Unit-III Introduction to PERL and Scripting

- Syllabus
- Introduction to PERL and Scripting
  - Scripts and Programs, Origin of Scripting, Scripting Today, Characteristics of Scripting Languages, Uses for Scripting Languages, Web Scripting, and the universe of Scripting Languages.
  - PERL- Names and Values, Variables, Scalar Expressions, Control Structures, arrays, list, hashes, strings, pattern and regular expressions, subroutines.

Script and Program

- A Script is a set of instructions that are interpreted. Scripts are written using a different kind of language called scripting languages like python, perl, ruby, shell script, VB Script, etc.

- A program is a set of instructions that are compiled. Programs are written using programming languages like C,C++ and JAVA etc.

Script and Program

Programming Languages	Scripting Languages
Compiler Based	Interpreter Based
More Syntax and highly coupled	Reduced syntax and loosely coupled
Convert all the code into binary and run	Run statement by statement
For big and complex programs	For small programs
Faster for large code	Faster for small code
Extra memory	No extra memory
Popular programming languages C,C++,JAVA and C#	Popular Scripting languages Python, Perl, Ruby, PHP

Origin of Scripting

- The word ‘script’ in a computing content in 1970’s, the originators of the UNIX of the UNIX operating system the term ‘shell script’ (sequence of commands)
- An Apple Macintosh Hypercard application, one of the early hypertext systems
- The associated HyperTalk language allowed the user to define sequence of actions to be associated with mouse click or movement, and these were called script.
- Initially a script as a sequence of commands to control an application or a device.

Scripting Today

- The term “scripting” is nowadays used with three different meanings.
  - A new style of programming which allows applications to be developed much faster than traditional methods.
    - The use of VB to develop GUI using pre built visual control
  - Using scripting language to manipulate, customize and automate the facilities of an existing system.
    - Use of client side scripting and Dynamic HTML to create interactive web pages.
  - Using scripting language with its rich functionality, we can use an alternate to a conventional languages for general programming tasks, particularly system programming and system administration.
    - Windows NT systems users PERL for administration . Apaches has embedded perl interpreter for GUI scripts.

Characteristics of Scripting Languages

- Both batch and interactive use
- Economy of expression
- Lack of declarations, simple scoping rules
- Flexible dynamic typing
- Easy access to other programs
- Sophisticated pattern matching and string manipulation
- High level data types.

Uses for Scripting Languages

- Scripting languages are of two kinds
  - Traditional Scripting
  - Modern Scripting
- Traditional Scripting
  - The activities which require traditional scripting include
    - System administration
    - Controlling Remote Applications
    - System and Application extensions
    - Experimental Programming
    - Command Line interface
- Modern Scripting
  - Visual Scripting
  - Scriptable components
  - Client side and Server side scripting

Web Scripting

- Web script, a computer programming language for adding dynamic capabilities to world wide web pages.
- Web scripting can be add information to a page as a reader uses it or let the reader enter information that may be passed on to the order department of an online business.
  - Processing web forms
  - Creating Dynamic web pages with enhanced visual effects and user interaction.
  - Dynamically Generating web pages "on the fly" from material held in a database.

The Universe of Scripting Languages

- Scripting can be traditional or modern scripting, and web scripting forms an important part of modern scripting.
- Scripting universe contains multiple overlapping worlds
  - The original UNIX world of traditional scripting using perl.
  - The Microsoft world of VB and Active controls
  - The world of VB for scripting compound documents.
  - The world of client –side and server side web scripting
  - The overlap is complex, for example web scripting can be done in Vbscript, JavaScript /Jscript, perl or TCL.
  - The universe has been enlarged as Perl and TCL are used to implement complex applications for large organizations.
    - Example TCL has been used to develop a major Banking system, and Perl has been used to implement an enterprise wide document management system for leading aerospace company.

Perl

- Perl is a programming language developed by Larry Wall, specially designed for text processing.
- Its typical use for extracting information from a textfile and printing out report for converting a text file into another form.
- This is because it got its name after the expression “Practical Extraction and Report Language”.
- Perl takes the best features from other languages, such as C, awk, sed, sh, and BASIC, among others.
- Perl's database integration interface DBI supports third-party databases including Oracle, Sybase, Postgres, MySQL, and others.
- Perl works with HTML, XML, and other mark-up languages.
- Perl supports Unicode.
- Perl supports both procedural and object-oriented programming.
- Perl supports both procedural and object-oriented programming.
- Perl interfaces with external C/C++ libraries through XS or SWIG (The Simplified Wrapper and Interface Generator) –Connector C libraries to scripting languages

Perl

- Perl is extensible. There are over 20,000 third party modules available from the Comprehensive Perl Archive Network (CPAN).

- The Perl interpreter can be embedded into other systems such as web servers and database servers.

- Perl is Open Source software, licensed under its Artistic License

- Perl can be embedded into Apache web servers to speed up processing.
- In Perl, a comment begins with a hash (#) character. Perl interpreter ignores comments at both compile-time and runtime.

Names and Values

- The data can be either number, characters, or more complex such as a list.
- Data is held as value. The following examples are values:

  10

  20.2

  "Perl syntax"

- To hold a piece of data, you need variables. You use a variable to store a value. And through the name of the variable, you can process the value.
- The following illustrates some variables in Perl:

```
$x = 10;
$y = 20;
$s = "Perl string";
```

Expressions

In Perl, an expression is anything that returns a value.

- The expression can be used in a larger expression or a statement.
- The expression can be a literal number, complex expression with operators, or a function call.
- For example, 3 is an expression that returns a value of 3. The $a + $b is an expression that returns the sum of two variables: $a and $b.

Statements

- A statement is made up of expressions. A statement is executed by Perl at run-time.
- Each Perl statement must end with a semicolon (;).
- The following example shows the statements in Perl:

```
$c = $a + $b;

print($c);
```

Blocks

- A block is made up of statements wrapped in curly braces {}.
- You use blocks to organize statements in the program.
- The following example illustrates a block in Perl:

```
{
        $a = 1;
        $a = $a + 1;
        print($a);
}
```

Whitespace

- Whitespaces are spaces, tabs, and newlines.
- Perl is very flexible in terms of whitespaces usages. Consider the following example:

  $x = 20;

  $y=20;
- Code language: Perl (perl)Both lines of code work perfectly.

Keywords

- Perl has a set of keywords that have special meanings to its language.

- Perl keywords fall into some categories such as built-in function and control keywords.

Variables

- A variable is a place to store values.
- They can be manipulated throughout the program.
- When variables are created they reserve some memory space.
- There are three types of variables:
  - Scalar defined by $
  - Arrays defined by @
  - Hashes defined by %
  - Subrounte defined by(&)

Variables

Scalar Variable

```
$name = "Anastasia";
$rank = "9th";
$marks = 756.5;
Print  “$name”
Print  “$rank”
Print  “$marks”
```

Scalar Operations

```
my $x = 5;
say $x;
my $y = 3;
say $y;
say $x + $y;
say $x . $y;
say $x x $y;
```

Variables-Scalar (single)-Expressions

SCALAR EXPRESSIONS

- **Scalar data items are combined into expressions using operators.**

```
$c=12;
$d=3+($c);
```

- **1) ARITHMETIC OPERATORS**

Perl provides usual arithmetic operator including auto-increment and auto-decrement.

Operator	Example	Result	Definition
+	7 + 7	= 14	Addition
-	7 - 7	= 0	Subtraction
*	7 * 7	= 49	Multiplication
/	7 / 7	= 1	Division
**	7 ** 7	= 823543	Exponents
%	7 % 7	= 0	Modulus

```
$c=17;
$d=++$c;   //increment then assign
$c=12;
$d=$c ++; //assign then increment
```

Binary arithmetic operation:

```
$a+=3;
$a=a+3;
```

Variables-An Array

- An array is a variable that stores an ordered list of scalar values.
- Array variables are preceded by an "at" (@) sign.
- To refer to a single element of an array, you will use the dollar sign ($) with the variable name followed by the index of the element in square brackets.
- Here is a simple example of using the array variables −

```
@ages = (25, 30, 40);
@names = ("John Paul", "Lisa", "Kumar");
print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n“
```

OUTPUT:

```
$ages[0] = 25
$ages[1] = 30
$ages[2] = 40
$names[0] = John Paul
$names[1] = Lisa
$names[2] = Kumar
```

Variables-Hashes

- A Perl hash is defined by key-value pairs.
- Perl stores elements of a hash in such an optimal way that you can look up its values based on keys very fast.
- Example

```
#!/usr/bin/perl
use warnings;
use strict;
# defines country => language hash
my %langs = ( England => 'English',
 France => 'French',
 Spain => 'Spanish',
 China => 'Chinese',
 Germany => 'German');
# get language of England
my $lang = $langs{'England'}; # English
print($lang,"\n");
```

Variables-Subroutine

- A Perl subroutine or function is a group of statements that together performs a task. You can divide up your code into separate subroutines.
- Define and Call a Subroutine
- The general form of a subroutine definition in Perl programming language is as follows −

```
sub subroutine_name  {
 body of the subroutine }
```

The typical way of calling that Perl subroutine is as follows −

```
subroutine_name( list of arguments );
```

EXAMPLE

```
sub Hello {
print "Hello, World!\n"; }
Hello(); # Function call
```

When above program is executed, it produces the following result −

```
Hello, World!
```

Control Structures

- If-else
- if-elsif-else
- while/until
- for/foreach

Control Structures

If-else

Syntax

The **if** statement allows an optional else clause:

```
if ( condition ) { true block } else { false block }
```

Example:

```
my ($a, $b);
print "Enter 2 numbers, one per line\n";
chomp($a = <STDIN>);
chomp($b = <STDIN>);
if ( $a < $b ) {
    print "$a then $b\n";
}
else {
    print "$b then $a\n";
}
```

Control Structures

if-elsif-else.

Syntax

The **if-else** statement can be augmented with **elsif** clauses, which are like a shortcut for "**else if …**". Notice there is no '**e**' before the '**i**' in **elsif**.

The syntax is

```
if     ( condition )   { block 1 }
elsif  ( condition )   { block 2 }
elsif  ( condition )   { block 3 }
# and so on …
else   {block N }
```

Control Structures

if-elsif-else

Example:

```
my ($a, $b);
print "Enter 2 numbers, one per line\n";
chomp($a = <STDIN>);
chomp($b = <STDIN>);
if ( $a < $b ) {
    print "$a is less than $b\n";
} elsif ( $b < $a ) {
    print "$b is less than $a\n";
} else {
    print "The numbers are equal.\n";
}
```

Control Structures

for:

Syntax

for (init statement; condition; increment/decrement)

{

Code to be Executed

}

Example:

```
# Perl program to illustrate the for loop

for ($count = 1 ; $count <= 3 ; $count++)
{
    print "$count"


}
```

Control Structures

```
foreach variable
 {
# Code to be Executed
 }
```

Example:

```
# Array
@data = (‘for', ‘each', ‘example');

# foreach loop
foreach $word (@data)
{
    print $word
}
```

Control Structures

```
while (condition)
{
 # Code to be executed
 }
```

Example:

```
$a = 10;
# while loop execution
while( $a < 20 )
 {
 printf "Value of a: $a\n";
$a = $a + 1;
}
```

OUTPUT:
10 11 12 13 14 15 16 17 18 19 20

do.. while
do { # statements to be Executed }
while(condition);

Example:

```
$a = 10;
# do..While loop
do {
      print "$a
";
    $a = $a - 1;
} while ($a >
0);
```

OUTPUT:
10 9 8 7 6 5 4 3 2 1

Control Structures

```
until (condition)
{
# Statements to be executed
}
```

Example:

```
# until loop
a=10
until ($a < 1)
{
     print "$a ";
     $a = $a - 1;

OUTPUT:
10 9 8 7 6 5 4 3 2 1
}
```

Perl Strings

- In perl, a string is a sequence of characters surrounded by some kinds of quotation marks.
- Ex: $str1="string with double quotes";
  $str2='string with single quote";
- Note: Double quoted string replaces variables inside it by their values, while the single quoted strings treats them as text.
- The operator make it easy to manipulate a string in different ways. There are two types of string operators. They are
  - Concatenation (.)
  - Repetition(X)
- **Example**
- Print "this" ."is" ."perl strings"; # output: this is perl strings
- Print "hello"X4 # prints hellohellohellohello

Perl Strings functions

- length(string);
- uc(string);
- lc(string);
- Index(string, substring);
- Substr(string, starting_position,ending_position)

Perl string length

To find the number of characters in a string, you use the `length()` function. See the following example:

```
my $s = "This is a string\n";
print(length($s),"\n"); #17
```

Perl Strings functions

Changing cases of string

To change the cases of a string you use a pair of functions `lc()` and `uc()` that returns the lower case and upper case versions of a string.

```
my $s = "Change cases of a string\n";
print("To upper case:\n");
print(uc($s),"\n");

print("To lower case:\n");
print(lc($s),"\n");
```

Perl Strings functions

Example

```
my $s=“Learning perl is easy”
my $sub=“perl”;
my $p=index($s, $sub);
Print(qw\The substring “$sub” found at position “$p” in string “$s”\,
    “\n”);
```

OUTPUT:

The substring perl found at position 9 in string Learning perl is eary.

Perl Strings functions

Get or modify substring inside a string

To extract a substring out of a string, you use the `substr()` function.

```
#!/usr/bin/perl
use warnings;
use strict;
# extract substring
my $s = "Green is my favorite color";
my $color  = substr($s, 0, 5);      # Green
my $end    = substr($s, -5);        # color

print($end,":",$color,"\n");

# replace substring
substr($s, 0, 5, "Red"); #Red is my favorite color
print($s,"\n");
```

Perl List

- A Perl list is a sequence of scalar values. You use parenthesis and comma operators to construct a list.

- Each value is the list is called list element.

- List elements are indexed and ordered. You can refer to each element by its position.

Simple Perl List

Simple Perl list

The following example defines some simple lists:

```
();
(10,20,30);
("this", "is", "a","list");
```

In the example above:

- The first list `()` is an empty list.
- The second list `(10,20,30)` is a list of integers.
- The third list `("this", "is", "a","list")` is a list of strings.

Complex Perl List

Complex Perl list

A Perl list may contain elements that have different data types. This kind of list called a complex list.

Let's take a look at the following example:

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


my $x = 10;
my $s = "a string";
print("complex list", $x , $s ,"\n");
```

Perl List –Accessing

- We can access the element of a list by using the zero based index. To access the nth element, we need to put (n-1) index inside square brackets.

- Syntax: $listname[index];

- Ranges: Perl allows you to build a list based on range of numbers or characters
  - Example: (1..20), (a..z)

Perl List –using qw function

Using qw function

Perl provides the `qw()` function that allows you to get a list by extracting words out of a string using the space as a delimiter. The qw stands for quote word. Two lists below are the same:

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

print('red','green','blue'); # redgreenblue
print("\n");

print(qw(red green blue)); # redgreenblue
print("\n");
```

Perl List –using Flattening List

Flattening list

If you put a list, called internal list, inside another list, Perl automatically flattens the internal list. The following lists are the same:

```
(2,3,4,(5,6))
(2,3,4,5,6)
((2,3,4),5,6)
```

Perl List –slicing List

```
@list1=(1, “hello”,3, “for”,5);
@list2=@list1[1,2,4]; #slice positions
Print “sliced list: @list2;
```

OUTPUT:

Hello 3 5

Perl Arrays

- A Perl array variable stores an ordered list of scalar values.
- To refer a single element of Perl array, variable name will be preceded with dollar ($) sign followed by index of element in the square bracket.
- **Syntax**

```
@arrayName = (element1, element2, element3..);
```

```
#!/usr/bin/perl
@num = (2015, 2016, 2017);
@string = ("One", "Two", "Three");
print "$num[0]\n";
print "$num[1]\n";
print "$num[2]\n";
print "$string[0]\n";
print "$string[1]\n";
print "$string[2]\n";
Print “@num”;
```

Output:

```
2015
2016
2017
One
Two
Three
```

Perl Array accessing

- To access a single element of a Perl array, use ($) sign before variable name.
- You can assume that $ sign represents singular value and @ sign represents plural values.
- Variable name will be followed by square brackets with index number inside it.
- Indexing will start with 0 from left side and with -1 from right side.

```
@months = qw/Jan Feb Mar Apr May Jun Jul Aug Sept Oct Nov Dec/;
print "$months[0]\n";
print "$months[1]\n";
print "$months[7]\n";
print "$months[9]\n";
print "$months[6]\n";
print "$months[-3]\n";
```

Output:

```
Jan
Feb
Aug
Oct
Jul
Oct
```

Perl Array accessing

Perl Array Size or Length

The size of an array is determined with scalar context on the array. The returned value will be always one greater than the largest index. In short the size of an array will be ($#array + 1). Here, $#array is the maximum index of the array.

```
@array = (you, me, us);
$array[5] = 4;
$size = @array;
$index_max = $#array;
print "Size:  $size\n";
print "Maximum Index: $index_max\n";
```

Output:

```
Size: 6
Maximum Index: 5
```

Perl Array Functions

- You can add or remove an element from an array using some array functions.
- We'll discuss following array Perl functions:
  - Push
  - Pop
  - Shift
  - Unshift

Push on Array

1) Push on Array

The push array function appends a new element at the end of the array.

```
@array = ("pink", "red", "blue");
push @array, "orange";
print "@array\n";
```

Output:

```
pink red blue orange
```

In the above program, "orange" element is added at the end of the array.

Pop on Array

2) Pop on Array

The pop array function removes the last element from the array.

```
@array = ("pink", "red", "blue");
pop @array;
print "@array\n";
```

Output:

```
pink red
```

In the above program, "blue" element is removed from the end of the array.

Shift on Array

3) Shift on Array

The shift array function removes the left most element of array and thus shorten the array by 1.

```
@array = ("pink", "red", "blue");
shift @array;
print "@array\n";
```

Output:

```
red blue
```

In the above program, "pink" is removed from the array.

Unshift on Array

4) Unshift on Array

The unshift array function adds a new element at the start of the array.

```
@array = ("pink", "red", "blue");
unshift @array, "orange";
print "@array\n";
```

Output:

```
orange pink red blue
```

In the above program, "orange" is added at the start of the array.

splice on Array

Perl Replacing Array Elements, splice()

The splice array function removes the elements as defined and replaces them with the given list.

```
@alpha = (A..Z);
print "Before - @alpha\n";
splice(@alpha, 8, 8, U..Z);
print "After - @alpha\n";
```

Output:

```
Before - A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
After -  A B C D E F G H U V W X Y Z Q R S T U V W X Y Z
```

In the above program, the replacement begins counting from 9th position (I) to 8 elements that is P.

sorting on Array

Perl Sorting Arrays, sort()

To sort an array, sort() array function is used. The sort() function sorts all the elements of an array according to the ASCII standard.

```
# defining array
@days = ("sun", "mon", "tue", "wed", "thu", "fri", "sat");
print "Original array: @days\n";
# sorting array
@days = sort(@days);
print "Sorted array: @days\n";
```

Output:

```
Original array: sun mon tue wed thu fri sat
Sorted array: fri mon sat sun thu tue wed
```

Merging TWO arrays

Perl Merging Two Arrays, merged()

Two arrays can be merged together using merged() function as a single string removing all the commas in between them.

```
#two arrays
@array1 = ("Girl", "in", "front", "of", "me");
@array2 = ("is", "very", "beautiful");
#merging both the arrays
@merged = (@array1, @array2);
print "@merged\n";
```

Output:

```
Girl in front of me is very beautiful
```

Splitting arrays

Perl Strings to Arrays, split()

With the help of split() function, we can split a string into array of strings and returns it.

```
# original string
$string = "Where-There-Is-A-Will-There-Is-A-Way";
# transforming strings into arrays.
@string = split('-', $string);
print "$string[4]\n";
```

Output:

```
Will
```

In the above program, we have transformed $string into array at hyphen (-) values. Now from this array, we have printed fourth element of the array.

Join Arrays

Perl Arrays to Strings, join()

The join() function is used to combine arrays to make a string. It combines the separate arrays into one string and returns it.

```
# original string
$string = "Where-There-Is-A-Will-There-Is-A-Way";
# transforming arrays into strings.
@string = split('-', $string);
$string_full = join( '-', @string );
print "$string_full\n";
```

Output:

```
Where-There-Is-A-Will-There-Is-A-Way
```

In the above program, the string is splitted at hyphens (-). We have used join() in $string_full and printed it.

Perl Hashes

- The hashes is the most essential and influential part of the perl language.
- A hash is a group of key-value pairs.
- The keys are unique strings and values are scalar values. Hashes are declared using my keyword.
- Hashes are same like as arrays, but hashes are unordered and also the hash elements are accessed using its value while array elements are accessed using its index value.

- The variable name starts with a (%) sign.

- Syntax

```
my %hashName = (  "key" => "value”);
```

Perl Hash Accessing

- To access single element of hash, ($) sign is used before the variable name and then key element is written inside {} braces.

Example:

```
my %capitals = ( "India"  => "New Delhi", "South Korea" => "Seoul", "USA"  => "Wa
    shington, D.C.", "Australia"  => "Canberra" );
print " $capitals{'India'}\n";
print "$capitals{'South Korea'}\n";
print "$capitals{'USA'}\n";
print "$capitals{'Australia'}\n";
```

OUTPUT:

New Delhi

Seoul

Washington D.C

Canberra

Perl sorting Hash by key using foreach

- You can sort a hash using either its key element or value element. Perl provides a sort() function for this. In this example, we'll sort the hash by its key elements.

```
my %capitals = (
    "India"  => "New Delhi",
    "South Korea" => "Seoul",
    "USA"  => "Washington, D.C.",
    "Australia"  => "Canberra"
);
# Foreach loop
foreach $key (sort keys %capitals) {
    print "$key: $capitals{$key}\n";
}
```

Output:

```
Australia: Canberra
India: New Delhi
South Korea: Seoul
USA: Washington: D.C.
```

Look at the output, all the key elements are sorted alphabetically.

Perl Removing Hash Elements

- To remove a hash element, use **delete()** function.
- Here, we have removed both the key-value pairs which were added in the last example.

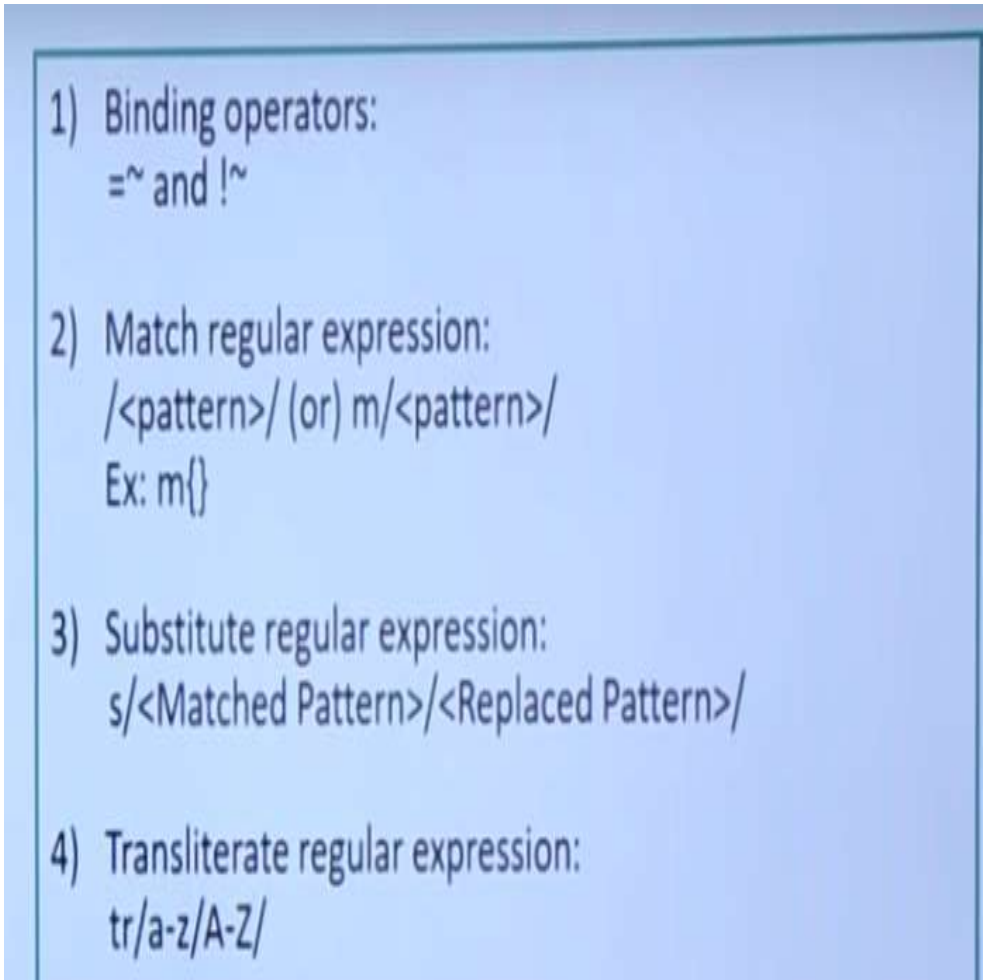
```
my %capitals = (
   "India"  => "New Delhi",
   "South Korea" => "Seoul",
   "USA"  => "Washington, D.C.",
   "Australia"  => "Canberra"
   "Germany "  => " Berlin"
   " UK "  => "London"
);
while (($key, $value) = each(%capitals)){
    print $key.", ".$value."\n";  }
#removing element
delete($capitals{Germany});
delete($capitals{UK});
# Printing new hash
print "\n";
while (($key, $value) = each(%capitals)){
    print $key.", ".$value."\n";  }
```

Output:

```
Australia, Canberra
India, New Delhi
USA, Washington D.C.
South Korea, Seoul
```

Patterns

- A pattern is a sequence of characters to be searched for in a character string.
- In perl, patterns are normally enclosed in slash character : ex. /def/



Patterns

- Example:

```
$line = "This is perl regular expression ";
if ($line =~ /perl/)
{
   print "Matching\n";
}
else
{
   print "Not Matching\n";
}
```

OUTPUT: Matching

```
$line = "This is perl regular expression ";
if ($line =!~ /perl/)
{
   print "Matching\n";
}
else
{
   print "Not Matching\n";
}
```

OUTPUT: Not Matching

Patterns

```
$true = ($foo =~ m/foo/);
```

will set $true to 1 if $foo matches the regex, or 0 if the match fails.

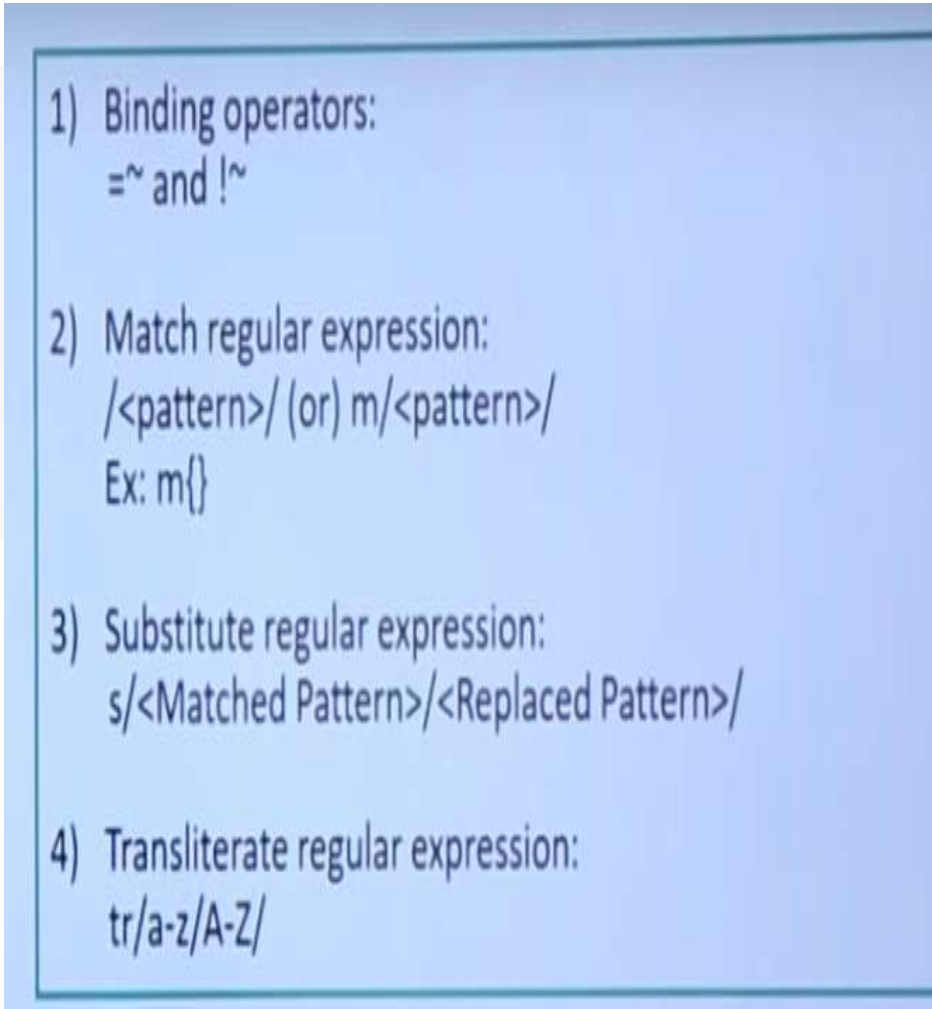
```
$string = “perl has four kinds of  variables";

$string =~ s/four/4/;

# Printing the updated string

print "$string\n";
```

OUTPUT:
perl has 4 kinds of variables



Patterns

1) Case insensitive:
   m/<pattern>/i

2) Global flag:
   s/<pattern>/<replace>/g → replaces all occurrences
   s/<pattern>/<replace>/ → replaces the first occurrence

Regular Expressions

- A regular expression is a string of characters that defines the pattern or patterns you are viewing
- A RE is also referred as regex or regexp
- A regular expression can be either simple or complex, depending on the pattern you want to match
- Syntax: string =~regx;
- There are three RE operators with perl
  - Match regular expression –m//
  - Substitute regular expression –s//
  - Translator regular expression –tr//

Regular Expressions-Examples

- $foo =~ m/this\|that/
  - Matches the string either this or that
- $string =~ s/a/b/;
  - This will replace the first “a” in $string with a “b”.
- $string =~ s/a/b/g;
  - put a “g” for global at the end of the line
- $string =~ s/[^0-9]//g;
  - This replaces anything matched by the first expression,
- $string =~ s/[aeiou]/[AEIOU]/g;
  - to make all vowels uppercase
- $string =~ tr/aeiou/AEIOU/;
  - Translation works on a per character basis, replacing each item in the first list with the character at the same position in the second list.

Complex Regular Expressions: Meta characters

^	Matches beginning of a line followed
$	Matches end of the line
.	Matches single character except newline. To match a newline use m//
[..]	Any character inside square brackets
[^...]	Matches excluding the characters inside square brackets
*	Matches 0 or more preceding expressions
+	Matches 1 or more preceding expressions
?	Matches 0 or 1 preceding expressions
{n}	Matches n occurrences of preceding expression
{n,}	Matches n or more occurrences of preceding expression

Subroutines

- A perl function or subroutine is a group of statements that together perform a specific task.
- The word subroutines is used most in perl programming because it created using “sub”
- Syntax

```
Sub subroutine_name
{
#body of the function or subroutine
}
```

- In perl scripting, subroutines can be called by passing the parameters list to it as follows
- Subrountine_name(parameters_list);

Subroutines

- **Example1**

```
sub add(a,b);
{
return (a+b);
}
$sum=add(4,6);
Print “$sum”;
```

Example2:

```
#!/usr/bin/perl # below is function definition
sub Print_hello
{
print(″Hello, Perl¥n″);
}
 # below is function call
Print_hello();
```

Subroutines

- Advantages
  - It helps to reuse the code
  - Organizing the code in structural format
  - It increases code readability