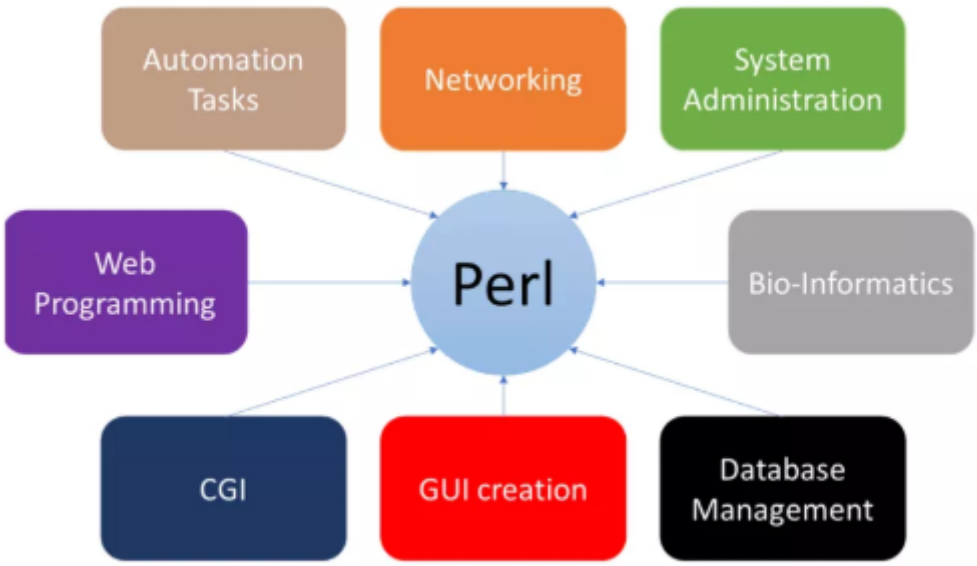
PERL

Creator

- Larry Wall, programmer, linguist (person who studies foreign language), author
- Born March 10, 1949 in Duncan, British Columbia, Canada.
- Perl creation in 1987.

What is Perl?

- A popular general-purpose programming language- extensively used in Bioinformatics.
- Originally developed for text manipulation. Used for wide range of tasks
  - Web development
  - System administration
  - Network programming
  - Code generation
  - Database interaction and more
- Supports both procedural and OO programming
- It has one of the world's most impressive collections of modules (http://www.cpan.org)
- An application, like any other application we can install in our system
- Freely available, runs on different operating systems:
  - Linux, Windows, Machintosh etc.
- Application takes the program in text format, translates it into instructions so that computers can understand and runs the program.
- It is a scripting language
- It is called Perl and not PERL



Perl - Pros

- Open source and free licencing
- Fast, flexible, secure and fun
- Excellent text handling and regular expressions
- Interpreted language
- Code developed is generally cross platform (portability)
- Very easy to write powerful programs in a few lines code
- Availability of modules (CPAN)
- Illegible (not clear) syntax
- Programmers and readability (hard to read programs)
- Limited GUI support
- Too many special constructs
- Hard to build data structures
- Difficult to fix bugs
- Define interfaces

Beginning with Perl:

- Perl –v displays the version Perl
- perldoc perl gives list of manual pages
- perldoc –h gives brief summary of options available.
- Perl file ends with .pl (demo.pl)
- The complete document for Perl is found in (http://www.perl.com)
- For windows operating system the current standard Perl distribution: ActivePerl from ActiveState
  http://www.activestate.com/activeperl/

- How to write: text format (notepad) – program name.pl
- How to run :

```
perl  program name.pl  // windows
./filename.pl  // unix
```

First line of Perl:

#!/usr/bin/perl
#! is commonly called a "shebang" and it tells the computer how to run a script.

/usr/bin/perl is the command used to run perl interpreter (location of the Perl binary)
So to start a script we need to add above lines as a first line to make Perl script executable.

Basic syntax:

- '#' is used for commenting the line
- All statements should end with ';'
- '$_' is the special variable called default variable
- Perl is case sensitive
- Perl program is compiled and run in a single operation.

Simple program:

- #!/usr/bin/perl
- print "Come and Enjoy Perlfalls!";
- This program displays
- Come and Enjoy Perlfalls!
- #!/usr/bin/perl -c
- print "Come and Enjoy Perlfalls";
- This program displays
- Syntax ok

Basic Options:

- -c : check syntax and exit without executing the script
- -v : prints the version of Perl
- -w : print warnings
- -e : used to enter and execute a line of script on the command line
- Ex:
- On the prompt/shell
- perl –e 'print "Hello, World!"'

Basic syntax for print:

- print "Hai! I am going to learn Perl";
- print 'Hai! I am going to learn Perl';
- print("Hai! I am going to learn Perl");

Standard Files:

- STDIN : normal input channel for the script
- STDOUT : normal output channel
- STDERR : normal output channel for errors

Three basic data type:

- Scalar $
- Array @
- Hash (Associative array) %

Scalar :

- represents number (integer, float),
  - string (a set of characters)
- Single unit of data
  - Numeric scalar
  - String scalar

$number = 50;
#number is a variable name;

$float =20.5;

$name = Aheli;
print $name;

print “Welcome $name”;

```
@array = (Aheli, 20, Ankur, 21);
print “@array \n”;

print “\$array[0] = $array[0] \n”;
print “\$array[1] = $array[1] \n”;
print “\$array[2] = $array[2] \n”;
print “\$array[3] = $array[3] \n”;
```

```
%data = ('John ', 45, 'Lisa', 30, 'Kumar', 40);
print "\$data{'John '} = $data{'John '}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

Output:

```
$data{'John'} = 45
$data{'Lisa'} = 30
$data{'Kumar'} = 40
```

SCALAR

Scalar - a single unit of data

Data : integer number
floating point
character
string
paragraph

Example :

```
#!/usr/bin/perl
$age = 20 ;        # An integer assignment
 $marks = 80.5;      # A floating point
$name = "Naga";      # A string
print "Age = $age\n";
print "Marks = $marks\n";
print "Name = $name\n";
```

Output:

Age = 20
Marks = 80.5
Name =Naga

Numeric Scalars

```
#!/usr/bin/perl
$integer = 100;
$negative = -100;
$float = 100.123;
$bigfloat = -1.2E-23;
print "Integer = $integer\n";
print "Negative = $negative\n";
print "Float = $float\n";
print "Bigfloat = $bigfloat\n";
```

String Scalars

```
 #!/usr/bin/perl
$var = "It is a string scalar!";
$single = 'I am inside single quote - $var';
$double = "It is inside double quote - $var";
$escape = "It is example of escape-\tHello, World!";
print "var = $var\n";
print "single quote = $single\n";
print "double = $double\n";
print "escape = $escape\n";
```

Scalar operations

```
#!/usr/bin/perl
$str = "hello" . "world";
     # Concatenates strings
$num = 5 + 10;       # adds two numbers.
$mul = 4 * 5;  # multiplies two numbers
$mix = $str . $num;
     # concatenates string and number
print "str = $str\n";
print "num = $num\n"; print "mul = $mul\n";
print "mix = $mix\n";
```

Perl Function

1. Chomp and chop

```
$a = "name\n";
chomp($a);
print $a;
```

```
$a = "name\n";
chop($a);
print $a;
```

abs

syntax : abs()

```
print abs(10) #print10

print abs(-10) #print 10
```

chr

- returns the ASCII character represented by number

```
print chr(97)
#print the letter 'a'
     hex  - returns the decimal equivalent
$number= hex("ffff10");
# 16776976
```

index

index str, substr

```
$str= "This is a line of text."
$substr= "line";
$position = index $str, $substr;
# position = 10
```

int

int expr

```
$int = int (10.23);
print lc “PERL”;
print lcfirst “PERL”;
```

uc

ucfirst

substr

```
$string = “Now is the time for all good people”;
$sub1 =substr($string,7,8);
$sub2 =substr($string,7);
```

Perl Data Types

Arrays
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.

Array creation
```
@number=(1,2,3,4,5);

@names =(“John”, “Mary”, “Larry”);

@mixarray=(“John”, 40, “Mary”, 35);
```

Array creation

```
@numbers=(1,2,3,4,5);
        @names =("John", "Mary", "Larry");
        print @numbers;
        print "\n";
        print "@numbers\n";
        print @names;
        print "\n";
        print "@names\n";
```

```
12345
1 2 3 4 5
JohnMaryLarry
John Mary Larry
Press any key to continue . . .
```

Array creation

```
@numbers=(1,2,3,'hello');
        @array = qw/This is an array/;
        #@array = qw(This is an array);
        print "@numbers\n";
        print "@array\n ";
```

```
@numbers=(1,2,3,'hello');
        @array = qw/This is an array/;
        #@array = qw(This is an array);
        print "@numbers\n";
        print "@array\n ";
```

```
1 2 3 hello
This is an array
 Press any key to continue . . .
```

Retrieve elements from an Array

```
#!/usr/bin/perl
@days = qw/Mon Tue Wed Thu Fri Sat Sun/; print "$days[0]\n";
     print "$days[1]\n";
     print "$days[2]\n";
     print "$days[6]\n";
     print "$days[-1]\n";
     print "$days[-7]\n";
     Output??
```

Sequential Number Array

```
#!/usr/bin/perl
@var_10 = (1..10);
@var_20 = (10..20);
@var_abc = (a..z);
print "@var_10\n";
print "@var_20\n";
print "@var_abc\n";
     Output??
```

Array Size

```
scalar : size of an array
$#array_name  : max index
```

```
#!/usr/bin/perl

@array = (1,2,3);
$array[50] = 4;

$size = @array;
$max_index = $#array;

print "Size:  $size\n";
print "Max Index: $max_index\n";
```

This will produce the following result –

```
Size: 51

Max Index: 50
```

Adding and removing elements in Array

```
#!/usr/bin/perl

# create a simple array
@bases = qw(A T G  C C);
print "@bases = @bases\n";

# add one element at the end of the array
push(@bases, "A");
print "\@bases = @bases\n";

# add one element at the beginning of the array
unshift(@bases, "A");
print "\@bases = @bases\n";
```

Adding and removing elements in Array

```
#!/usr/bin/perl
# create a simple array
@bases = qw (A T G  C C);
print "\@bases = @bases\n";
# remove one element from the end
     $end=pop(@bases); print "$end\n";
     print "\@bases = @bases\n";
# remove one element from the beginning of the array
$begin=shift(@bases); print "$begin\n";
   print "\@bases = @bases\n";
```

Slicing Array Elements

```
     #!/usr/bin/perl
     @days = qw/Mon Tue Wed Thu Fri Sat Sun/; @weekdays =
@days[3,4,5];
     #@weekdays = @days[3..5];
      print "@weekdays\n";
```

Replacing Array element

```
     splice (@ARRAY, OFFSET, LENGTH , LIST)
```

-remove the elements of @ARRAY designated by OFFSET and LENGTH, and replaces them with LIST, if specified.

Replacing Array element

```
#!/usr/bin/perl
@nums = (1..20);
print "Before - @nums\n";
splice(@nums, 5, 5, 21..25);
print "After - @nums\n";
```

Replacing Array element

```
Before - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
After - 1 2 3 4 5 21 22 23 24 25 11 12 13 14 15 16 17 18 19 20
```

Reverse of an Array

```
     @nums = (1..5);
     print “@nums\n”;
 @reverse = reverse @nums ;
     print “@reverse\n”;
```

Sorting Arrays

```
     # define an array
     @program = qw(Perl C Python Java)
      print “Before: @program\n”;
          #alphabetic order
      @program = sort(@program);
          print “After: @program\n”;
```

Sorting Arrays

```
# define an array
@nos = (0,100,-5,50,300);
 print “Before: @nos\n”;
 @nos = sort(@nos);
     print “After: @nos\n”;
```

Sorting Arrays

```
# define an array
@nos = (0,100,-5,50,300);
 print “Before: @nos\n”;
 @nos = sort(@nos);
     print “After: @nos\n”;
```

Sorting Arrays

```
# define an array
@unsorted = (0,100,-5,50,300);
 print “Before: @unsorted\n”;
#sort numerically (Ascending order)
@sorted = sort {$a<=>$b} @unsorted;
 print “After: @sorted\n”;
```

Sorting Arrays

```
# define an array
@unsorted = (0,100,-5,50,300);
 print “Before: @unsorted\n”;
#sort numerically (Descending order)
@sorted = sort {$b<=>$a} @unsorted;
 print “After: @sorted\n”;
```

Merging Arrays

```
#!/usr/bin/perl
@numbers = (1,3,(4,5,6));
print "numbers = @numbers\n";
```

Merging Arrays

```
@odd = (1,3,5);
@even = (2, 4, 6);
@numbers = (@odd, @even);
print "numbers = @numbers\n";
```

Selecting elements from lists

```
#!/usr/bin/perl
$var = (5,4,3,2,1)[4];
print "value of var = $var\n" ;
```

Selecting elements from lists

```
#!/usr/bin/perl
 @list = (5,4,3,2,1)[1..3];
     print "Value of list = @list\n";
```

split function

```
To split the string by pattern
@splitted_array = split(“pattern”,$string_name);

Ex:
#!/usr/bin/perl
  $str=“a:b:c:d:e”;
 @slist = split(‘:’,$str);
  print "Value of list = @slist\n";
Output: Value of list = a b c d e
```

join function

```
To combine the elements of a list into the string
@joined_string = join(“pattern”,@list_name);

Ex:
     #!/usr/bin/perl
  @genes=qw/ns5 ns3 ska lr/;
  $glist = join(‘:’,@genes);
  print "Value of list = $glist\n";
Output: Value of list = ns5:ns3:ska:lr
```

Transform string to Array

```
#!/usr/bin/perl
# define Strings
$var_string = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$var_names = "Larry,David,Roger,Ken,Michael,Tom";
# transform above string into array
@string = split('-', $var_string);
@names = split(',', $var_names);
```

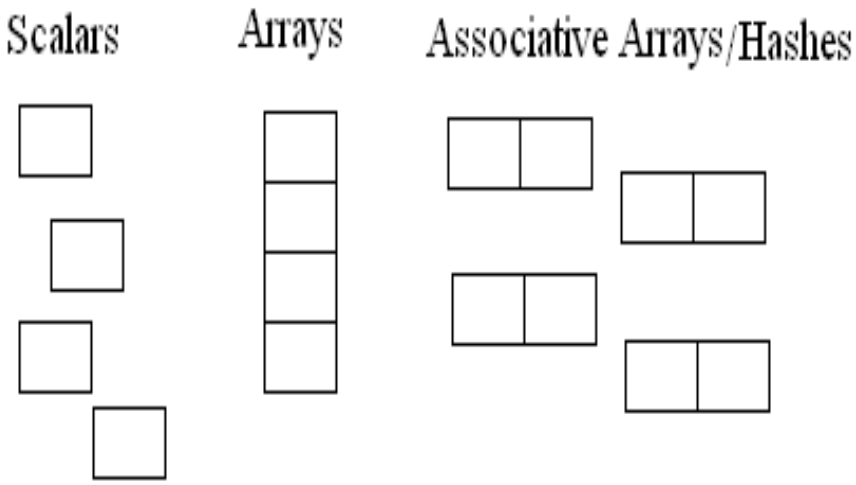
```
print "$string[3]\n"; # This will print Roses

print "$names[4]\n"; # This will print Michael
```

Transform Array to string

```
#!/usr/bin/perl
# define Strings
$var_string = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$var_names = "Larry,David,Roger,Ken,Michael,Tom";
# transform above string into array
 @string = split('-', $var_string);
@names = split(',', $var_names);
$string1 = join( '-', @string );
$string2 = join( ',', @names );
print "$string1\n";
print "$string2\n";

#Rain-Drops-On-Roses-And-Whiskers-On-Kittens
#Larry,David,Roger,Ken,Michael,Tom

#scalar context and list context

@bases = ('A', 'C', 'G', 'T');
print "@bases\n\n";

$scalar = @bases;

print "$scalar\n";

($a,$b,$c,$d) = @bases;
print "$a \n";
print "$b \n";
print "$c \n";
print "$d \n";
exit;
```

```
$scalar = @bases;
Assignment statement assigns an array to a scalar variable
$scalar
scalar context- an array evaluates the size of the array ie,
the number of elements in the array.

($a,$b,$c,$d) = @bases; (assigning an array to another list)
list context- an array evaluates the list of its elements.
```

Perl Data Types



Creating Hashes

```
%data =('John', 45, 'Mary', 35, 'Larry', 25);
%data =('John' => 45, 'Mary' => 35, 'Larry' => 25);
%data =(-JohnPaul => 45, -Mary =>35, -Larry => 25);
Note: if –is used the key must be single word.
```

Creating Hashes

```
%hashdata  = (key1 => value1, key2 => value2, key3 => value3);
```

Accessing Hash Elements

```
print "$data{'John'}\n";
print "$data{'Mary'}\n";
print "$data{'Larry'}\n";
```

Extracting Slices

```
%data =('John', 45, 'Mary', 35, 'Larry', 25);
@array = @data{'John', 'Mary'};
print "@array\n";
```

Extracting Keys and Values

```
%data =('John' =>45, 'Mary' => 35, 'Larry'=> 25);
@names = keys %data;
print "$names[0]\n";
print "$names[1]\n";
print "$names[2]\n";
```

keys function used to list of all keys
o/p: John

Mary
Larry

Extracting Keys and Values

```
%data= ('John' =>45, 'Mary' => 35, 'Larry'=> 25);
@ages = values %data;
print "$ages[0]\n";
print "$ages[1]\n";
print "$ages[2]\n";
```

values function used to list of all values
o/p: 45
35
25

Getting Hah Size

```
%data= =('John' =>45, 'Mary' => 35, 'Larry'=> 25);
@names = keys %data;
$size=@names;
print "Hash size= $size\n";
```

Getting Hash Size

```
%data= =('John' =>45, 'Mary' => 35, 'Larry'=> 25);
@ages = values %data;
$size1=@ages;
print "Hash size= $size1\n";
```

Add and Remove elements in Hashes

```
%data= =('John' =>45, 'Mary' => 35, 'Larry'=> 25);
@keys = keys %data;
$size=@keys;
print "1:no of Key elements = $size\n";
$data{'Alam'}=50; #adding element
@keys = keys %data;
$size=@keys;
```

Add and Remove elements in Hashes

```
print "2: no of Key elements = $size\n";
delete $data{'Alam'}; # delete of element
@keys = keys %data;
$size=@keys;
print "3:no of Key elements = $size\n";
```

Perl Data Types

Associative Array/Hash

Make a list of genes and their functions using Hash

```
%genelist = (
     "grpE", "heat shock protein",
     "thrC", "threonine synthase",
     "dnaQ", "DNA polymerase",
     "mutL", "DNA mismatch repair protein",
     );
```

Associative Array/Hash

```
     %genelist = (
     grpE => "heat shock protein",
     thrC => "threonine synthase",
     dnaQ => "DNA polymerase",
     mutL => "DNA mismatch repair protein",
     );
```

Associative Array/Hash

Note: Hash key must be unique.
Values may not be unique
Using the keys operator to print all the data in %marks :

```
%marks= (
     'John' => 100,
     'Mary' => 90,
     'Kumar'=> 80,
     'Meena'=>70,
     );
print  %marks;
print "\n";
foreach $key (keys %marks) {
     print $key . ":" . $marks{$key}. "\n";
}
%marks= (
     'John' => 100,
     'Mary' => 90,
     'Kumar'=> 80,
     'Meena'=>70,
     );

foreach (keys %marks) {
     print "$_ : $marks{$_} \n";
```

```
}
```

Output:????

```
%marks= (
      'John' => 100,
      'Mary' => 90,
      'Kumar'=> 80,
      'Meena'=>70,
      );
$number= keys(%marks);
print $number;
```

Output:????

```
#Sorting hashes

print "@key \n";
print "@value";

#Sorting hashes using foreach

%marks= (
      'John' => 100,
      'Mary' => 90,
      'Kumar'=> 80,
      'Meena'=>70,
      );

print"\n ======using foreach with sort ===\n";
      foreach $key(sort keys(%marks))
{
      $value = $marks{$key};
      print "key = $key, value = $value\n" ;
}

# A HASH is assigned into an array

%score = (
      Nina=> 300,
      Kamal=> 10,
      Milan=> 200,
      Ella =>500,
);
print %score;
print "\n\n";
@array = %score;
print "@array\n";
```

```
Kamal10Nina300Ella500Milan200

Kamal 10 Nina 300 Ella 500 Milan 200
Press any key to continue . . .
```

Hash operators-values

```
%marks= (
      'John' => 100,
      'Mary' => 90,
      'Kumar'=> 80,
      'Meena'=>70,
      );
@marks1 =values(%marks);
print @marks1 ;
```

```
Kamal10Nina300Ella500Milan200

Kamal 10 Nina 300 Ella 500 Milan 200
Press any key to continue . . .
```

```
%marks= (
      'John' => 100,
      'Mary' => 90,
      'Kumar'=> 80,
      'Meena'=>70,
      );
@value = values(%marks);
foreach $value (@value) {
     print "$value\n";
}
exit;
```

output: ????

```
100
70
90
80
Press any key to continue . . .
```

Hash operator -each

```
%marks= (
      Tarun => 100,
      Divya => 90,
      Ajit=> 80,
      Praveen=>70,
      );
while (($name, $score)  =each(%marks)) {
print "$name:  $score \n";
}
```

```
>perl each.pl
Tarun:  100
Praveen:  70
Ajit:  80
Divya:  90
```

Note: The list of names and marks is printed just like earlier example using the keys operator

Emptying Hash

```
%marks= (
     Tarun => 100,
     Divya => 90,
     Ajit=> 80,
     Praveen=>70,
     );
print "%marks\n";
#empty the hash
%marks = ();
print "%marks\n";
```

Matrices

A matrix is a two-dimensional array,

This matrix contains 12 elements in total, on 4 rows and 3 columns.

The matrix can be assigned like this
`$array[$row][$column] = $variable;`

The first index indicates the row number and the second index the column number

Matrices - program

```
#!/usr/bin/perl 1

 $id[0][0] = "ARATH";
 $id[0][1] = "Arabidopsis thaliana";

 $id[1][0] = "HUMAN";
 $id[1][1] = "Homo sapiens";

 $id[2][0] = "MOUSE"; 6
 $id[2][1] = "Mus musculus";

 $id[3][0] = "RICE"; 8
 $id[3][1] = "Oryza sativa";

#Access matrix
 print "$id[0][0] $id[0][1]";
 print "$id[1][0] $id[1][1]";
 print "$id[2][0] $id[2][1]";
```

Matrices – program - output

The print out will be

```
%>perl matrix.pl
ARATH Arabidopsis thaliana
HUMAN Homo sapiens
MOUSE Mus musculus
```

Matrices – assign value during declaration

```
@matrix = (
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
);
# Change 6, the element at row  1, column 2 to 100
$matrix[1][2] = 100;
```

Input

In order to read input data from the terminal window you can use the diamond operator <> or<STDIN>, which expects a scalar value from the standard input.

Input - chomp

Since a newline character always follows a <> expression the output string will be written on two different lines. To remove the newline character simply use the `chop()` or `chomp()` function

```
#!/usr/bin/perl   -w

print "Enter your age: ";
$age = <>;
chomp($age);
print "Your age is $age !!!\n";
```

which will result in

```
%>perl test.pl

Enter your age: 7
Your age is 7!!!
```

It is also valid to write

```
chop($age = <>);
```

Input - Arrays

It is also possible to read several lines at one time and store them in an array

```
#!/usr/bin/perl -w
print :Enter lines:";
@age = <>;
chomp(@age);
```

Each line will be stored as a separate element in the array.
The elements are indexed from 0 and each element will be a string that ends with a newline character.
After pushing <enter> will not terminate input, but the program will stop and wait for the next line to be read.
In order to terminate, push <ctrl-d> instead.
If you push <enter> without giving any input value, a newline character will be stored on the next index.

Perl Conditional Statement

Conditional statement
1 if statement

Sr.No.	Statement & Description
1	**if statement** An **if statement** consists of a boolean expression followed by one or more statements.
2	**if...else statement** An **if statement** can be followed by an optional **else statement**.
3	**if...elsif...else statement** An **if statement** can be followed by an optional **elsif statement** and then by an optional **else statement**.

Example

```
print "What is your age? ";
my $age = <STDIN>;
if ($age < 6) {
    print "You are before school\n";
} elsif ($age < 18) {
    print "You must go to school\n";
} elsif ($age < 23) {
    print "In most countries you can vote.\n";
} else {
    print "You can drink alcohol in the USA\n";
}
```

2. unless statement

-An unless statement consists of a boolean expression followed by one or more statements.

Syntax

```
unless(boolean_expression){
     # statement(s) will execute if the given condition is
false
}
```

unless statement

```
$a = 20;
 # check the boolean condition using unless statement
  unless( $a < 20 ) {
     # if condition is false then print the following
  print "a is not less than 20\n";
 }
print "value of a is : $a\n";
```

unless..else statement

-An unless statement can be followed by an optional else statement.

Syntax

```
unless(boolean_expression){
# statement(s) will execute if the given condition is false
}else{
# statement(s) will execute if the given condition is true
 }
```

unless..else statement

```
$a = 100;
# check the boolean condition using unless statement
unless( $a == 20 ){
 # if condition is false then print the following
 print "given condition is false\n";
}else{
 # if condition is true then print the following
 print "given condition is true\n";
}
print "value of a is : $a\n";
```

unless..elsif..else statement

- An unless statement can be followed by an optional elsif statement and then by an optional else statement.

unless..elsif..else statement

```
unless(boolean_expression 1){
# Executes when the boolean expression 1 is false
}
elsif( boolean_expression 2) {
# Executes when the boolean expression 2 is true
}
 elsif( boolean_expression 3) {
 # Executes when the boolean expression 3 is true
 }
else {
# Executes when the none of the above condition is met
   }
```

unless..elsif..else statement

```
$a = 20;
# check the boolean condition using if statement unless( $a ==
20 ) {
# if condition is false then print the following print "a has
a value which is not 20\n";
 } elsif( $a == 30 ) {
# if condition is true then print the following
print "a has a value which is 30\n";
}else {
# if none of the above conditions is met
 print "a has a value which is $a\n";
}
```

3. Switch statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

Compare the argument value with multiple data structures, such as array, hash and other patterns.

Ex: compare the scalar value equality with an array, hash or patterns

Example

```
use Switch;
$a=10;
switch($a)
```

```
{
case 10          {print "i am 10\n"; }
case [5..15]    {print "i am in list"}
 else              {print "not true"}
}
```

Output: I am 10

Next statement

Even though the second case construct matches, the switch construct will not check for another construct.
If you want to examine for the another construct, we can use next statement.

Example2

```
use Switch;
$a=10;
switch($a)
{
case 10          {print "i am 10\n"; next;}
case [5..15]   {print "i am in list"}
else              {print "not true"}
}
```

Output:

```
I am 10
I am in list
```

Switch construct with variables

You can examine the case values by the stored variables like array, hash etc.

For that we have to use
  \variable_name with specifier

Steps to install Switch.pm

Open a terminal

Enter CPAN

At the prompt cpan type
   install Switch

After installation, exit the terminal

Example3

```
#!/usr/bin/perl
use 5.010;
use Switch;

$var = 10;
@array = (10,20,30);
%hash = ('key1' => 10, 'key2' => 20);

switch($var)
{
     case 10                 { print "number 100\n" }
     case "a"                { print "String a\n" }
     case [1..10, 42]     { print "number in list\n" }
     case (\@array)      { print "number in array\n" }
     case (\%hash)       { print "number in hash\n" }
     else
}
```

The ? : Operator

used to replace if...else statements.
Exp1 ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

Example

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


$name = "Ali";

$age = 10;


$status = ($age > 60 )? "A senior citizen" : "Not a senior citizen";


print "$name is  - $status\n";
```

This will produce the following result –

```
Ali is - Not a senior citizen
```