

NAME: - SHALMON NATHANEL ANANDAS

CLASS: - M.Sc. PART - II

COURSE: - BIOINFORMATICS

ACADEMIC YEAR: - 2022-2023

ROLL NO.: - 91

PAPER CODE: - GNKPSB|304

COURSE TITLE: - INTRODUCTION TO PERL AND MONGODB

GURU NANAK KHALSA COLLEGE

MATUNGA, MUMBAI-400 019.

DEPARTMENT OF BIOINFORMATICS

CERTIFICATE

This is to certify that **Ms. Shalmon Nathanel Anandas (Roll.No,91)** of M.Sc. Part II Bioinformatics has satisfactorily completed the practical Semester III course prescribed by the University of Mumbai during the academic year 2022-2023.

TEACHER INCHARGE

HEAD OF DEPARTMENT

INDEX:

SR.NO.	EXPERIMENTS	PAGE NO.	DATE	SIGN
1.	Simple programs on Variable types used in Perl		09/07/22	
2.	Conditional statements and Loops		09/07/22	
3.	Operators used on scalar, array and hash variables		20/07/22	
4.	Subroutine		01/08/22	
5.	References and Dereferences, and Scope of variables		01/08/22	
6.	Regular Expressions		17/08/22	
7.	Metacharacters, Quantifiers and Substrings			
8.	Perl Formatting			

Practical 1

Simple Programs on Variable types using Perl

Aim: To understand and write simple Perl programs on variable and data types

Theory:

- Created by Larry Wall in 1987, Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development and more.
- Pros:
 - Good for Quick and complex scripts
 - Parsing & restructuring data
 - High level programming, Networking, Graphical, Database
- Cons:
 - Hardware Drivers
 - Many modules are incomplete
- Basics:
 - Statement must end in ;
 - Comment is #
 - Multiline comment:
 - =begin
 - =cut
 - Naming Scheme is .pl
 - The Perl interpreter ignores whitespaces
 - Single quote [''] the statement in printed as is
 - Double quote [""] the statement operators are executed
- Data types / Variable types
 - Perl has 3 Basic Data types
 - Scalar: Scalars are simple variables. They are preceded by a dollar sign (\$). A scalar is either a number, a string, or a reference. A reference is actually an address of a variable.
 - Array: Arrays are ordered lists of scalars that you access with a numeric index, which starts with 0. They are preceded by an "at" sign (@).
 - Hash: Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%).

Q1. Write a Perl Script to store DNA sequence in Scalar variable entered by user and display an output

CODE:

```
print("Enter your DNA seq: ");  
$seq = <stdin>;  
print("This is the DNA seq you entered: $seq");
```

OUTPUT:

```
Enter your DNA seq: AAA  
This is the DNA seq you entered: AAA
```

Q2. Write a Perl script to ask user to enter RNA sequence using an array without loops

CODE:

```
@sequence;
print("Enter the first RNA sequence: ");
$sequence[0] = <stdin>;
print("Enter the Second RNA sequence: ");
$sequence[1] = <stdin>;
print("Enter the Third RNA sequence: ");
$sequence[2] = <stdin>;
print("Enter the fourth RNA sequence: ");
$sequence[3] = <stdin>;
print("Enter the fifth RNA sequence: ");
$sequence[4] = <stdin>;
print("The sequences you entered are: \n");
print("Sequence 1: $sequence[0]");
print("Sequence 2: $sequence[1]");
print("Sequence 3: $sequence[2]");
print("Sequence 4: $sequence[3]");
print("Sequence 5: $sequence[4]");
```

OUTPUT:

```
Enter the first RNA sequence: AAA
Enter the Second RNA sequence: AUG
Enter the Third RNA sequence: AGG
Enter the fourth RNA sequence: UUG
Enter the fifth RNA sequence: UUU
The sequences you entered are:
Sequence 1: AAA
Sequence 2: AUG
Sequence 3: AGG
Sequence 4: UUG
Sequence 5: UUU
```

Q3. Write a perl script to store codon using hash variables

CODE:

```
%codons = (1 => AUG, 2 => AAA, 3 => UUU, 4 => AGG);
print("codon 1 is $codons{1}\n");
print("codon 2 is $codons{2}\n");
print("codon 3 is $codons{3}\n");
print("codon 4 is $codons{4}\n");
```

OUTPUT:

```
codon 1 is AUG
codon 2 is AAA
codon 3 is UUU
codon 4 is AGG
```

Practical 2

Conditional Statements and loop

Aim: To understand and write code using conditional statements and loops

Theory:

- Perl conditional statements helps in the decision making, which require that the programmer specifies one or more conditions to be evaluated or tested by the program
- Along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- Syntax

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

- A perl statement can be followed by an optional else statement, which execute when the Boolean expression is false
- Syntax

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

- An if statement can be followed by an optional elsif...else statement, which is very useful to test the various conditions using single if...elsif statement

```
if(boolean_expression 1) {  
    # Executes when the boolean expression 1 is true  
} 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 true  
}
```

- A Perl unless statement consists of a Boolean expression followed by one or more statements

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

- The unless statement can then again be followed by an else statement

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

- This unless statement can then also be followed by an elsif 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
}

```

- 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

```

use Switch;

switch(argument) {
case 1 { print "number 1" }
case "a" { print "string a" }
case [1..10,42] { print "number in list" }
case (@array) { print "number in list" }
case /\w+/ { print "pattern" }
case qr/\w+/ { print "pattern" }
case (%hash) { print "entry in hash" }
case (&sub) { print "arg to subroutine" }
else { print "previous case not true" }
}

```

- A loop statement allows us to execute a statement or group of statements multiple times
- While loop statement in perl programming language repeatedly executes a target statement as long as a given condition is true

```

while(condition) {
statement(s);
}

```

- An until loop statement in perl programming language repeatedly executes a target statement as long as a given condition is false

```

until(condition) {
statement(s);
}

```

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times

```

for ( init; condition; increment ) {
statement(s);
}

```

- The foreach loop iterates over a list value and sets the control variable to be each element of the list in turn

```

foreach var (list) {
...
}

```

- A do...while loop is similar to the while loop except that a do while loop is guaranteed to execute at least one time

```

do {
statement(s);
}while( condition );

```

- A loop can be nested inside of another loop, this is known as a nested for loop


```
for ( init; condition; increment ) {  
for ( init; condition; increment ) {  
statement(s);  
}  
statement(s);  
}
```

Q1. Write a perl script to ask user to enter a number and check whether entering number is even or odd

CODE:

```
print("Enter a number: ");  
$num = <stdin>;  
if($num % 2 == 0){  
    print("Number is Even\n");  
}else{  
    print("Number is Odd\n");  
}
```

OUTPUT:

```
Enter a number: 95  
Number is Odd
```

```
Enter a number: 46  
Number is Even
```

Q2. Write a perl script to ask user to enter number to display Fibonacci series

CODE:

```
print("How many digits of fibonacci are to be printed: ");  
$limit = <stdin>;  
$cur_num = 1;  
$prev_num = 0;  
print("$prev_num, $cur_num");  
for($i=1; $i<=($limit-2); $i++){  
    $ans = $cur_num + $prev_num;  
    $prev_num = $cur_num;  
    $cur_num= $ans;  
    print("$ans ");  
}
```

OUTPUT:

```
How many digits of fibonacci are to be printed: 9  
0 1 1 2 3 5 8 13 21
```

Q3. Write a Perl script to ask user to enter a number and check whether entered number is negative or positive

CODE:

```
print("Enter a number: ");
$number = <stdin>;
if($number % 2 == 0){
    print("Number is even\n");
}else{
    print("Number is odd\n");
}
```

OUTPUT:

```
Enter a number: 5
Number is odd
```

```
Enter a number: 6
Number is even
```

Q4. Write a Perl script to ask user to enter RNA sequence using an array with for and foreach loops

CODE:

```
@rna_seq;
for($i=0;$i<5;$i++){
    print("Enter sequence # $i: ");
    $rna_seq[$i] = <stdin>;
}
foreach $seq(@rna_seq){
    print("Sequence: $seq");
}
```

OUTPUT:

```
Enter sequence #0: AAA
Enter sequence #1: AUG
Enter sequence #2: UUU
Enter sequence #3: AGU
Enter sequence #4: GUA
Sequence: AAA
Sequence: AUG
Sequence: UUU
Sequence: AGU
Sequence: GUA
```

Q5. Write a Perl script to ask user to enter DNA sequence using a hash and display keys and values separately

CODE:

```
%dna_seq;
for($i=0;$i<5;$i++){
```

```

$num = $i+1;

print("Enter sequence $num: ");

$dna_seq{$i} = <stdin>;
}
for($i=0; $i<5;$i++){
    $num = $i+1;

    print("Sequence{$num} = $dna_seq{$i}");
}

```

OUTPUT:

```

Enter sequence 1: AATGGGCCCAAATAA
Enter sequence 2: ATGGGGGGUUUUUCC
Enter sequence 3: ATGCCCCCCCCTAAUC
Enter sequence 4: ATGGUCCAATUGGUGG
Enter sequence 5: GTAUUUGATAAAAGAC
Sequence{1} = AATGGGCCCAAATAA
Sequence{2} = ATGGGGGGUUUUUCC
Sequence{3} = ATGCCCCCCCCTAAUC
Sequence{4} = ATGGUCCAATUGGUGG
Sequence{5} = GTAUUUGATAAAAGAC

```

Q6. Write a Perl script to ask user to enter number and find factorial of an entered number

CODE:

```

print("Enter a number: ");

$number = <stdin>;

$ans = 1;

for($i=$number;$i>=1;$i--){
    $ans = $i*$ans;
}

print("Factorial of $number is $ans\n");

```

OUTPUT:

```

Enter a number: 10
Factorial is 3628800

```

Q7. Write a Perl script to store DNA sequence and check entered sequence is DNA or not

```

use Switch;

$seq = <stdin>;

$is_DNA = false;

for($i=0;$i<length($seq)-1;$i++){

```

```

$check = substr($seq, $i, 1);
switch($check){
    case "A"      {$is_DNA = true}
    case "T"      {$is_DNA = true}
    case "G"      {$is_DNA = true}
    case "C"      {$is_DNA = true}
    default : print "It is not a DNA Sequence";
    exit;
}
}
if($is_DNA == true){
    print("It is a DNA sequence");
}

```

OUTPUT:

```

Enter a DNA sequence: ATGGTTCCAAA
It is a DNA sequence

```

```

Enter a DNA sequence: ATUUUGTTTATA
It is not a DNA Sequence

```

Q8. Perl script to display the following pattern

A AAAA

A AAA

A AA

A A

A

CODE:

```

$num = 5;
for($i=0;$i<6;$i++){
    for($j=$num;$j>0;$j--){
        print("A");
    }
    print("\n");
    $num--;
}

```

OUTPUT:

A A A A A
A A A A
A A A
A A
A

Practical 3

Operators used on scalar, array and hash variables

Aim: To understand and write perl programs for operators used on scalar, array and hash variables

Theory:

- An operator can be explain by using the expression $4+5=9$ where 4 and 5 are operands and + is the operator
- Perl support many operator types,
 - Arithmetic Operator
 - Logical Operators
 - Equality Operators
 - Miscellaneous Operators
- Arithmetic operators are:
 - + Addition (Adds elements)
 - - Subtraction (Subtracts elements)
 - * Multiplication (Multiplies elements)
 - / Division (Divides elements and returns quotient)
 - % Modulus (Divides elements and returns remainder)
 - ** Exponent (Gives exponents of a number up to the number specified)
- Equality operators are:
 - == Equal to (Checks if left and right values are equal to each other)
 - <=> Comparison (Checks if value are equal or not and returns -1, 0, 1 depending on whether the left value is less than, equal to, grater than the right value)
 - > Greater than (Checks if left value is greater than the right value)
 - < Less than (Checks if left value is less than the right value)
 - >= Greater than or equal to (Checks if left value is greater than or equal to the right value)
 - <= Less than or equal to (Checks if left value is less than or equal to the right value)
 - The following operators are used for strings
 - lt : Less than
 - gt : Greater than
 - le : Lass than or equal to
 - ge : Greater than or equal to
 - eq : Equal to
 - ne : Not equal to
 - cmp : Compares and returns values -1, 0, 1 depending on which values Is greater or lesser
- Logical operators are:
 - && (and) : IF both operands are true then the condition becomes true
 - || (or) : If one or the other operand is true then the condition becomes true
 - Not : Used to reverse the condition
- Miscellanous operators are:
 - . dot operator (combines 2 strings)
 - x repeat operator (String on the left is repeated the amount of times specified on the right)
 - ++ increment operator (Increases the values of the int by 1)
 - - decrement operator (Decreases the values of the int by 1)
 - -> Arrow operator (Used in dereferencing a method or variable from an object or a class name)

Q1. Write a Perl script accept two number and a string and perform the following operations

- a. Perl Arithmetic Operators
- b. Miscellaneous Operators

CODE:

```
print("Enter 1st Number: ");
$num1 = <stdin>;
print("Enter 2nd Number: ");
$num2 = <stdin>;
print("Enter 1st String: ");
$string1 = <stdin>;
print("Enter 2nd String: ");
$string2 = <stdin>;
print("\nExecuting arithmetic operators...\n");
print("Addition: ");
print($num1+$num2);
print("\n");
print("Subtraction: ");
print($num1-$num2);
print("\n");
print("Division: ");
print($num/$num2);
print("\n");
print("Multiplication: ");
print($num*$num2);
print("\n");
print("Modulo ");
print($num%$num2);
print("\n");
print("Exponent ");
print($num**$num2);
print("\n");
print("\nExecutingmiscellenous...\n");
print("Concatenate: ");
print("$string1.$string2");
print("\n");
```

```
print("Repetition ");
print("$string1"x3);
print("\n");
print("Range ");
print($num1..$num2);
print("\n");
print("Autoincrement(num1): ");
print($num1++);
print("\n");
print("Autodecrement(num1) ");
print($num1--);
print("\n");
```

OUTPUT:

```
Enter 1st Number: 45
Enter 2nd Number: 37
Enter 1st String: Shalmon
Enter 2nd String: Anandas

Executing arithmetic operators...
Addition: 82
Subtraction: 8
Division: 0
Multiplication: 0
Modulo 0
Exponent 0

Executing miscellenous...
Concatenate: Shalmon
.Anandas

Repetition Shalmon
Shalmon
Shalmon

Range
Autoincrement(num1): 45

Autodecrement(num1) 46
```

Q2. Write a perl script to accept three number and display smallest number

CODE:

```
print("Enter #1: ");
$a = <stdin>;
print("Enter #2: ");
$b = <stdin>;
print("Enter #3: ");
$c = <stdin>;

if($a < $b && $a < $c){
    print("Biggest number is $a");
}elseif($b < $a && $b < $c){
    print("Smallest number is $b");
}elseif($c < $a && $c < $b){
    print("Smallest number is $c");
}
```

OUTPUT:

Enter #1: 10	Enter #1: 15	Enter #1: 15
Enter #2: 25	Enter #2: 2	Enter #2: 34
Enter #3: 46	Enter #3: 45	Enter #3: 4
Biggest number is 10	Smallest number is 2	Smallest number is 4

Q3. Write a perl script to enter two string and check wheter its equal or not

CODE:

```
print("Enter 1st String: ");
$string1 = <stdin>;

print("Enter 2nd String: ");
$string2 = <stdin>;

if($string1 eq $string2){
    print("\nStrings are equal\n");
}else{
    print("\nStrings are not equal\n");
}
```

OUTPUT:

```
Enter 1st String: Shalmon
Enter 2nd String: Shalmon

Strings are equal
```

```
Enter 1st String: Shalmon
Enter 2nd String: Anandas

Strings are not equal
```

Q4. Write a Perl script to store elements in an array and perform the following

- Find length of the array
- Add one element at end of an array
- Remove one element at beginning of an array
- Add one element at beginning of an array
- Remove one element at end of an array

CODE:

```
@array = (45,6,3,42,35,22,67,54,23);

print("Array is @array\n");

print("\nLength of the array is $#array\n");

print("\nAdding 25 to the end of the array...\n");
push(@array, 25);
print("Array after adding 25 is @array\n");

print("\nRemoving an element from beginning of the array...\n");
shift(@array);
print("Array after removing element from beginning is @array\n");

print("\nAdding 25 to the beginning of the array...\n");
unshift(@array, 25);
print("Array after adding element to beginning is @array\n");

print("\nRemoving an element from end of the array...\n");
pop(@array);
print("Array after removing element from end of array is @array\n");
```

OUTPUT:

```
Array is 45 6 3 42 35 22 67 54 23

Length of the array is 8

Adding 25 to the end of the array...
Array after adding 25 is 45 6 3 42 35 22 67 54 23 25

Removing an element from beginning of the array...
Array after removing element from beginning is 6 3 42 35 22 67 54 23 25

Adding 25 to the beginning of the array...
Array after adding element to beginning is 25 6 3 42 35 22 67 54 23 25

Removing an element from end of the array...
Array after removing element from end of array is 25 6 3 42 35 22 67 54 23
```

Q5. Write a perl script create elements in an array like ATGCA, ATTG, AATGC, AAAT and perform the following:

1. Find length of an array
2. Add one element i.e., ATGC at bottom of an array
3. Remove one element at beginning of an array
4. Add one element i.e., ATGCC at top of an array
5. Remove one element at end of an array

CODE:

```
@element=("ATGCA","ATTG","AATGC","AAAT");
@elem = qw/ATGCA ATTG AATGC AAAT/;
print "The length of the array is $#element\n";

push(@element,"ATGC"); #add elem at the end
print"\nThe array after adding is
@element\n";

shift(@element);      #remove elem from start
print"\nArray after removing one element from start
@element\n";

unshift(@element, "ATGCC");    #add elem from start
print"\nArray after adding one element from start
@element\n";

pop(@element);          #remove one elem from end
print"\nArray after removing one from start:
```

```
@element\n";
```

OUTPUT:

```
The length of the array is 3

The array after adding is
ATGCA ATTG AATGC AAAT ATGC

Array after removing one element from start
ATTG AATGC AAAT ATGC

Array after adding one element from start
ATGCC ATTG AATGC AAAT ATGC

Array after removing one from start:
ATGCC ATTG AATGC AAAT
```

Q6. Write a perl script to create an array and perform following operations such as merge, reverse and sorting

CODE:

```
@array1= (1,2,3,4,5,7);
@array2= (8,9,11,10,13,12);
@array3= (@array1, @array2);
@chr= qw/a b d s e f g y z/;
print"@array3\n";

@sorted=sort {$a <=> $b} @array3;
print("@sorted\n");

@rev = reverse(@sorted);
print("@rev\n");
```

OUTPUT:

```
1 2 3 4 5 7 8 9 11 10 13 12
1 2 3 4 5 7 8 9 10 11 12 13
13 12 11 10 9 8 7 5 4 3 2 1
```

Q7. Display array in descending order:

CODE:

```
@array1= (1,2,3,4,5,7,8,9,10,11,12);
```

```
Print("@array1\n");  
@sorted=sort {$b <=> $a} @array1;  
print("@sorted\n");
```

OUTPUT:

```
1 2 3 4 5 7 8 9 11 10 13 12  
13 12 11 10 9 8 7 5 4 3 2 1
```

Q8. Write a perl script to store string of an array and display index number 3,4,5 at once

CODE:

```
@strings = ("ATGU", "ATTG", "ATCG", "TGAC", "TCGA", "TACG", "GCTA");  
print(@strings[3..5]);
```

OUTPUT:

```
TGAC TCGA TACG
```

Q9. Write a perl script to demonstrate splice operator

CODE:

```
@arr = (0..9);  
print("Complete array @arr\n");  
  
@replacement = splice(@arr, 3, 4, a..d);  
print("Added Elements @arr\n");
```

OUTPUT:

```
Complete array 0 1 2 3 4 5 6 7 8 9  
Added Elements 0 1 2 a b c d 7 8 9
```

Q10. Write a perl script to sort hashes using keys

CODE:

```
%data = ('b' => 2, 'a' => 1, 'e' => 5, 'd' => 4, 'c' => 3);  
@data_sorted = sort(%data);  
print(@data_sorted);
```

OUTPUT:

```
12345abcde
```

Q11. Write a Perl program to determine the frequency of nucleotide bases in given nucleotide sequence using nested if else

CODE:

```
$seq = <stdin>;
```

```
$a = 0;
$t = 0;
$g = 0;
$c = 0;

for($i=0; $i<length($seq); $i++){
    $n = substr($seq, $i, 1);
    if($n eq "a"){
        $a++;
    }elseif($n eq "t"){
        $t++;
    }elseif($n eq "g"){
        $g++;
    }elseif($n eq "c"){
        $c++;
    }
}

print("a was found $a times\n");
print("t was found $t times\n");
print("g was found $g times\n");
print("c was found $c times\n");
```

OUTPUT:

```
attgggtccaaaaa
a was found 6 times
t was found 4 times
g was found 3 times
c was found 2 times
```

Practical 4

Subroutines

Aim: To understand and write program for perl subroutine

Theory:

- Perl subroutines are functions in which a group of statements that together perform a task
- Code can be divided among various subroutines and each subroutine can perform a specific task which increases the modularity of the code
- The terms subroutine, method and function are used interchangeably in perl
- Subroutine is defined as

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

- This subroutine is then called by doing

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

- Arguments can be passed to a subroutine by using the special array `@_;`
- Arrays hashes can be passed as normal scalar arguments

Q1. Write a perl program to create a subroutine named calculate and find area and perimeter of rectangle

CODE:

```
sub Calculate{  
    print("Give length of the rectangle: ");  
    $len = <stdin>;  
    print("Give breadth of the rectangle: ");  
    $bre = <stdin>;  
    $area = $len * $bre;  
    $peri = 2*($len + $bre);  
    print("Area of the rectangle is $area\n");  
    print("Perimeter of the rectangle is $peri\n");  
}  
Calculate();
```

OUTPUT:

```
Give length of the rectangle: 10  
Give breadth of the rectangle: 6  
Area of the rectangle is 60  
Perimeter of the rectangle is 32
```

Q2. Write a perl program to create a subroutine named calculate and find area and perimeter of a rectangle with parameters

CODE:

```
sub Calculate{  
    my($l, $b) = @_;  
    $area = $l*$b;  
    $peri = 2*$l + 2*$b;  
    print("Area of the Rectangle is $area\n");  
  
    print("Perimeter of the Rectangle is $peri\n");  
}  
  
print("Enter length of the Rectangle: ");  
$l = <stdin>;  
  
print("Enter breadth of the Rectangle: ");  
$b = <stdin>;  
  
Calculate($l,$b);
```

OUTPUT:

```
Enter length of the Rectangle: 10  
Enter breadth of the Rectangle: 8  
Area of the Rectangle is 80  
Perimeter of the Rectangle is 36
```


Practical 5

References and Dereferences, and scope of variables

Aim: To understand and write perl program for references and dereferences and scope of variables

Theory:

- References is a scalar datatype that holds the location of another value which could be scalar, array or hash.
- Because of its scalar nature, areference can be used anywhere a scalar can be used
- References are created by

```
$scalarref = \$foo;  
$arrayref = \@ARGV;  
$hashref = \%ENV;  
$coderef = \&handler;  
$globref = \*foo;
```

- Dereferencing returns the value from a reference point to the location.
- Deferencing is done by simple using \$, @ or % prefix of the reference variable depending on whether the reference is pointing to which datatype
- References to functions are done by using the \& signal handler

```
$cref = \&PrintHash;  
  
&$cref(%hash);
```

- By defaultperl using global variables which means the variables can be accessed from anywhere in the program. But you can create a private variable by using the my keyword

```
sub somefunc {  
    my $variable; # $variable is invisible outside somefunc()  
    my ($another, @an_array, %a_hash); # declaring many variables at once  
}
```

- State variables are variables whose values can be changed once they are initialized

```
state $count = 0;
```

Q1. Write a perl script to accept a number and create reference of scalar variable and display a value using dereferencing

CODE:

```
print("Enter a number: ");  
  
$num = <stdin>;
```

```
$ref_num = \$num;
print("Number entered and stored in Reference is ", ${$ref_num}, "\n");
```

OUTPUT:

```
Enter a number: 23
Number entered and stored in Reference is 23
```

Q2. Write a perl script to store an array and use reference and dereference

CODE:

```
@arr = qw/Biology Zoology Mathematics Physics Chemistry/;
$ref_arr = \@arr;
print("Array entered and stored in Reference is ", @{$ref_arr}, "\n");
```

OUTPUT:

```
Array entered and stored in Reference is BiologyZoologyMathematicsPhysicsChemistry
```

Q3. Write a perl script to store a hash and use reference and dereferencing

CODE:

```
%subjects = (1=>"Biology", 2=>"Zoology", 3=>"Mathematics", 4=>"Physics", 5=>"Chemistry");
$ref_hash = \%subjects;
print("Hash entered and stored in Reference is ", %{$ref_hash}, "\n");
```

OUTPUT:

```
Hash entered and stored in Reference is 3Mathematics1Biology5Chemistry4Physics2Zoology
```

Q4. Write a perl script to create a subroutine and use reference and dereferencing

CODE:

```
sub default{
    print("This is a subroutine\n");
}
$sub_ref = \&default;
print("The reference will be called now\n\n");
&{$sub_ref};
```

OUTPUT:

```
The reference will be called now

This is a subroutine
```

Q5. Write a perl script to store number in global variable and demonstrate the scope of it

CODE:

```
sub g_pr{
```

```
$global_num = 70;

print("Printing global variable inside the subroutine it was declared in: $global_num\n");
}

g_pr();

print("Printing global variable outside the subroutine it was declared in: $global_num\n");
```

OUTPUT:

```
Printing global variable inside the subroutine it was declared in: 70
Printing global variable outside the subroutine it was declared in: 70
```

Q6. Write a perl script to store a number in private variable and demonstrate the scope of it

CODE:

```
sub pr_var{
    my $private_var = 70;
    print("Printing Private variable inside the subroutine it was declared in: $private_var\n");
}

pr_var();

print("Printing private variable outside the subroutine it was declared in: $private_var\n");
```

OUTPUT:

```
Printing Private variable inside the subroutine it was declared in: 70
Printing private variable outside the subroutine it was declared in:
```

Q7. Write a perl script to store a number in state variable and display an OUTPUT

CODE:

```
use feature 'state';

state $static_number = 70;

print("Printing Static variable $static_number\n");

print("Trying to change value of static variable from 70 to 100\n");

$static_number = "shalmon";

print("Printing Static variable after changing $static_number\n");
```

OUTPUT:

```
Printing Static variable 70
Trying to change value of static variable from 70 to 100
Printing Static variable after changing shalmon
```

Practical 6

Regular Expression

Aim: To understand concepts revolving around regular expressions and solve problems related to it.

Theory:

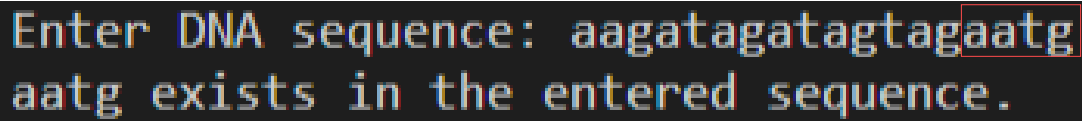
- A regular expression is a string of characters that defines the pattern or patterns you are viewing.
- Perl syntax for regex is similar to that of other programs that support regex
- Regular expression is applied by using a binding operator “=~” and “!~”.
- There are a total of 3 operators withing regex in perl
 - Match operator : m/PATTERN/MODIFIERS
 - Match operator has modifiers that can be stated at the end of the regex. These modifiers are:
 - i – matches case insensitive
 - m – specifies that if the string has newline or carriage return characters
 - o – Evaluates the expression only one
 - s – allows used of . to match newline character
 - x – Allows use of whitespace in the expression
 - g – globally finds all matches
 - cg – continues search even after global match fails
 - Substitute operator: s/PATTERN/REPLACEMENT/MODIFIERS
 - Substitute operator has almost the same modifiers as the math operator. Those are:
 - i – matches case insensitive
 - m – specifies that if the string has newline or carriage return characters
 - o – Evaluates the expression only one
 - s – allows used of . to match newline character
 - x – Allows use of whitespace in the expression
 - g – globally finds all matches
 - e – Evaluates the replacement as if it were a perl statement and uses it sreturn value as the replacement text
 - Transliterate operator: tr/SEARCHLIST/REPLACEMENT/MODIFIERS
 - Transliterate has the least amount of modifiers
 - c – complements SEARCHLIST
 - d – deletes found but unreplaced characters
 - s – squashes duplicate replaced character

Q1. Write a perl script to accept a DNA sequence and match against pattern “aatg” is found on entered sequence and check whether the match case case insensitive

CODE:

```
print("Enter DNA sequence: ")
seq = <stdin>;
if($seq =~ m/aatg/i){
print("aatg exists in the entered sequence.\n");
}else{
print("aatg does not exist in the entered sequence");}
```

OUTPUT:



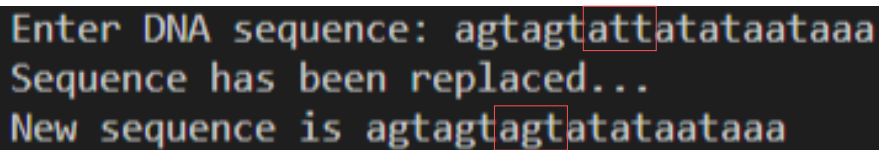
```
Enter DNA sequence: aagatagatagtagaatg
aatg exists in the entered sequence.
```

Q2. Write a perl script to accept DNA sequence from user and search pattern “att” and replace with “agt”

CODE:

```
print("Enter DNA sequence: ");
$seq = <stdin>;
if($seq =~ m/att/){
    $seq =~ s/att/agt/;
print("Sequence has been replaced...\n");
print("New sequence is $seq\n");
}else{
print("att does not exist in sequence");
}
```

OUTPUT:



```
Enter DNA sequence: agtagtattatataataaa
Sequence has been replaced...
New sequence is agtagtagtatataataaa
```

Q3. Write a perl script to accept RNA sequence form user and search pattern “auu” and replace with “agcu” in whose sequence

CODE:

```
print("Enter RNA sequence: ");
$seq = <stdin>;
if($seq =~ m/auu/){
print("Replacing in sequence.....\n");
    $seq =~ s/auu/agcu/g;
print("The new sequence is $seq\n");
}else{
print("The patter 'auu' does not exist in the sequence");
}
```

OUTPUT:

```
Enter RNA sequence: auggucaggagaauugcagauu
Replacing in sequence.....
The new sequence is auggucaggagaagcugcagagcu
```

Q4. Write a perl script to accept RNA sequence and convert that into DNA sequence

CODE:

```
print("Enter an RNA sequence: ");
$seq = <stdin>;

if($seq =~ m/u/){
print("The entered RNA sequence will now be converted into DNA sequence...\n");
    $seq =~ s/u/t/g;
print("The converted sequence is : $seq");
}else{
print("Enter an RNA sequence!!");
}
```

OUTPUT:

```
Enter an RNA sequence: augaugaccaguauaga
The entered RNA sequence will now be converted into DNA
sequence...
The converted sequence is : atgatgaccagtataga
```

Q5. Write a perl script to accept a string and remove duplicate characters from entered string

CODE:

```
print("Enter a string: ");
$string = <stdin>;
$string =~ tr/a-z/a-z/s;
print("Entered string without duplicate letters is: $string\n");
```

OUTPUT:

```
Enter a string: abbdceeeefghhhighssss
Entered string without duplicate letters is: abdcefghighs
```

Practical 7

Metacharacters, Quantifiers and Substring

Aim:

To understand concepts revolving around metacharacters, quantifiers and substring regular expressions and solve problems related to it.

Theory:

For all their power and expressivity, patterns in perl recognize the same 12 traditional metacharacters found in many other regular expression packages: `\ | () [{ & $ * + ? .`

Some simple metacharacters stand by themselves, like `.` and `^` and `$` they don't directly affect anything around them. Some metacharacters work like prefix operators, governing what follows them, like `\`. Others work like postfix operators, governing what immediately precedes them, like `*`, `+`, and `?`. One metacharacter, `|`, acts like an infix operator, standing between the operands it governs. There are even bracketing metacharacters that work like circumfix operators, governing something contained inside them, like `(...)` and `[...]`. Parentheses are particularly important, because they specify the bounds of `|` on the inside, and of `*`, `+`, and `?` on the outside.

If you learn only one of the twelve metacharacters, choose the backslash. That's because backslash disables the other. When a backslash precedes a nonalphanumeric character in a perl pattern, it always makes the next character a literal. If you need to match one of the twelve metacharacters in a pattern literally, you write them with a backslash in front. Thus, `\.` matches a real dot, `\$` a real dollar sign, `\\` a real backslash, and so on. This is known as "escaping" the metacharacter, or "quoting it" or sometimes just "backslashing" it.

Metacharacters:

Symbol	Meaning
<code>\...</code>	De-meta next nonalphanumeric character, meta next alphanumeric character (maybe)
<code>... ...</code>	Alternation (match one or the other)
<code>(...)</code>	Grouping (treat as a unit)
<code>[...]</code>	Character class (Match one character from a set)
<code>^</code>	True at beginning of string (or after newline, maybe)
<code>.</code>	Match one character (Except newline, normally)
<code>\$</code>	True at end of string (or before any newline, maybe)

Quantifiers:

Symbol	Meaning
<code>*</code>	Match 0 or more times (maximal)
<code>+</code>	Match 1 or more times (maximal)
<code>?</code>	Match 1 or 0 times (maximal)
<code>{COUNT}</code>	Match exactly COUNT times
<code>{MIN,}</code>	Match at least MIN times {maximal}

{MIN,MAX}	Match at least MIN but not more than MAX times (maximal)
*?	Match 0 or more times (minimal)
+	Match 1 or more times (minimal)
??	Match 0 or 1 time (minimal)
{MIN,}	Match at least MIN times (minimal)
{MIN, MAX}	Match atleast MIN but not more

Q1. Write a perl program to check that a string contains only a certain set of characters (in this case a-z, A-Z, 0-9)

Code:

```
print("Enter a string: ");
$string = <stdin>;

if($string =~ m/^w/){
    print("its a match\n");
}else{
    print("Its not a match\n");
}
```

Output:

```
Enter a string: Department of Bioinformatics
its a match
```

```
Enter a string: (&($)&#)@&$)@#(&$)@#
Its not a match
```

Q2. Write a perl program that matches a string that has an a followed by zero or more t's

Code:

```
print("Enter a sequence: ");
$seq = <stdin>;

if($seq =~ m/at*/){
    print("There is an a followed by 0 or more t's\n");
}else{
    print("There is not an a followed by 0 or more t's\n");
}
```

Output:


```
Enter a sequence: acgctcgctagactaatcgaga
There is an a followed by 0 or more t's
```

```
Enter a sequence: gctgctcgctcgctcgctcgctcc
There is not an a followed by 0 or more t's
```

Q3. Write a perl program that matches a string that has an a followed by one or more b's

Code:

```
print("Enter a sequence: ");
$string = <stdin>;

if($string =~ m/ab+/){
    print("TRUE\n");
}else{
    print("FALSE\n");
}
```

Output:

```
Enter a sequence: abasydtgabcactbacabbbbaayagtdga
TRUE
```

```
Enter a sequence: atbcgcbcgatagacaca
FALSE
```

Q4. Write a perl program that matches a string that has an 'a' followed by anything, ending in 'b'

Code:

```
print("Enter a string: ");
$string = <stdin>;

if($string =~ m/a/ && $string =~ m/b$/){
    print("TRUE\n");
}else{
    print("FALSE\n");
}
```

Output:

```
Enter a string: Department of Bioinformatics
FALSE
```

```
Enter a string: an exam was taken by jacob
TRUE
```

Q5. Write a perl program that matches a word at the beginning of a string

Code:

```
print("Enter a string that starts with 'Quintessential': ");
$string = <stdin>;

if($string =~ m/^Quintessential/){
    print("You followed the order\n");
}else{
    print("You did not follow the order\n");
}
```

Output:

```
Enter a string that starts with 'Quintessential': Quintessential is a big word
You followed the order
```

```
Enter a string that starts with 'Quintessential': Department of Bioinformatics
You did not follow the order
```

Q6. Write a perl program to check for a number at the end of a string

Code:

```
print("Enter a string with a number at the end: ");
$string = <stdin>;

if($string =~ m/d$/){
    print("You followed the order\n");
}else{
    print("You did not follow the order\n");
}
```

Output:

```
Enter a string with a number at the end: The cost of a pen is Rs.10
You followed the order
```

```
Enter a string with a number at the end: a pen costs 10 rupees
You did not follow the order
```

Q7. Write a Perl program to demonstrate substr() function

Code:

```
$string = "This is Department of bioinformatics";
```

```
$sub_string = substr($string, 5, 10);
```

```
print("$string\n");
```

```
print("$sub_string\n");
```

Output:

```
This is Department of bioinformatics
is Departm
```

Q8. Write a perl program to count number of nucleotides in a sequence using regex

Code:

```
print("Enter a string: ");
```

```
$string = <stdin>;
```

```
print("Length of the string is: ");
```

```
print($string =~ s/\w/\w/g);
```

```
print("\n");
```

Output:

```
Enter a string: Department of Bioinformatics
Length of the string is: 26
```

Q9. Write a perl program accept a string from user and convert into upper case, lower case, and display length

Code:

```
print("Enter a string: ");
```

```
$string = <stdin>;
```

```
print("Lowercase: ");
```

```
$string =~ tr/a-zA-Z/a-z/;
```

```
print($string);
```

```
print("Uppercase: ");
```

```
$string =~ tr/a-zA-Z/A-Z/;
```

```
print($string);
```

```
print("Length: ");
```

```
print($string =~ s/\w/\w/g);
```

```
print("\n");
```

Output:

```
Enter a string: department
Lowercase: department
Uppercase: DEPARTMENT
Length: 10
```

Q10. Write a perl program accept an RNA sequence and display the index number of an entered character

Code:

```
print("Enter an RNA sequence: ");
$string = <stdin>;

print("Which nucleaotide do you want: ");
$char = <stdin>;

$in_string = index($string, $char);

print($in_string);
```

Output:

```
Enter an RNA sequence: AGCU
Which nucleaotide do you want: U
3
```

Q11. Write a perl program to match pattern t or u from an entered sequence

Code:

```
$seq = <stdin>;

if($seq =~ m/t/ || $seq =~ m/u/){
    print("True");
}else{
    print("False");
}
```

Output:

```
Department of Bioinformatics
True
```

```
Attending a lecture
True
```

```
Good morning
False
```

Q12. Write a perl script to split on character “#”

Code:

```
print("Enter a sentence with '#': ");
$string = <stdin>;

@split_string = split("#", $string);

foreach $n(@split_string){
    print("$n\n");
}
```

Output:

```
Enter a sentence with '#': Shalmon#Anandas
Shalmon
Anandas
```

Q13. Write a perl script to split on pattern i.e. whitespaces

Code:

```
print("Enter a sentence: ");
$string = <stdin>;

@split_string = split(" ", $string);

foreach $n(@split_string){
    print("$n\n");
}
```

Output:

```
Enter a sentence: Department of Bioinformatics          Khalsa College
Department
of
Bioinformatics
Khalsa
College
```

Q14. Write a perl script to split on digit and join using single comma

Code:

```
print("Enter a sentence: ");
$string = <stdin>;

@string_split = split(/\d/, $string);

print(@string_split);

$string_join = join(",", @string_split);

print($string_join);
```

Output:

```
Enter a sentence: Rahul60%Aishwarya47%Ruchita88%
Rahul%Aishwarya%Ruchita%
Rahul,,%Aishwarya,,%Ruchita,,%
```

Q15. Write a perl script to demonstrate on splitting on string

Code:

```
print("Enter a sentence: ");
$string = <stdin>;

@split_string = split(" ", $string);

foreach $n(@split_string){
    print("$n\n");
}
```

Output:

```
Enter a sentence: Department of Bioinformatics
Department
of
Bioinformatics
```

Practical 8

Perl Formatting

AIM:

Perl uses a writing template called a 'format' to output reports. To use the format feature of Perl, you have to define a format first and then you can use the format to write formatted data

```
format FormatName =  
fieldline  
value_one, value_two, value_three  
fieldline  
value_one, value_two  
.
```

Here FormatName represent the name of the format. The fieldline is the specific way, the data should be formatted. The values lines represent the values that will be entered into the field line. You end the format with a single period.

Next field can contain any text or fieldholder. The fieldholders hold space for data that will be placed there at a later date. A fieldholder has the format

```
@<<<<<<<<<<  
@>>>>>>>>>>  
@|||||
```

Q1. Write a perl script to display text at center

```
Code:  
format CENTER =  
=====
```

```
@|||||
```

```
$string  
=====
```

```
.  
  
select(STDOUT);  
$~ = CENTER;  
$string = "Department of Bioinformatics";  
write;
```

Output:

```
=====
```

```
Department of Bioinformatics
```

```
=====
```

Q2. Write a perlscript to accept a number and display using perl formatter

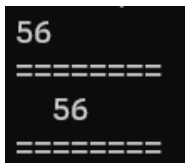
Code:

```
format NUMBER =
=====
@|||||
$number
=====
.

select(STDOUT);
$~ = NUMBER;

$number = <stdin>;
write;
```

Output:



```
56
=====
56
=====
```

Q3. Write a perl script to accept an array and display the values using perl formatter

Code:

```
format ARRAY =
=====

@|||||||||||||||||||||
$n
=====
.

select(STDOUT);
$~ = ARRAY;

@array = qw/mondaytuesdaywednesdaythursdayfridaysaturdaysunday/;
foreach $n(@array){
    $n;
    write;
}
```


Output:

```
=====
monday
=====
tuesday
=====
wednesday
=====
thursday
=====
friday
=====
saturday
=====
sunday
=====
```

Q4. Write a perl script to accept an array and display only keys using right/center formatting

Code:

```
format ARRAY =
=====
@|||
$~
=====
.

select(STDOUT);
$~ = ARRAY;

%hash = ("Rohan" => 1, "Aishwarya" => 2, "Aniket" => 3, "Rama" => 4);

foreach $n(keys %hash){
    $n;
    write;
}
```

Output:

```
=====
                    Rama
=====
                    Rohan
=====
                    Aniket
=====
                    Aishwarya
=====
```

Q5. Write a perl script to accept a decimal number and display only 4 values after decimal using formatting

Code:

```
format DECIMAL =
```

```
=====
```

```
@#####.####
```

```
$number
```

```
=====
```

```
.
```

```
select(STDOUT);
```

```
$~ = DECIMAL;
```

```
$number = <stdin>;
```

```
write;
```

Output:

```
Enter a number: 58.6784
=====
                    58.6784
=====
```

Code:

Output:

Q7. Write a perl script to display name, age and salary by name employee using report formatting

Code:

format EMPLOYEE TOP =

Page Number @<
\$%

Name	Age	Salary
------	-----	--------

•

```
(a)<<<<<<<<<<(a)<<<<<<<<<<(a)<<<<<<<<<<
$name,$age,$salary
=====
```

.

$$\hat{\$} = \text{EMPLOYEE_TOP};$$
$$\$_{\sim} = \text{EMPLOYEE};$$

```
@age = qw/25 27 26 23/;
```

```
@salary = qw/60000 43000 28000 54000/;
```

```
$name = $ _;
```

```
$age = $age[$i];
```

```
$salary = $salary[$i++];
```

```
write;
```

$$\}$$

Output:


```

@age = qw/25 27 26 23/;
@health_issue = qw/Diabetes Covid Scoliosis Cataract/;
@cost_of_treatment = qw/15000 4000 8000 55000/;

foreach (@name){
    $name = $_;
    $age = $age[$i];
    $health_issue = $health_issue[$i];
    $cost_of_treatment = $cost_of_treatment[$i++];
    write;
}

```

Output:

```

=====
Patient Details Page 1
=====
Name           Age           Health Issue       Cost of Treatment
=====
Sumitra        25           Diabetes           15000
=====
Paresh         27           Covid              4000
=====
Vidya          26           Scoliosis          8000
=====
Shalmon        23           Cataract           55000
=====

```

Q9. Write a perl script to diaplysequence_id sequence and alphabet name using report formatting

Code:

```

format SEQUENCE_DETAILS_TOP =

```

```

Sequence Details   Page @<

```

```

    $%

```

ALPHABET NAME

$$\}$$

Output:

Sequence Details Page 1		
SEQ ID	SEQUENCE	ALPHABET NAME
Q59RL7	MSAAKQLFKIVPLTPTEINFLQSLAPVVKEHGVTVTSTMYKYMFTYPEVRSYFNMTNQK	FHP_CANNO
Q9ULY5	MLSQYDEQLAAGDNNGFNKQGNATLYSFDFVDADDFLDSISGALPNNGHNNVNPNTNDIS	CPH2_CANAL
Q9R0Q8	MNSTKSPASHHTERGCFKNSQVLSWTIAGASILFLSGCFITRCVVTYRSSQISGQNLQPH	CLC4E_MOUSE
Q03331	MNSSKSSETQCTERGCFSSQMFLWTVAGIPILFLSACFITRCVVTFRIFQTCDEKKFQLP	CLC4E_HUMAN

Practical 9

OOPS in Perl

Aim:

To understand OOPS concept in perl and write programs to apply the concepts.

Introduction:

As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Class:

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

- **Class name:** The name should begin with a initial letter (capitalized by convention).
- **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword 'use'.
- **Constructors(if any):** Constructors in Perl subroutines returns an object which is an instance of the class. In Perl, the convention is to name the constructor "new".
- **Body:** The class body surrounded by braces, { }.

Object:

It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Perl program creates many objects, which as you know, interact by invoking methods. An object consists of :

- **State :** It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior :** It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity :** It gives a unique name to an object and enables one object to interact with other objects.

Method:

A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods are time savers and help us to reuse the code without retyping the code.

Polymorphism:

Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. This is done by Perl with the help of the signature and declaration of these entities.

Polymorphism in Perl are mainly of 2 types:

- Overloading in Perl
- Overriding in Perl

Inheritance:

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in perl by which one class is allowed to inherit the features(fields and methods) of another class.

Important Terminology:

- **Super Class:** The class whose features are inherited is known as superclass(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.
- Class can be created using packages
 - >use package_name

Encapsulation:

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.
- Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

Abstraction:

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes, etc in the car. This is what abstraction is.

Q1. Write a Perl Script to create class Square and find area of square

Code:

```
package Square;

sub new{
    $class = shift;
    $ref = {};
    bless $ref, $class;
    return $ref;
}
```

```
sub area{  
    my $ans, $side;  
    $side = 43;  
    $ans = $side * $side;  
    print("Area of the square is $ans\n");  
}  
1;
```

```
use Square;  
$obj = new Square();  
$obj -> area();
```

Output:

```
Area of the square is 1849  
Press any key to continue . . .
```

Q2. Write a Perl script to create student class and initialize it with name and roll number by using hash reference. Make methods to Display. It should display all information's of the student

Code:

```
package Student;  
sub new{  
    $class = shift;  
    $ref = {name => shift, roll=>shift};  
  
    bless $ref, $class;  
    return $ref;  
}  
sub display{  
    $ref = shift;  
    $name_of_stu = $ref -> {name};  
    $roll_of_stu = $ref -> {roll};  
    print("Name: $name_of_stu\n");  
    print("Roll number: $roll_of_stu\n");  
}  
1;
```

```
use Student;

$obj = new Student('Shalmon','92');

$obj -> display();
```

Output:

```
Name: Shalmon
Roll number: 92
Press any key to continue . . . _
```

Q3. Write a Perl script to create class Perimeter to calculate perimeter of a square and triangle using inheritance

Code:

```
package Triangle_peri;

sub new{
    $class = shift;
    $ref = {};

    bless $ref, $class;
    return $ref;
}

sub tri_peri{
    $ans, $side1, $side2, $side3;
    $side1=24;
    $side2=53;
    $side3=24;
    $ans = $side1+$side2+$side3;
    print("Perimeter of the triangle is $ans\n");
}

1;
```

```
package Square_peri;

sub new{
    $class = shift;
```

```
        $ref = {};  
        bless $ref, $class;  
        return $ref;  
    }  
    sub sq_peri{  
        $ans, $side;  
        $side=26;  
        $ans = 4*$side;  
        print("Perimeter of the square is $ans\n");  
    }  
1;  
package Perimeter;  
use parent 'Square_peri';  
use parent 'Triangle_peri';  
sub new{  
    $class = shift;  
    $ref = {};  
    bless $ref, $class;  
    return $ref;  
}  
1;
```

```
use Perimeter;  
$obj = new Perimeter();  
$obj->sq_peri;  
$obj->tri_peri;
```

Output:

```
Perimeter of the square is 104  
Perimeter of the triangle is 101  
Press any key to continue . . .
```

Q4. Write a perl program that would print the information (name, year of joining, salary, address) of three employees by creating a class named 'Employee'. The output should be as follows:

Name	Year of joining	Address
Robert	1994	64C- WallsStreat
Sam	2000	68D- WallsStreat
John	1999	26B- WallsStreat

Code:

```
package Employee;
sub new{
    $class = shift;
    $ref = {};
    bless $ref, $class;
    return $ref;
}
sub display{
    @name = qw/Robert Sam John/;
    @yoj = qw/1994 2000 1999/;
    @addr = qw/64C-WallStreet 68C-WallStreet 26C-WallStreet/;
    print("Name          Year of Joining          Address\n");
    for($i=0;$i<=$#name;$i++){
        print("@name[$i]          @yoj[$i]          @addr[$i]\n");
    }
}
1;
```

```
use Employee;
$obj = new Employee();
$obj -> display();
```

Output:

```
Enter #1: 34
Enter #2: 54
Enter #3: 23
The everage of the entered numbers is 95.6666666666667
Press any key to continue . . . _
```

Q6. Write a Perl script create class and display only DNA sequence by applying conditions

Practical 10

DBI in perl

Aim:

To understand how DBI in Perl works and write programs to Apply the concepts

Introduction:

Perl DBI module provides a useful and easy-to-use API that allows you to interact with many of databases including Oracle, SQL Server, MySQL, Sybase, etc. In this tutorial, we are going to show you to interact with the MySQL database.

Accessing a Database in Perl generally takes two steps. The DBI module provides an API for database access. A program uses the functions of DBI to manipulate the database. The second stage of database access from Perl is a database driver (DBD) module. Each different database system requires its own driver. This approach allows a Perl database application program to be relatively independent of the particular database it will access.

Database Independent Interface (DBI):

As the name suggests, DBI provides an independent interface for Perl programs. This means that the Perl code doesn't depend on the database running in the backend. DBI module provides abstraction, i.e, we can write our code without worrying about the database that runs in the back-end.

To import the functions of the Database Independent Interface module, we need to import or include the module with the help of "use" pragma. The use DBI pragma allows us to use DBI module to manipulate the database that we are connecting to.

```
> Use DBI;
```

Connecting to the database:

The connect() method is used to connect to the specified database. It takes three arguments:

- A string of three values separated by a ':' in this example, it is "DBI:mysql:test". The first value specifies that we are using DBI. the second value specifies the database engine, which, in this case, is MySQL. the third value specifies the name of the database that you want to connect to.
- The next argument to the connect() method is the username. In this case, user is 'root'.
- The last argument is the password of your local system. In this example, it is 'password'

```
> my $dbh = DBI->connect ("DBI:mysql:test", "root", "password") or die "Can't connect: " . DBI->errstr();
```

Preparing Queries:

The prepare() method takes in one parameter, the SQL query to be executed. The SQL query is taken in the form of a string that contains the SQL statement. This SQL statement is the same as the SQL statements that you would execute in MySQL. It returns an object called a statement handle that can be used to execute queries.

```
> my $sth = $dbh->prepare( " CREATE TABLE emp( id INT PRIMARY KEY, name VARCHAR(10), salary INT, ");
```

Executing the queries:

The execute() method executes the query written in the prepare() method. It does not take any arguments. It is called using the statement handle object created when the 'prepare' statement is executed.

```
> $sth->execute();
```


Fetching Values from the result:

The fetchrow() method is used to retrieve the next row of data from the result of the executed query. If a select query is executed, then the fetchrow() method fetches the next row from the result. It returns one row from the result which can be assigned to variables. When used in a while loop, we can fetch and display all the rows in the database using the fetchrow() method.

```
> ($id, $name, $salary) = $sth->fetchrow();
```

Q1. Write a Perl script to make a connection with MySQL for any database

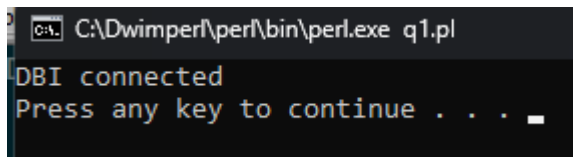
Code:

```
use DBI;

$dbh = DBI->connect("DBI:mysql:database=PRAC10", "root", "root") or die $DBI::errstr;

print("DBI connected\n");
```

Output:



Q2. Write a Perl Script to create table Employee under database Company.

Code:

```
use DBI;

$dbh = DBI->connect("DBI:mysql:database=Company", "root", "root") or die $DBI::errstr;

print("Database Connected\n");

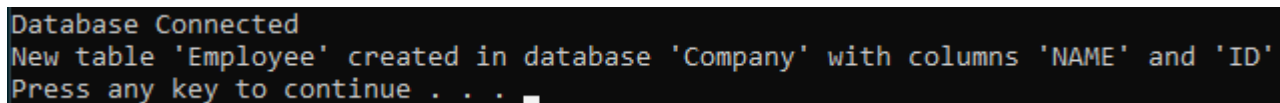
$sth = $dbh->prepare("CREATE TABLE Employee(NAME varchar(50), ID int(6))");

$sth->execute() or die $DBI::errstr;

$sth->finish();

print("New table 'Employee' created in database 'Company' with columns 'NAME' and 'ID'\n");
```

Output:



Q3. Write a Perl Script to insert two values (one without binding Values and one with binding Values respectively).

Code:

```
use DBI;

$dbh = DBI->connect("DBI:mysql:College", "root", "123") or die $DBI::errstr;
```

```

print("Database connected\n");

$sth = $dbh->prepare("insert into STUDENT(NAME, ROLLNO, SUBJECT) values ('Shalmon', '24', 'Bioinformatics')");

$sth->execute() or die DBI::errstr;

$sth->finish();

print("Values inserted into Table without using Binding values\n");

$sth = $dbh->prepare("insert into STUDENT(NAME, ROLLNO, SUBJECT) values (?, ?, ?)");

$sth->execute("Ramesh", 22, "Chemistry");

print("Values inserted into Table using Binding values\n");

```

Output:

```

Database connected
Values inserted into Table without using Binding values
Values inserted into Table using Binding values
Press any key to continue . . .

```

NAME	ROLLNO	SUBJECT
Shalmon	24	Bioinformatics
Ramesh	22	Chemistry

Q4. Write a Perl Script to display all data from employee table.

Code:

```

use DBI;

$dbh = DBI->connect("DBI:mysql:OFFICE", "root", "123") or die $DBI::errstr;

print("Database connected\n");

$sth = $dbh->prepare("select * from EMPLOYEE");

$sth->execute();

while(my @row = $sth->fetchrow()){
    my($first_name, $last_name, $employee_id, $salary) = @row;
    print("
First Name : $first_name
Last Name  : $last_name
Employee ID : $employee_id
Salary     : $salary
\n");
}

$sth->finish();

```

Output:

```
Database connected
First Name : Shalmon
Last Name  : Anandas
Employee ID : 893724
Salary     : 50000

First Name : Liza
Last Name  : Patel
Employee ID : 489302
Salary     : 54000

First Name : Pavithra
Last Name  : Pillai
Employee ID : 267382
Salary     : 53000

First Name : Pradeep
Last Name  : Chavan
Employee ID : 674389
Salary     : 64000

First Name : Kaustubh
Last Name  : Pahurkar
Employee ID : 748930
Salary     : 56000

Press any key to continue . . .
```

Q5. Write a Perl Script to display the employee data who salary greater than entered salary using binding values (ask user to enter).

Code:

```
use DBI;

$dbh = DBI->connect("DBI:mysql:OFFICE", "root", "root") or die $DBI::errstr;

print("Database connected\n");

print("Enter a salary value: ");

$threshold = <stdin>;

$sth = $dbh->prepare("select SALARY from EMPLOYEE where SALARY>$threshold");

$sth->execute();

print("First Name      Last Name      Employee ID  Salary\n");

while(@row = $sth->fetchrow()){
```

```

my($salary) = @row;

$sth2 = $dbh->prepare("select * from EMPLOYEE where SALARY=$salary");

$sth2->execute();

while(@details = $sth2->fetchrow()){

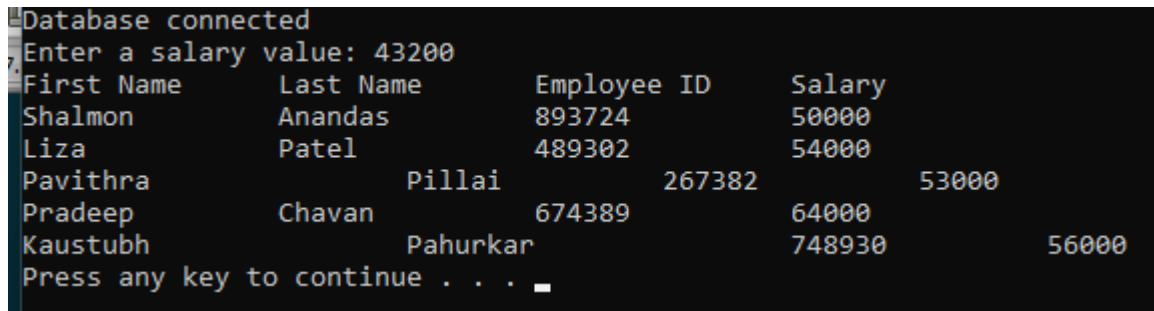
    print("$details[0]          $details[1]          $details[2]          $details[3]\n");

}

}

```

Output:



```

Database connected
Enter a salary value: 43200
First Name      Last Name      Employee ID    Salary
Shalmon         Anandas       893724         50000
Liza            Patel         489302         54000
Pavithra        Pillai        267382         53000
Pradeep         Chavan        674389         64000
Kaustubh        Pahurkar      748930         56000
Press any key to continue . . .

```

Q6. Write a Perl Script to update a record of employee whose employee id is 4 and 5 using one without binding Values and one with binding Values respectively.

Code:

```

use DBI;

$dbh = DBI->connect("DBI:mysql:q6", "root", "root") or die $DBI::errstr;

print("Database connected\n");

# display the table without any changes

$sth1 = $dbh->prepare("select * from details");

$sth1->execute() or die $DBI::errstr;

print("=====\n");

print("Current Table\n");

print("=====\n");

while(my(@row) = $sth1->fetchrow()){

    my(@details) = @row;

    print("@details\n");

}

$sth1->finish();

```

```

# make changes to the table

$sth2 = $dbh->prepare("update details set NAME='Rakesh', DEPT='Marketing' where EMP_ID=4") or die $DBI::errstr;
$sth2->execute() or die $DBI::errstr;
$sth2->finish();


# display table after making changes
$sth2 = $dbh->prepare("select * from details");
$sth2->execute();

print("\n=====\\n");
print("Table after table updated without using binding values\\n");
print("=====\\n");
while(my(@row) = $sth2->fetchrow()){
    my(@details) = @row;
    print("@details\\n");
}
$sth2->finish();


# make changes to the table
$name = "Santosh";
$dept = "Accounts";
$sth3 = $dbh->prepare("update details set NAME=?, DEPT=? where EMP_ID=5") or die $DBI::errstr;
$sth3->execute($name, $dept) or die $DBI::errstr;
$sth3->finish();


# display table after making changes
$sth3 = $dbh->prepare("select * from details");
$sth3->execute();

print("\n=====\\n");
print("Table after table updated using binding values\\n");
print("=====\\n");
while(my(@row) = $sth3->fetchrow()){
    my(@details) = @row;

```

```
print("@details\n");  
}  
$sth3->finish();
```

Output:

```
Database connected  
=====  
Current Table  
=====  
1 Shalmon Administration  
2 Nathanel Administration  
3 Vidya Administration  
4 Rakesh Marketing  
5 Santosh Accounts  
6 Shyam Accounts  
6 Shyam Accounts  
7 Chris Accounts  
8 Robert Accounts  
  
=====  
Table after table updated without using binding values  
$=====  
1 Shalmon Administration  
2 Nathanel Administration  
3 Vidya Administration  
4 Rakesh Marketing  
5 Santosh Accounts  
6 Shyam Accounts  
6 Shyam Accounts  
7 Chris Accounts  
8 Robert Accounts  
  
$=====  
Table after table updated using binding values  
$=====  
1 Shalmon Administration  
2 Nathanel Administration  
3 Vidya Administration  
4 Rakesh Marketing  
5 Santosh Accounts  
6 Shyam Accounts  
6 Shyam Accounts  
7 Chris Accounts  
8 Robert Accounts  
Press any key to continue . . .
```

Q7. Write a Perl Script to delete a record of employee whose employee id is 4 and 5 using one without binding Values and one with binding Values respectively.

Code:

```
use DBI;  
  
$dbh = DBI->connect("DBI:mysql:q6", "root", "root") or die $DBI::errstr;  
print("Database connected\n");
```

```

# display the table without any changes
$sth1 = $dbh->prepare("select * from q7_details");
$sth1->execute() or die $DBI::errstr;
print("=====\n");
print("Current Table\n");
print("=====\n");
while(my(@row) = $sth1->fetchrow()){
    my(@details) = @row;
    print("@details\n");
}
$sth1->finish();

# make changes to the table
$sth2 = $dbh->prepare("delete from q7_details where EMP_ID=4") or die $DBI::errstr;
$sth2->execute() or die $DBI::errstr;
$sth2->finish();

# display table after making changes
$sth2 = $dbh->prepare("select * from q7_details");
$sth2->execute();
print("\n=====\n");
print("Table after table updated without using binding values\n");
print("=====\n");
while(my(@row) = $sth2->fetchrow()){
    my(@details) = @row;
    print("@details\n");
}
$sth2->finish();

# make changes to the table
$sth3 = $dbh->prepare("delete from q7_details where EMP_ID=?") or die $DBI::errstr;
$sth3->execute(5) or die $DBI::errstr;
$sth3->finish();

```

```

# display table after making changes
$sth3 = $dbh->prepare("select * from q7_details");
$sth3->execute();
print("\n=====\\n");
print("Table after table updated using binding values\\n");
print("=====\\n");
while(my(@row) = $sth3->fetchrow()){
    my(@details) = @row;
    print("@details\\n");
}
$sth3->finish();

```

Output:

```

Database connected
=====
Current Table
=====
1 Shalmon Administration
2 Nathanel Administration
3 Vidya Administration
4 Rakesh Marketing
5 Santosh Accounts
6 Shyam Accounts
6 Shyam Accounts
7 Chris Accounts
8 Robert Accounts

=====
Table after table updated without using binding values
=====
1 Shalmon Administration
2 Nathanel Administration
3 Vidya Administration
5 Santosh Accounts
6 Shyam Accounts
6 Shyam Accounts
7 Chris Accounts
8 Robert Accounts

=====
Table after table updated using binding values
=====
1 Shalmon Administration
2 Nathanel Administration
3 Vidya Administration
6 Shyam Accounts
6 Shyam Accounts
7 Chris Accounts
8 Robert Accounts
Press any key to continue . . .

```


Q8. Write a Perl Script to drop employee table.

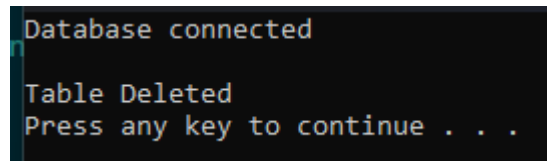
Code:

```
use DBI;

$dbh = DBI->connect("DBI:mysql:q6", "root", "root") or die $DBI::errstr;
print("Database connected\n\n");

$sth = $dbh->prepare("drop table employee");
$sth->execute() or die $DBI::errstr;
$sth->finish();
print("Table Deleted\n");
```

Output:



```
Database connected
n
Table Deleted
Press any key to continue . . .
```