows Internet Explorer

http://localhost/MSW/formprocessing3_decode.pl?name=Mareeswari+V&age=1111&ge

Nice to meet you Mareeswari V

Age:1111

Gender :female

Address :VIT UNIVERSITY VELLORE

Programme :Ms(SE)

Subject:iwp

Using CGI Module

```
use CGI  qw(:standard);
use CGI::Carp  qw(warningsToBrowser  fatalsToBrowser);
use strict;
print header;
print start_html("Home Page");
my %form;
foreach my $i (param()) {
   $form{$i} = param($i);               #$form{'age'};
  print "$i = $form{$i}<br>\n";
}
print end_html;
```

Run → http://localhost/first.html

C:\windows\temp\first.html

```
<form action="second.pl">
<input type="text"
   name="name">
<input type="text" name="age">
<input type="submit"
   value="Click Here">
</form>
```

C:\windows\temp\second.pl

```
use CGI  qw(:standard);
print "content-
   type:text/html\n\n";
print "<html><body>";
$n1=param("name");
$n2=param("age");
print "<h1>Welcome",$n1;
print "<h2>Your age",$n2;
print "</body></html>";
```

Exercise

1. Write a form that provides two input fields that are added together when the user submits it.

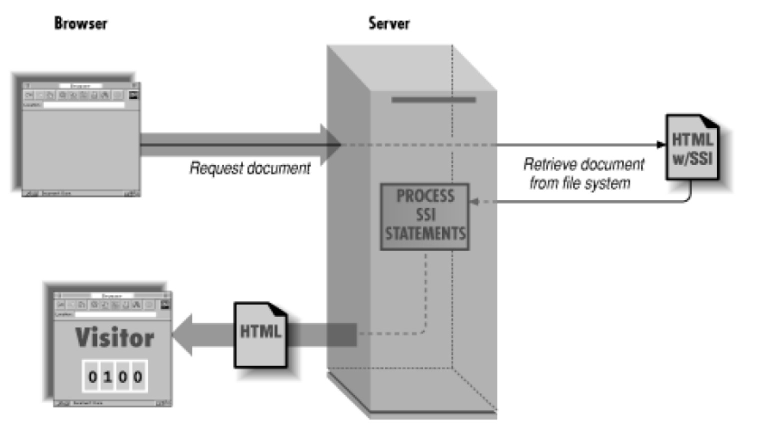
SSI - **Server Side Includes**

- SSI is a simple interpreted server-side scripting language used almost exclusively for the Web. The most frequent use of SSI is to include the contents of one or more files into a web page on a web server.
- For example, a web page containing a daily quotation could include the quotation by placing the following code into the file of the web page:

  <!--#include virtual="../quote.txt" -->
- With one change of the quote.txt file, all pages including the file will display the latest daily quotation. The inclusion is not limited to files, and may also be the text output from a program, or the value of a system variable such as the current time.
- Server Side Includes are useful for including a common piece of code throughout a site, such as a page header, a page footer and a navigation menu. Conditional navigation menus can be conditionally included using control directives.

How do Server Side Includes work?

When the client requests a document from the SSI-enabled server, the server parses the specified document and returns the evaluated document.



Disadvantages

- First, it can be quite costly for a server to continually parse documents before sending them to the client.
- And second, enabling SSI creates a security risk. Novice users could possibly embed directives to execute system commands that output confidential information.
- Despite these shortcomings, SSI can be a very powerful tool if used cautiously.

Basics

- In order for a web server to recognize an SSI-enabled HTML file and therefore carry out these instructions, either the filename should end with a special extension, by default .shtml,.stm, .shtm, or, if the server is configured to allow this, set the execution bit of the file.
- As a simple programming language, SSI supports only one type: text. Its control flow is rather simple, choice is supported, but loops are not natively supported and can only be done by recursion using include or using HTTP redirect.
- Apache, nginx, lighttpd and IIS are the four major web servers that support this language.
- However, there is a CGI program called *fakessi.pl* that you can use to emulate Server Side Includes if your server does not support them.
- **Syntax:** <!--#directive parameter=value parameter=value -->

  Directives are placed in HTML comments so that if SSI is not enabled, users will not see the SSI directives on the page, unless they look at its source. Note that the syntax does not allow spaces between the leading "<" and the directive.

SSI Directives

Command	Parameter	Description
echo	*var*	Inserts value of special SSI variables as well as other environment variable
include		Inserts text of document into current file
	file	Pathname relative to current directory
	virtual	Virtual path to a document on the server
fsize	*file*	Inserts the size of a specified file
flastmod	*file*	Inserts the last modification date and time for a specified file
exec		Executes external programs and inserts output in current document
	cmd	Any application on the host
	cgi	CGI program
config		Modifies various aspects of SSI
	errmsg	Default error message
	sizefmt	Format for size of the file
	timefmt	Format for dates

Additional Environment Variable

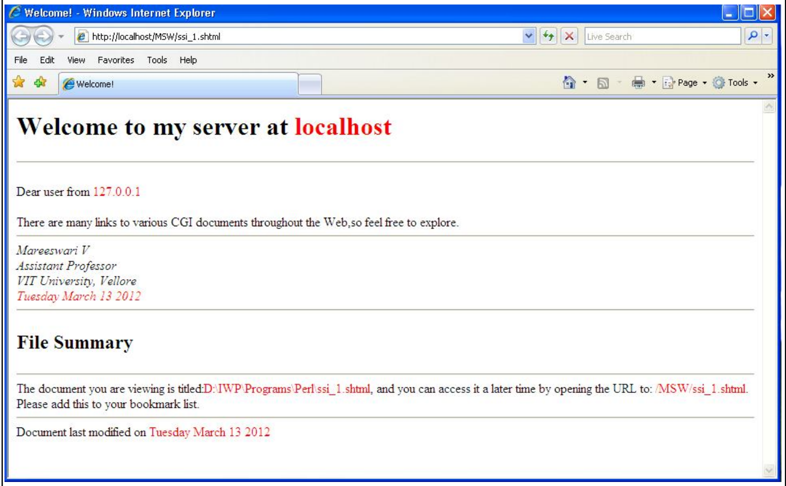
Environment Variable	Description
DOCUMENT_NAME	The current file
DOCUMENT_URI	Virtual path to the file
QUERY_STRING_UNESCAPED	Undecoded query string with all shell metacharacters escaped with "\"
DATE_LOCAL	Current date and time in the local time zone
DATE_GMT	Current date and time in GMT
LAST_MODIFIED	Last modification date and time for current file

The *echo* SSI command with the *var* parameter is used to display the IP name or address of the serving machine, the remote host name, and the local time.

```
<HTML> <HEAD><TITLE>Welcome!</TITLE></HEAD> <BODY>
<H1>Welcome to my server at <span style="color:red">
<!--#echo var="SERVER_NAME"--></span></H1>
<HR>
<P> Dear user from <span style="color:red">
<!--#echo var="REMOTE_HOST"--></span>
<P>There are many links to various CGI documents throughout the Web,so feel
   free to explore.
<HR>
<ADDRESS>Mareeswari V<br> Assistant Professor <br> VIT University,
   Vellore <br><span style="color:red">
 <!--#echo var="DATE_LOCAL"--></span></ADDRESS>
```

```
<HR> <H2>File Summary</H2> <HR>
The document you are viewing is titled:<span style="color:red">
<!--#echo var="DOCUMENT_NAME"--></span>,
and you can access it a later time by opening the URL to:
<span style="color:red"><!--#echo var="DOCUMENT_URI"-->
   </span>. Please add this to your bookmark list.
<HR>Document last modified on <span style="color:red">
<!--#echo var="LAST_MODIFIED"--></span>
</BODY></HTML>
```

This will display the name, URL (although the variable is titled DOCUMENT_URI), and modification time for the current HTML document.


Welcome! - Windows Internet Explorer
http://localhost/MSW/ssi_1.shtml
Live Search
File Edit View Favorites Tools Help
Welcome!
Page Tools

Welcome to my server at localhost

Dear user from 127.0.0.1

There are many links to various CGI documents throughout the Web,so feel free to explore.

Mareeswari V
Assistant Professor
VIT University, Vellore
Tuesday March 13 2012

File Summary

The document you are viewing is titled:D:\IWP\Programs\Perl\ssi_1.shtml, and you can access it a later time by opening the URL to: /MSW/ssi_1.shtml. Please add this to your bookmark list.

Document last modified on Tuesday March 13 2012

Insert a file into your various HTML documents with the SSI *include* command.

address.html (shtml)

```
<HR>
<ADDRESS>
<PRE>
V Mareeswari    Assistant Professor
SITE            VIT University
Vellore         632014
The address information was last
   modified,
<!--#echo var="LAST_MODIFIED"-->.
</PRE>
</ADDRESS>
```

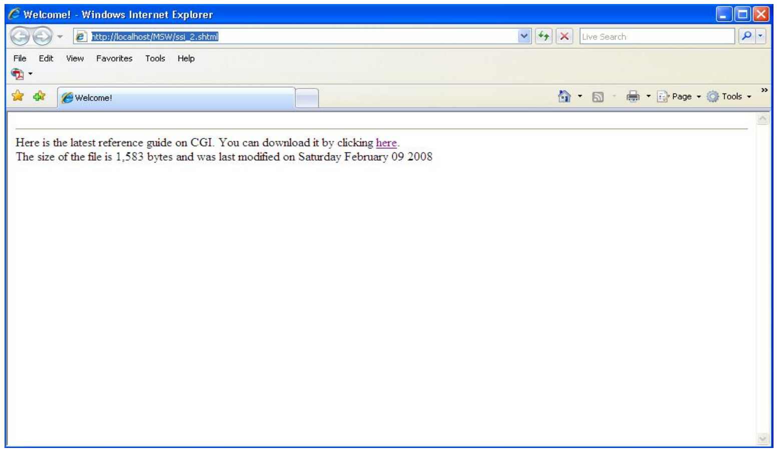
ssi_1.shtml

- `<!--#include file="address.html"-->`
- `<!--#include virtual="MSW/address.html"-->`

directory relative to the server root

There are SSI directives that allow you to retrieve certain information about files located on your server. For example, say you have a hypertext link in one of your documents that points to a manual describing your software that users can download. In such a case, you should include the size and modification date of that manual so users can decide whether it's worth their effort to download a document; it could be outdated or just too large for them to download.

```
<HTML> <HEAD><TITLE>Welcome!</TITLE></HEAD>
   <BODY>
Here is the latest reference guide on CGI. You can download it
by clicking <A HREF="O'Reilly CGI Programming.pdf">here</A>.
The size of the file is
<!--#fsize file="O'Reilly CGI Programming.pdf"--> bytes and was last
   modified on <!--#flastmod file="O'Reilly CGI Programming.pdf"-->
</BODY></HTML>
```


Welcome! - Windows Internet Explorer
http://localhost/MSW/ssi_2.shtml
Live Search
File Edit View Favorites Tools Help
Welcome!
Page Tools

Here is the latest reference guide on CGI. You can download it by clicking here.
The size of the file is 1,583 bytes and was last modified on Saturday February 09 2008

Executing CGI Programs

- You can use Server Side Includes to embed the results of an entire CGI program into a static HTML document, using the *exec cgi directive.*
- For example, there are many times when you want to display just one piece of dynamic data, such as:
- This page has been accessed 4883 times since December 10, 1995.
- This page has been accessed <!--#exec cgi="counter.pl"--> times.

Tailoring SSI Output

- The *config* SSI command allows you to select the way error messages, file size information, and date and time are displayed.
- For example, if you use the *include* command to insert a non-existing file, the server will output a default error message like the following:
- [an error occurred while processing this directive]
- By using the *config* command, you can modify the default error message. If you want to set the message to *"Error,* contact mareeswari@vit.ac.in" you can use the following:

<!--#config errmsg="*Error,* contact mareeswari@vit.ac.in" -->

- Here is how you can change the time format:

  <!--#config timefmt="%D %r"-->

  The file address.html was last modified on: <!--#flastmod file="address.html"-->.
- The output will look like this:
- The file address.html was last modified on: 12/23/95 07:17:39 PM
- The %D format specifies that the date should be in mm/dd/yy format, while the %r format specifies "hh/mm/ss AM|PM" format.

Common Errors

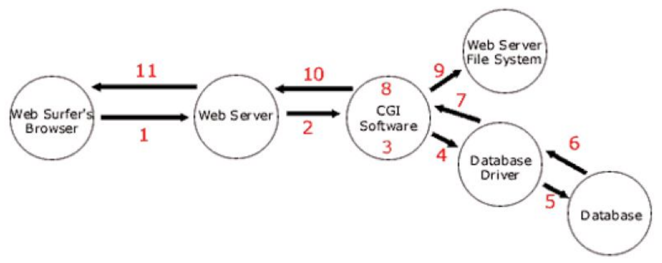
- There are two common errors that you can make when using Server Side Includes.
- First, you should not forget the "#" sign:

  <!--echo var="REMOTE_USER"-->
- Second, do not add extra spaces between the "-" sign and the "#" character:

  <!- - #echo var="REMOTE_USER"-->
- If you make either of these two mistakes, the server will not give you an error; rather it will treat the whole expression as an HTML comment.

Custom Database Script

- Web based applications often need to access information that is stored in some kind of a database, the information needed includes everything from user account information to product lists.
- Microsoft Access is a fairly simple database system intended to support small databases for individual use, or use by small workgroups.



- In this section we will explore the Win32:ODBC Perl module which provides access to nearly any database stored on a Windows based PC from a Perl program.
- ODBC stands for "Open Database Connectivity". This is a Microsoft technology that provides access to many relational database systems with one single programming interface. This simplifies the writing of programs that use relational databases, since you don't need to worry about the specifics of which database is used (Access vs. Orcale vs. whatever...), the program is the same no matter what database system is used. To use ODBC you *register* your specific database with the ODBC manager - this associates a name (called a "Data Source Name") with the specific database you have on your PC. Any program that wants to access your database needs to know the "Data Source Name", but nothing else (the program doesn't need to know if the database is actually part of an Oracle database or a Microsoft Access database).

Program: Database_Perl_ODBC.pl

```
use Win32::ODBC;
print "content-type:text/html\n\n";
#Create a database object and make sure the database was found
if (!($db = new Win32::ODBC('DSN_Account')))
{ printf("Error - the DSN_Account database could not be
   found\n");}
#Tell the database we want all the names and passwords in the
   Table_Account table
if($db->Sql("SELECT Name, Password FROM Table_Account"))
{   printf("Error reading from database\n");  }
```

```
#loop through all the rows in the database
print "<h1>CGI Program connects with MS Access Database
   </h1><hr><br>";
while ($db->FetchRow())
{
  # for each row (record) grab the name and password
  # we know the order - it matches our SELECT command!
  ($n,$p)=$db->Data();
  # print out the name and password
  print "Name:",$n,"<br>Password:",$p,"<hr>";
}
#Close the connection
$db->Close();
```

113 V.MAREESWARI / AP / SITE 8 July 2014

Insert Statement

- add a new name,password pair to the database

  `$db->Sql("INSERT INTO Table_Account (Name, Password) VALUES ('$n', '$p');");`

114 V.MAREESWARI / AP / SITE 8 July 2014

Server Security Issues

- **Why Should I Care?**
- CGI scripts are installed on Web servers and are thus open to visitors anywhere over the Internet. They are available for attack 24 hours a day, 365 days in a year. For standalone systems or systems in an intranet, attacks are only possible from a limited subset of users. On the Internet, attacks may originate from anywhere in the World, hundreds of miles away.
- Such kinds of sensitive data include the PIN of clients, their record of transactions and personal information like address or social security number etc. People are unlikely to have confidence in a company being unable to keep such data from being improperly manipulated. This directly leads to loss in revenue.

115 V.MAREESWARI / AP / SITE 8 July 2014

Some Forms of Attack

- HTML Form Tampering. Always carry out all verifications possible at the best of your knowledge before committing anything.
- Any of these hosts is able to read the content of the messages, or even to modify it. Under certain circumstances, other malicious hosts may also be able to eavesdrop the traffic through these hosts. Therefore, there is no confidentiality at all.

116 V.MAREESWARI / AP / SITE 8 July 2014

Safe CGI Scripting Guidelines

- You, as a CGI script and Web developer, may do your part to protect your customers by using the POST form transmission method instead of the GET method whenever privacy-sensitive form data are involved because form data are not carried in the URI. That does not eliminate the need for encrypted tunnels such as SSL to keep out of prying eyes over the network.
- Implement rings of security where applicable. A single line of defense is generally weak because if this line of defense is broken, intrusion will be successful.
- Configuration of SSL (Secure Socket Layer) or TLS (Transport Layer Security) is performed at the Web server and no modification to your CGI programs is needed.

What CGI scripts are known to contain security holes?

- Many of the ones that are identified here have since been caught and fixed, but if you are running an older version of the script you may still be vulnerable. Get rid of it and obtain the latest version.
- HotMail : The CGI scripts that run the popular HotMail e-mail system use a flawed security system that allows unauthorized individuals to break into user's e-mail accounts and read their mail. This problem is known to affect the version of HotMail that was in place as of December 1998.

Summary

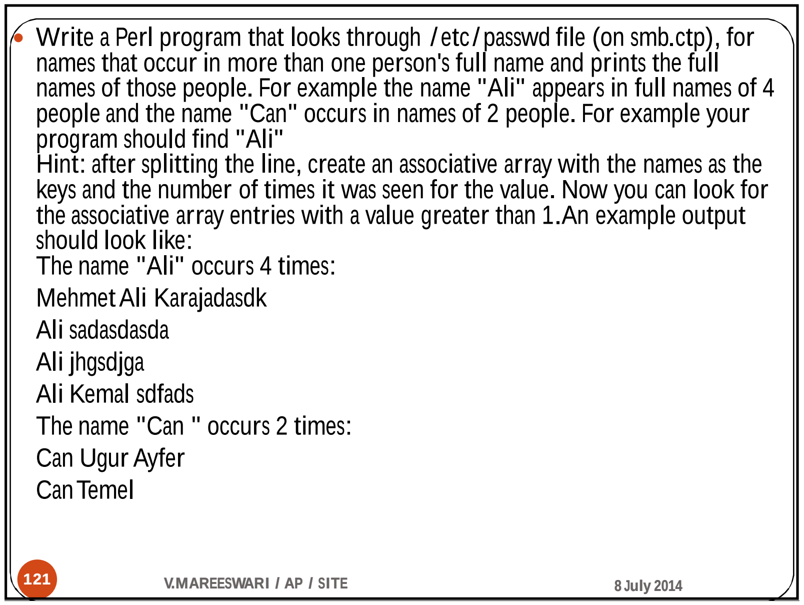
- CGI is a standard for interfacing executable files with Web servers.
- It allows for the interactive, dynamic, flexible features that have become standard on many Web sites, such as guestbooks, counters, bulletin boards, chats, mailing lists, searches, shopping carts, surveys, and quizzes. Several newer, faster means for accomplishing these same kinds of tasks have been developed, but CGI is more flexible in a number of ways.
- CGI is commonly used whenever one needs a Web server to run a program in real-time, take some kind of action, and then send the results back to a user's browser.
- Scripts can be written in any language that allows a file to be executed, but the most common language for CGI scripts is Perl.

Exercise

- Write a Perl program to input one line from the standard input (keyboard), with multiple words in it, and print these words on separate lines to the standard output (terminal).
  Eg. If the user types in “VIT University Vellore Tamilnadu India", your program should produce the lines

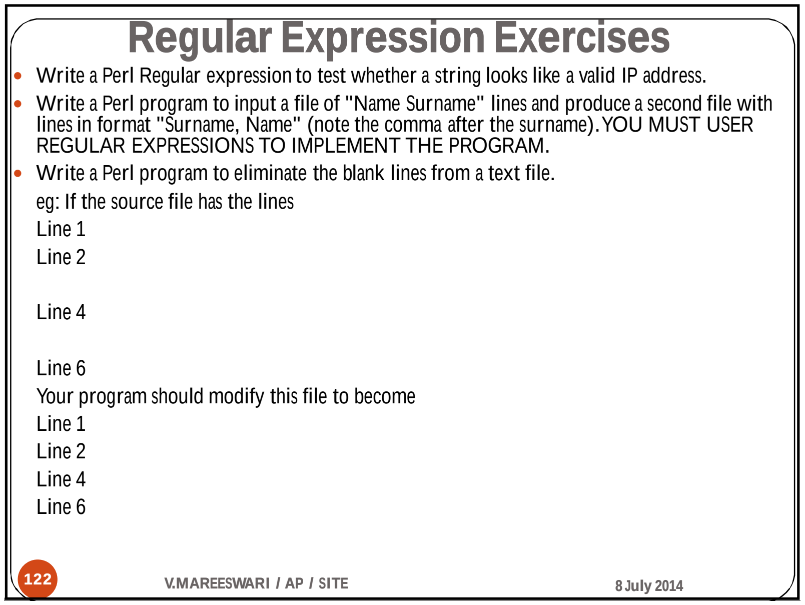
VIT
University
Vellore
Tamilnadu
India

- Write a Perl program that asks a user for a filename, then open that file and replace all "A"s with "Q"s and "a"s with "q"s. Then write the changes back to the file.



- Write a Perl program that looks through /etc/passwd file (on smb.ctp), for names that occur in more than one person's full name and prints the full names of those people. For example the name "Ali" appears in full names of 4 people and the name "Can" occurs in names of 2 people. For example your program should find "Ali"
  Hint: after splitting the line, create an associative array with the names as the keys and the number of times it was seen for the value. Now you can look for the associative array entries with a value greater than 1.An example output should look like:
  The name "Ali" occurs 4 times:
  Mehmet Ali Karajadasdk
  Ali sadasdasda
  Ali jhgsdjga
  Ali Kemal sdfads
  The name "Can " occurs 2 times:
  Can Ugur Ayfer
  Can Temel

121 V.MAREESWARI / AP / SITE 8 July 2014



Regular Expression Exercises

- Write a Perl Regular expression to test whether a string looks like a valid IP address.
- Write a Perl program to input a file of "Name Surname" lines and produce a second file with lines in format "Surname, Name" (note the comma after the surname). YOU MUST USER REGULAR EXPRESSIONS TO IMPLEMENT THE PROGRAM.
- Write a Perl program to eliminate the blank lines from a text file.

  eg: If the source file has the lines

  Line 1

  Line 2

  Line 4

  Line 6

  Your program should modify this file to become

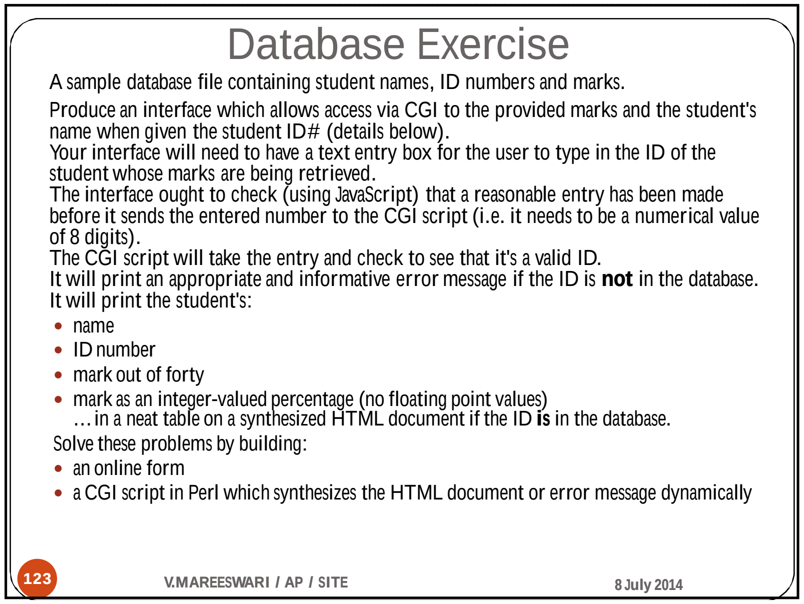
  Line 1

  Line 2

  Line 4

  Line 6

122 V.MAREESWARI / AP / SITE 8 July 2014



Database Exercise

A sample database file containing student names, ID numbers and marks.

Produce an interface which allows access via CGI to the provided marks and the student's name when given the student ID# (details below).
Your interface will need to have a text entry box for the user to type in the ID of the student whose marks are being retrieved.
The interface ought to check (using JavaScript) that a reasonable entry has been made before it sends the entered number to the CGI script (i.e. it needs to be a numerical value of 8 digits).
The CGI script will take the entry and check to see that it's a valid ID.
It will print an appropriate and informative error message if the ID is **not** in the database.
It will print the student's:

- name
- ID number
- mark out of forty
- mark as an integer-valued percentage (no floating point values)
  ... in a neat table on a synthesized HTML document if the ID **is** in the database.

Solve these problems by building:

- an online form
- a CGI script in Perl which synthesizes the HTML document or error message dynamically

123 V.MAREESWARI / AP / SITE 8 July 2014