<u>NAME</u>: - SHALMON NATHANEL ANANDAS

<u>CLASS</u>: - M.Sc. PART - II

<u>COURSE</u>: - BIOINFORMATICS

<u>ACADEMIC YEAR</u>: - 2022-2023

<u>ROLL NO</u>.: - 91

<u>PAPER CODE</u>: - GNKPSB|304

<u>COURSE TITLE</u>: - 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:

Practical 1

Simple Programs on Variable types using Perl

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

Theory:

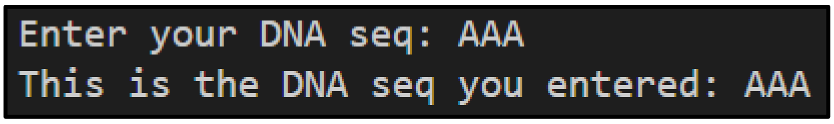
- Created by Larry Wall in 1987, Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development and more.
- Pros:
  - Good for Quick and complex scripts
  - 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:



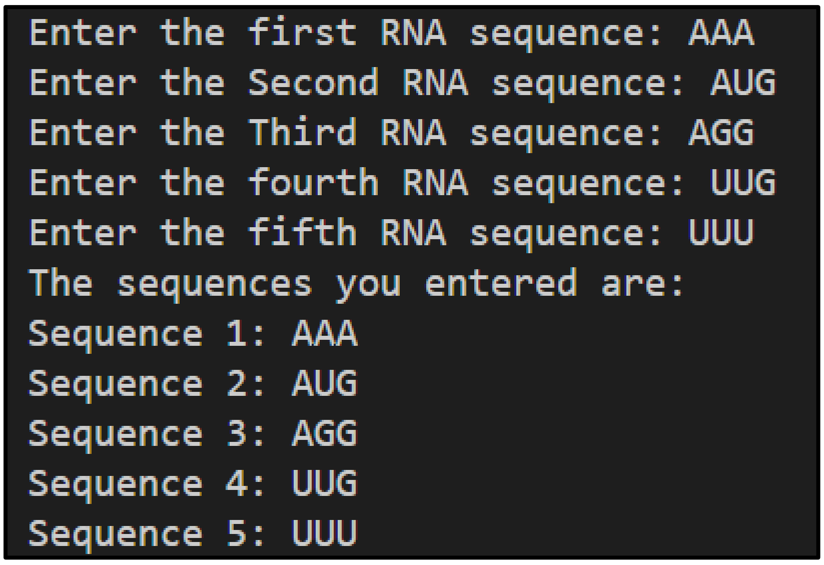
```
Enter your DNA seq: AAA
This is the DNA seq you entered: AAA
```

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:



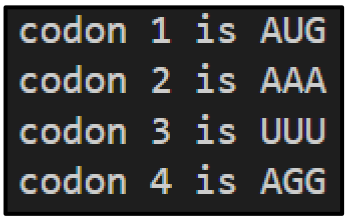
```
Enter the first RNA sequence: AAA
Enter the Second RNA sequence: AUG
Enter the Third RNA sequence: AGG
Enter the fourth RNA sequence: UUG
Enter the fifth RNA sequence: UUU
The sequences you entered are:
Sequence 1: AAA
Sequence 2: AUG
Sequence 3: AGG
Sequence 4: UUG
Sequence 5: UUU
```

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

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 fo 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

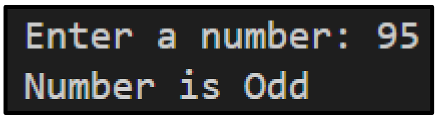
```
for ( init; condition; increment ) {
for ( init; condition; increment ) {
statement(s);
}
statement(s);
}
```

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

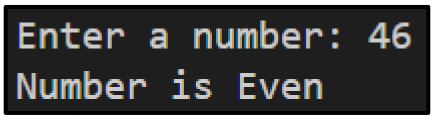
CODE:

```
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
```



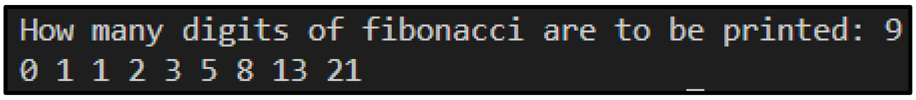
```
Enter a number: 46
Number is Even
```

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:



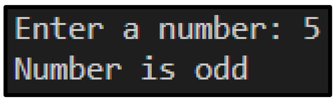
```
How many digits of fibonacci are to be printed: 9
0 1 1 2 3 5 8 13 21
```

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

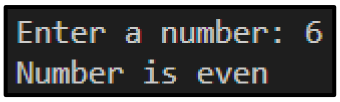
CODE:

```
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
```



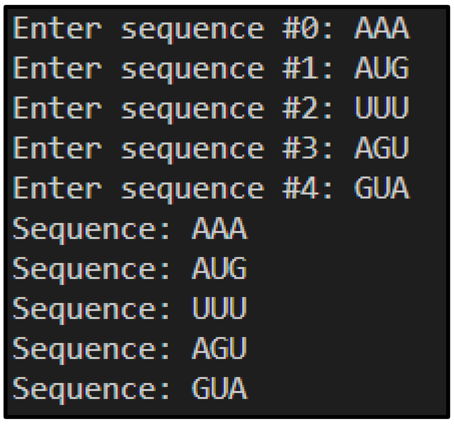
```
Enter a number: 6
Number is even
```

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:



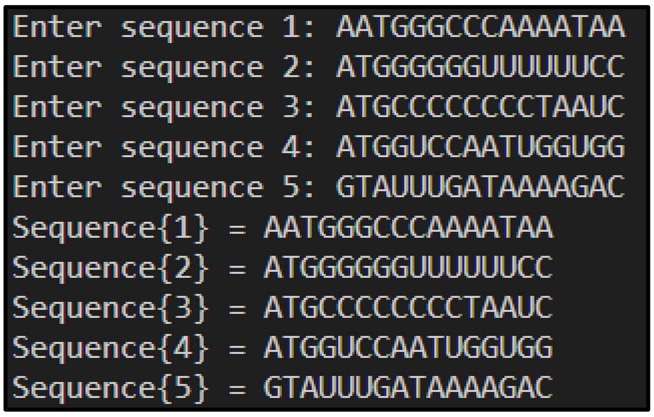
```
Enter sequence #0: AAA
Enter sequence #1: AUG
Enter sequence #2: UUU
Enter sequence #3: AGU
Enter sequence #4: GUA
Sequence: AAA
Sequence: AUG
Sequence: UUU
Sequence: AGU
Sequence: GUA
```

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:



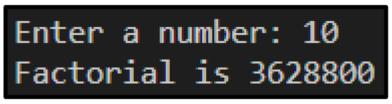
```
Enter sequence 1: AATGGGCCCAAAATAA
Enter sequence 2: ATGGGGGGUUUUUUCC
Enter sequence 3: ATGCCCCCCCCTAAUC
Enter sequence 4: ATGGUCCAATUGGUGG
Enter sequence 5: GTAUUUGATAAAAGAC
Sequence{1} = AATGGGCCCAAAATAA
Sequence{2} = ATGGGGGGUUUUUUCC
Sequence{3} = ATGCCCCCCCCTAAUC
Sequence{4} = ATGGUCCAATUGGUGG
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:



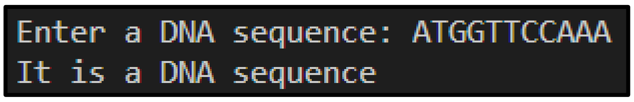
```
Enter a number: 10
Factorial is 3628800
```

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

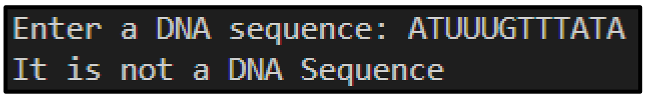
```
use Switch;
```

```
$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

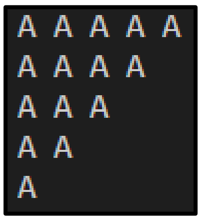
A A

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

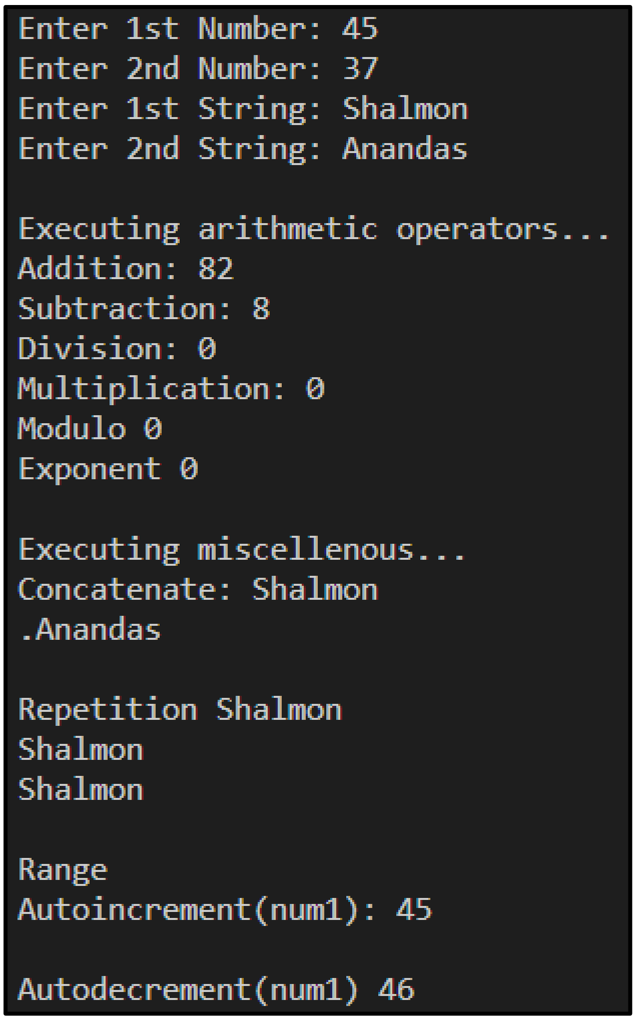
Q1. Write a Perl script accept two number and a string and perform the following operations

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:



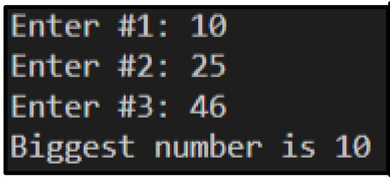
```
Enter 1st Number: 45
Enter 2nd Number: 37
Enter 1st String: Shalmon
Enter 2nd String: Anandas

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
```

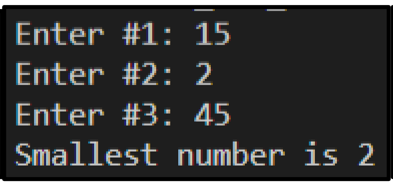
Q2. Write a perl script to accept three number and display smallest number

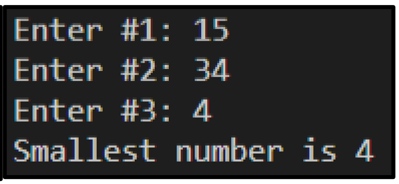
CODE:

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


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

OUTPUT:


```
Enter #1: 10
Enter #2: 25
Enter #3: 46
Biggest number is 10
```


```
Enter #1: 15
Enter #2: 2
Enter #3: 45
Smallest number is 2
```

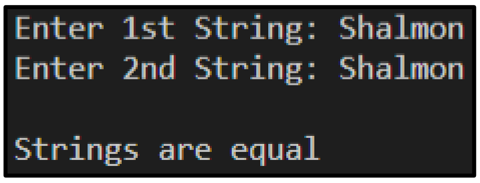

```
Enter #1: 15
Enter #2: 34
Enter #3: 4
Smallest number is 4
```

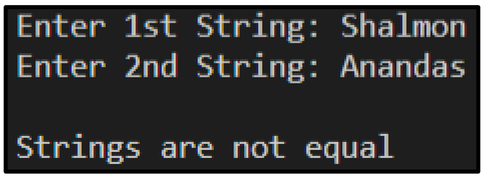
Q3. Write a perl script to enter two string and check wheter its equal or not

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
```

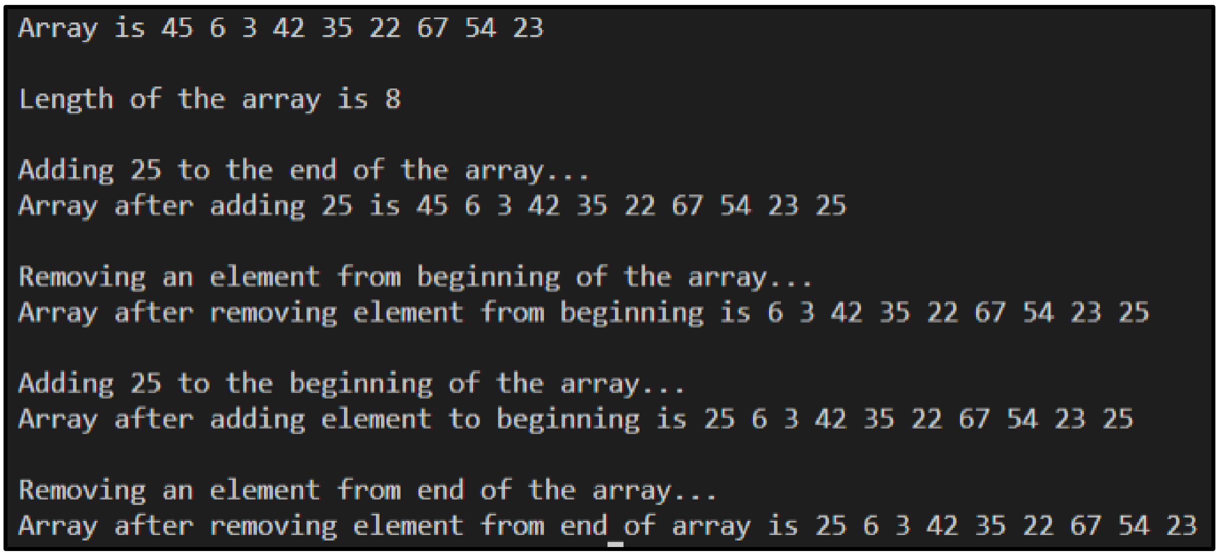
Q4. Write a Perl script to store elements in an array and perform the following

a. Find length of the array
b. Add one element at end of an array
c. Remove one element at beginning of an array
d. Add one element at beginning of an array
e. Remove on 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:

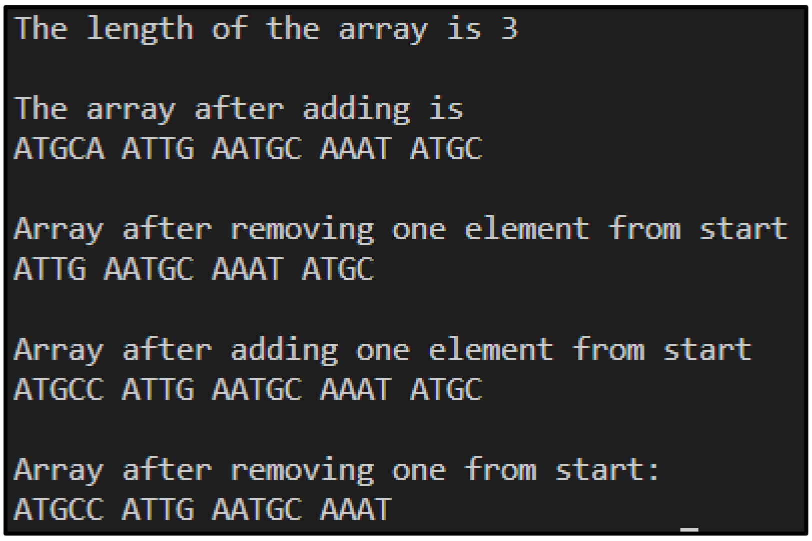
a. Find length of an array
b. Add one element i.e., ATGC at bottom of an array
c. Remove one element at beginning of an array
d. Add one element i.e., ATGCC at top of an array
e. Remove one element at end of an array

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
```

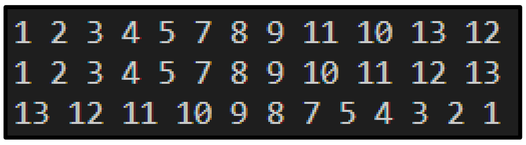
Q6. Write a perl script to create an array and perform following operations such as merge, reverse and sorting

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:



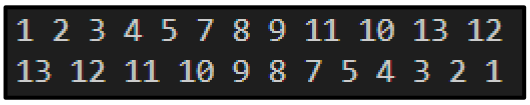
```
1 2 3 4 5 7 8 9 11 10 13 12
1 2 3 4 5 7 8 9 10 11 12 13
13 12 11 10 9 8 7 5 4 3 2 1
```

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:



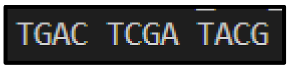
```
1 2 3 4 5 7 8 9 11 10 13 12
13 12 11 10 9 8 7 5 4 3 2 1
```

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
```

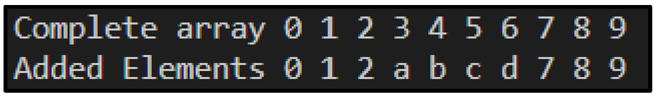
Q9. Write a perl script to demonstrate splice operator

CODE:

```
@arr = (0..9);

print("Complete array @arr\n");
```

```
@replacement = splice(@arr, 3, 4, a..d);

print("Added Elements @arr\n");
```

OUTPUT:



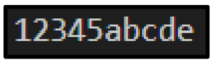
```
Complete array 0 1 2 3 4 5 6 7 8 9
Added Elements 0 1 2 a b c d 7 8 9
```

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
```

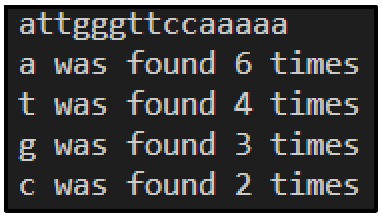
Q11. Write a Perl program to determine the frequency of nucleotide bases in given nucleotide sequence using nested if else

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++;

   }elsif($n eq "t"){

      $t++;

   }elsif($n eq "g"){

      $g++;

   }elsif($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:



```
attgggttccaaaaa
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 );
```

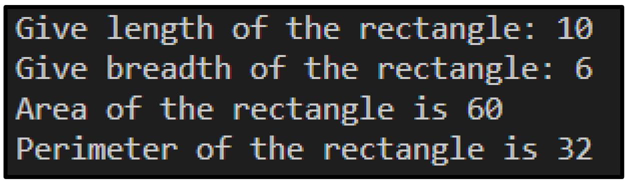
- Arguments can be passed to a subroutine by using the special array @_;
- Arrays hashes can be passed as normal scalar arguments

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:



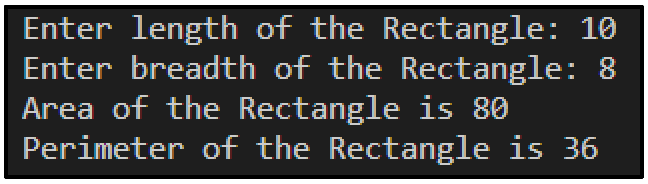
```
Give length of the rectangle: 10
Give breadth of the rectangle: 6
Area of the rectangle is 60
Perimeter of the rectangle is 32
```

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 default perl 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;
```

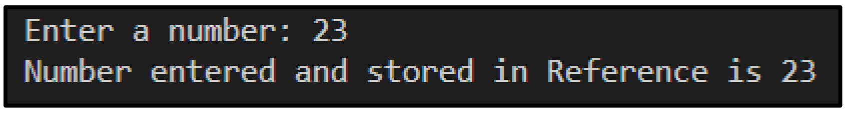
Q1. Write a perl script to accept a number and create reference of scalar variable and display a value using dereferencing

CODE:

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

$num = <stdin>;
```

```
$ref_num = \$num;

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

OUTPUT:



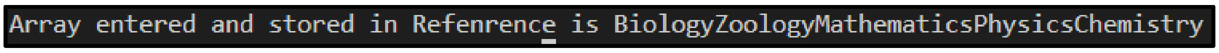
```
Enter a number: 23
Number entered and stored in Reference is 23
```

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 Refenrence is ", @{$ref_arr},"\n");
```

OUTPUT:



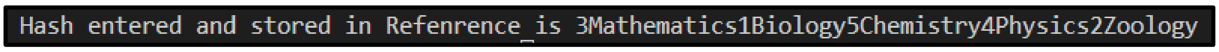
```
Array entered and stored in Refenrence is BiologyZoologyMathematicsPhysicsChemistry
```

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

CODE:

```
%subjects = (1=>"Biology", 2=>"Zoology", 3=>"Mathematics", 4=>"Physics", 5=>"Chemistry");

$ref_hash = \%subjects;

print("Hash entered and stored in Refenrence is ",%{$ref_hash},"\n");
```

OUTPUT:



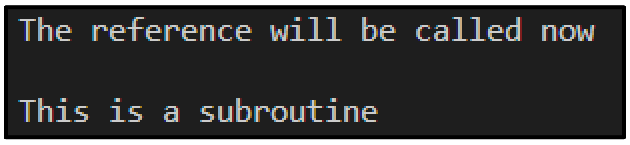
```
Hash entered and stored in Refenrence is 3Mathematics1Biology5Chemistry4Physics2Zoology
```

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:



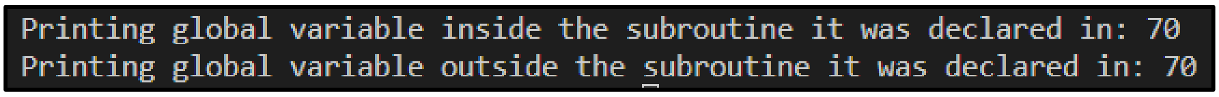
```
The reference will be called now

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:



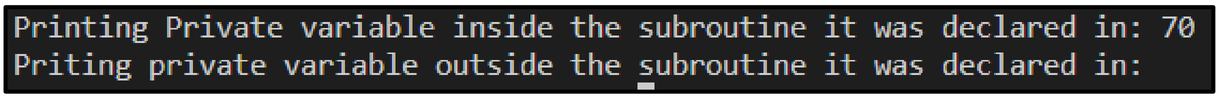
```
Printing global variable inside the subroutine it was declared in: 70
Printing global variable outside the subroutine it was declared in: 70
```

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

OUTPUT:



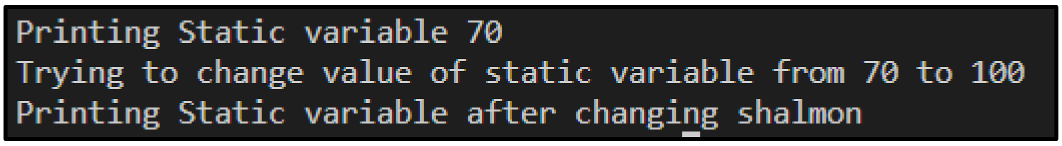
```
Printing Private variable inside the subroutine it was declared in: 70
Priting private variable outside the subroutine it was declared in:
```

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
```