PERL INTRODUCTION

Perl is a programming language which was originally developed for script manipulation. But now Perl is used for a variety of purpose including web development, GUI development, system administration and many more. It is a stable, cross platform programming language.

For web development, Perl CGI is used. CGI is the gateway which interacts with the web browser and Perl in a system.

Its typical use is extracting information from a text file and printing out report for converting a text file into another form. This is because it got its name after the expression, "**P**ractical **E**xtraction and **R**eport **L**anguage".

Programs written in Perl are called **Perl scripts**, whereas system programs to execute Perl scripts are called **Perl program**.

Perl is an interpreted language. When a Perl program run, it is first compiled into a byte code, then it is converted into machine instructions. So writing something in Perl instead of C saves your time.

It supports most of the operating systems and is listed in Oxford English dictionary. Its concepts and syntax is taken from many languages like awk, bourne shell, C, sed and even English.

Perl History

Perl was developed by Larry Wall in 1987 as a scripting language to make report processing easier.

It was first released with version 1.0 on December 18, 1987.

Perl 2, released in 1988 adding a much better regular expression engine.

Perl 3, released in 1989 adding support for binary data streams.

Perl 4, released in 1991 with a better documentation than earlier.

Perl 5, released on October 17, 1994. It added many new features to its last version like objects, variables, references and modules.

The latest version 5.24 is released on May 9, 2016.

Perl Features

- It has a very simple Object-oriented programming syntax.
- It is easily extendible as it supports 25,000 open source modules.
- It supports Unicode.
- It includes powerful tools to process text to make it compatible with mark-up languages like HTML, XML.

- It supports third party database including Oracle, MySQL and many others.
- It is embeddable in other systems such as web servers and database servers.
- It is open source software licensed under GNU.
- Many frameworks are written in Perl.
- It can handle encrypted web data including e-commerce transactions.
- It is a cross platform language.
- It offers a regular expression engine which is able to transform any type of text.

Perl First Example
After perl installation, we'll perform our first perl program.

Click on **Start-> All Programs->SWIN Perl->Padre,** it opens an editor where you can write your script.

Type the following:

1. print "Hello World! with Perl\n";
   Perl statements end with semicolon (;) as shown above. The (\n) is used to denote a new line. As it is a string, it will be enclosed in double quotes (""). And finally 'print' will display it on the screen.

Saving File:

Save the file with (.pl) extension.

Running Script:

To run it, go to **Run-> Run Script** in the above tool bar. Or in short you can also run it by pressing F5.

Output:

Output will be shown in a different command line window displaying the following letters.

1. Hello World! with Perl
   -

Hello World using Perl.

Just to give you a little excitement about Perl, I'm going to give you a small conventional Perl Hello World program, You can try it using Demo link.

```
#!/usr/bin/perl

# This will print "Hello, World"
print "Hello, world\n";
```

Perl Variables

A variable is a place to store values. They can be manipulated throughout the program. When variables are created they reserve some memory space.

There are three types of variables:

- Scalar defined by $
- Arrays defined by @
- Hashes defined by %

The same variable can be used for all these three types of variables in a program. For example, $name, @name and %name, all three variables will be considered different in a program.

Perl Variable Declaration

The equal sign (=) is used to assign values to variables. At the left of (=) is the variable name and on the right it is the value of the variable.

```
1. $name = "Anastasia";
2. $rank = "9th";
3. $marks = 756.5;
```

Here we have created three variables $name, $rank and $marks.

Perl use strict

If you are using **use strict** statement in a program, then you have to declare your variable before using it. It is mandatory. Otherwise you'll get an error.

The $a and $b are special variables used in perl sort function. There is no need to declare these two variables. So it is recommended not to use these two variables except in connection to **sort.**

Variables can be declared using **my, our, use vars, state and $person::name** (explicit package name). Although, they all have different meanings.

```
1. use 5.010;
2. use strict;
3. my $x = 23;
4. say $x;
5. state $name = "Anastasia";
6. say $name;
7. our $rank = "9th";
8. say $rank;
9. use vars qw($marks);
```

```
$marks = 756.5;
say $marks;
$Person::name = 'John';
say $Person::name;
$a = 1224365;
say $a;
 $b = "Welcome at JavaTpoint";
say $b;
```

Output:

```
23
Anastasia
9th
756.5
John
1224365
Welcome at JavaTpoint
```

Look at the above output, the last two variable are $a and $b. so we have not defined them. Yet their output is displayed.

Perl Scalars

A scalar contains a single unit of data. It is preceded with a ($) sign followed by letters, numbers and underscores.

A scalar can contain anything a number, floating point number, a character or a string.

We can define scalars in two ways. First, we can declare and assign value together. Second, we will first declare and then assign value to the scalar.

In the following example, we'll show both the methods to define scalars.

Example:

```
use strict;
use warnings;
use 5.010;
#Declairing and assigning value together
my $color = "Red";
say $color;
#Declairing the variable first and then assigning value
my $city;
$city = "Delhi";
say $city;
```

Output:

```
Red
Delhi
```

Perl Data Types

Perl language is a loosely typed language, the Perl interpreter itself chooses the data type. Hence, there is no need to specify data type in Perl programming.

There are basically three data types in Perl:

- **Scalars**: Perl scalars are a single data item. They are simple variables, preceded by a ($) sign. A scalar can be a number, a reference (address of a variable) or a string.
- **Arrays**: Perl arrays are an ordered list of scalars. They are preceded by (@) sign and accessed by their index number which starts with 0.
- **Hashes**: Perl hashes are an unordered collection of key-value pairs. They are preceded by (%) sign and accessed using keys.

Perl Literals

In Perl there are two different types of scalar constants:

1. Numeric literal
2. String literal

Perl Numeric Literals

Perl numeric literals are numbers. Perl stores number internally as either signed integers or floating-point values.

Perl numeric literals can be assigned following types of formats:

Number	Type
526	Integer
5.5	Floating point
5e10	Scientific notation
5.5E	Scientific notation
5_549_63	A large number
010101	Binary number
0175	Octal number
AF0230	Hexadecimal number

Look at the above table,

- Integers are a group of consecutive digits.
- Floating-point numbers contain a decimal in between. A number containing '0' value on the right hand side of the number (234.00) is also a floating point number .
- A number containing an exponent notation (e or E) is the scientific notation.
- Commas are not allowed into a numeric literal but you can use underscores (_) instead of commas. Perl will remove underscores while using this value.
- Combination of 0 and 1 represents a binary number with base 2.
- Number with a leading 0 comes in the category of octal numbers with base 8.
- Number containing alphabets (a, b, c, d, e, f) are hexadecimal numbers with base 16.

Perl String Literals

Perl string literals contain an empty string, ASCII text, ASCII with high bits or binary data. There is no limit in a string to contain data. They are surrounded by either a single quote (′) or double quote (″).

Variable interpolation is allowed in double quote string but not in single quote string. Also special characters preceding with backslash (\) are supported by double quote strings only.

Escape Characters in string literals

Characters	Purpose
\n	Denotes newline
\r	Denotes carriage return
\ t	Denotes horizontal tab
\v	Denotes vertical tab
\Q	Backslash following all non-alphanumeric character
\a	Denotes alert
\f	Denotes form feed
\b	Denotes backspace
\u	Change next character to uppercase

\U	change all following characters to uppercase
\l	Change next character to lowercase
\L	Change all following character to lowercase
\E	Denotes \U, \L, \Q
\cX	Controls characters, X is a variable
\0nn	Create octal formatted numbers
\xnn	Create hexadecimal formatted numbers
\\	Denote backslash
\'	Denote single quote
\"	Denote double quote

Perl if-else Statement

The if statement in Perl language is used to perform operation on the basis of condition. By using if-else statement, you can perform operation either condition is true or false. Perl supports various types of if statements:

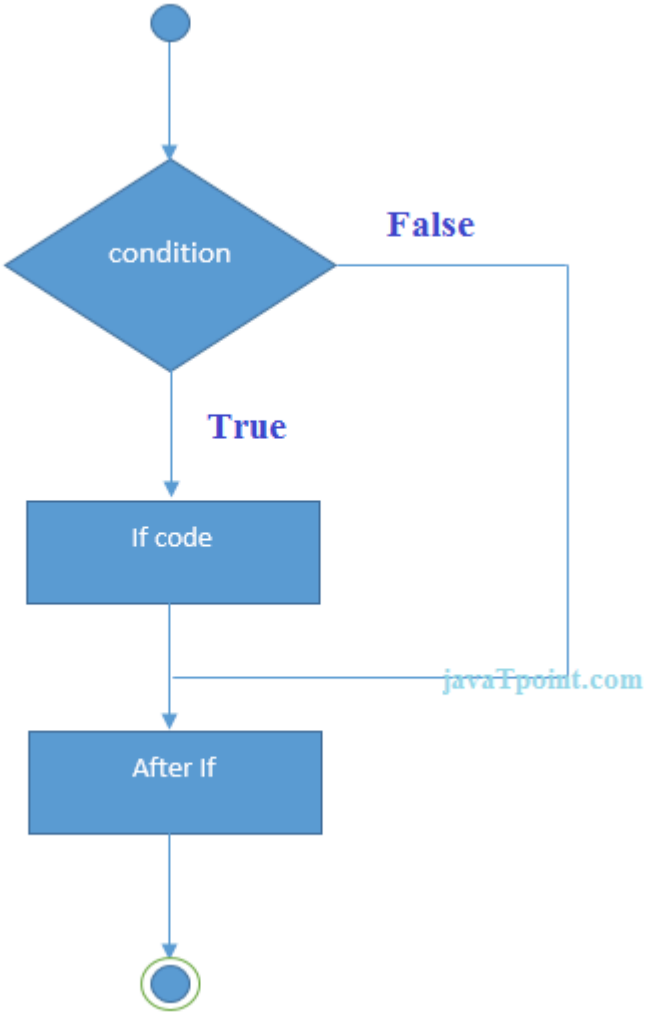
- If
- If-else
- If else-if

Perl If Example

The Perl single if statement is used to execute the code if condition is true. The syntax of if statement is given below:

```
1. if(expression){
2. //code to be executed
3. }
```

Flowchart of if statement in Perl



Let's see a simple example of Perl language if statement.

```
1. $a = 10;
2. if( $a %2==0 ){
3.     printf "Even Number\n";
4. }
5. }
```

Output:

```
Even Number
```

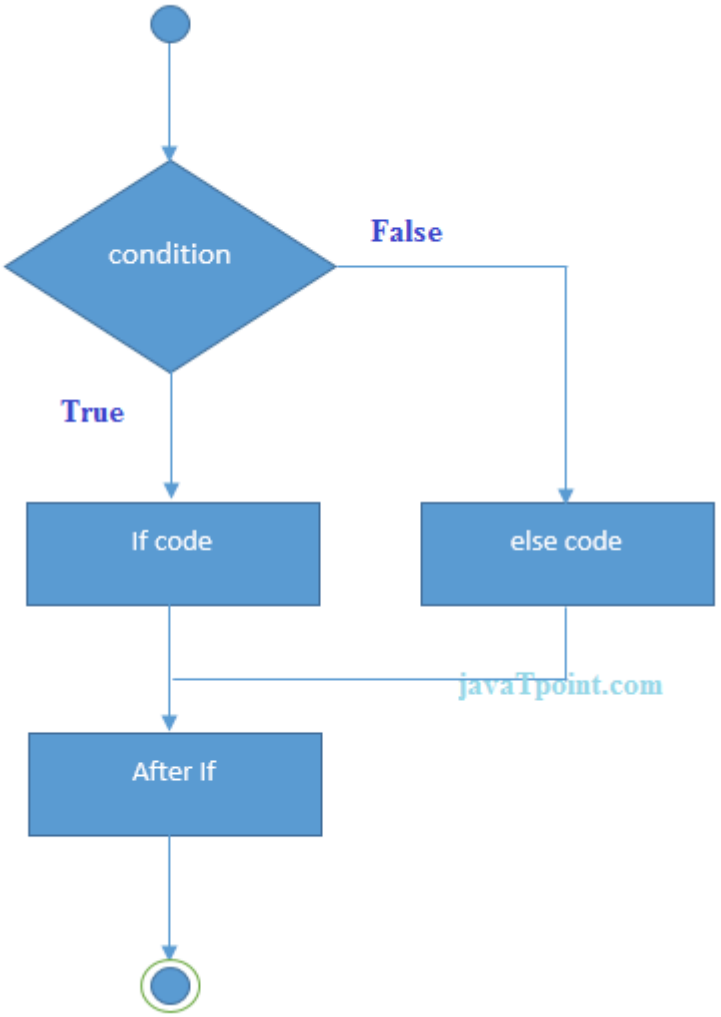
Here, output is even number as we have given input as 10.

Perl If-else Example

The Perl if-else statement is used to execute a code if condition is true or false. The syntax of if-else statement is given below:

```
1. if(expression){
2. //code to be executed if condition is true
3. }else{
4. //code to be executed if condition is false
5. }
```

Flowchart of if-else statement in Perl



Let's see the simple example of even and odd number using if-else statement in Perl language.

```
1. $a = 10;
2. if( $a %2==0 ){
3.     printf "Even Number\n";
4. }else{
5.     printf "Odd Number\n";
6. }
```

Output:

```
Even Number
```

Here, input is an even number and hence output is even.

Perl If-else

In this example, we'll take input from user by using standard input (<STDIN>/<>).

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

Output:

```
Enter a Number?
5
Odd Number
Enter a Number?
4
Even Number
```

In the first output, user has entered number 5 which is odd. Hence the output is odd.

In the second output, user has entered number 4 which is even. Hence the output is even.

Perl If else-if Example

The Perl if else-if statement executes one code from multiple conditions. The syntax of if else-if statement is given below:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Flowchart of if else-if statement in Perl

The example of if else-if statement in Perl language is given below.

```
1.  print "Enter a Number to check grade\n";
2.  $num = <>;
3.  if( $num < 0 || $num > 100){
4.     printf "Wrong Number\n";
5.  }elsif($num >= 0 && $num < 50){
6.     printf "Fail\n";
7.  }elsif($num >= 0 && $num < 60){
8.     printf "D Grade\n";
9.  }elsif($num >= 60 && $num < 70){
10.    printf "C Grade\n";
11. }elsif($num >= 70 && $num < 80){
12.    printf "B Grade\n";
13. }elsif($num >= 80 && $num < 90){
14.    printf "A Grade\n";
15. }elsif($num >= 90 && $num <= 100){
16.    printf "A+ Grade\n";
17. }
```

Output:

```
Enter a Number to check grade
66
C Grade
Enter a Number to check grade
2
Wrong Number
```

Perl Switch Statement

The perl switch statement is used to execute the code from multiple conditions. There is no case or switch statement in perl. Instead we use 'when' in place of case and 'given' in place of case.

The syntax of switch statement in perl language is given below:

```
given(expression){
when (value1)
{//code to be executed;}
when (value2)
{//code to be executed;}
when (value3)
{//code to be executed;}
default
{//code to be executed if all the cases are not matched.}
}
```

Perl Switch Example

Let's see a simple example of c language switch statement.

```
use feature qw(switch say);
print "Enter a Number\n ";
chomp( my $grade = <> );
given ($grade) {
   when ('10') { say 'It is 10' ;}
   when ('20') { say 'It is 20' ;}
   when ('30') { say 'It is 30' ;}
   default { say 'Not 10, 20 or 30';}
}
```

Output:

```
Enter a Number
10
It is 10
Enter a Number
55
'Not 10, 20 or 30'
```

Perl for Loop

Perl for loop is also known as **C-style for loop. The for loop iterates the statement or a part of the program several times.**

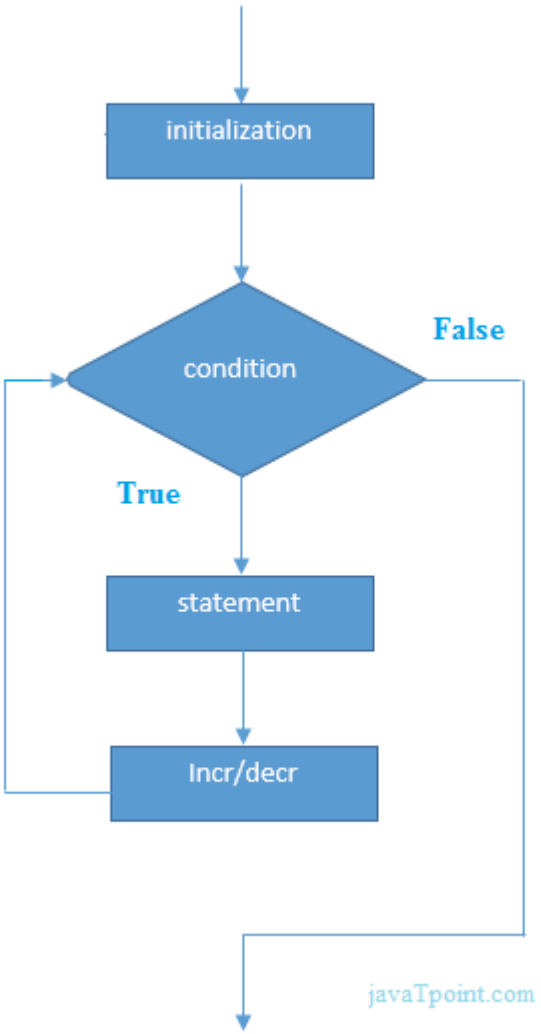
It has three parameters:

- **Initialize :** This part is executed first, and only once. It initializes and declares loop variable.
- **Condition :** The for loop executes till the condition is true. When condition is false, loop execution stops and execution terminates out of the loop.
- **Increment/Decrement :** The for loop variable increment or decrement as long as it satisfies the loop condition. When condition is not satisfied loop terminates and output is printed.

The syntax of for loop in Perl language is given below:

```
for(initialization;condition;incr/decr){
//code to be executed
}
```

Flowchart of for loop in Perl



Perl for loop Example:

Let's see the simple program of for loop that prints table of 1.

```
1. for( $a = 1; $a <= 10; $a++ ){
2.     print " $a\n";
3. }
```

Output:

```
1
2
3
4
5
6
7
8
9
```

```
10
```

Perl while Loop

The Perl **while loop** is used to iterate the part of program or statements many times.

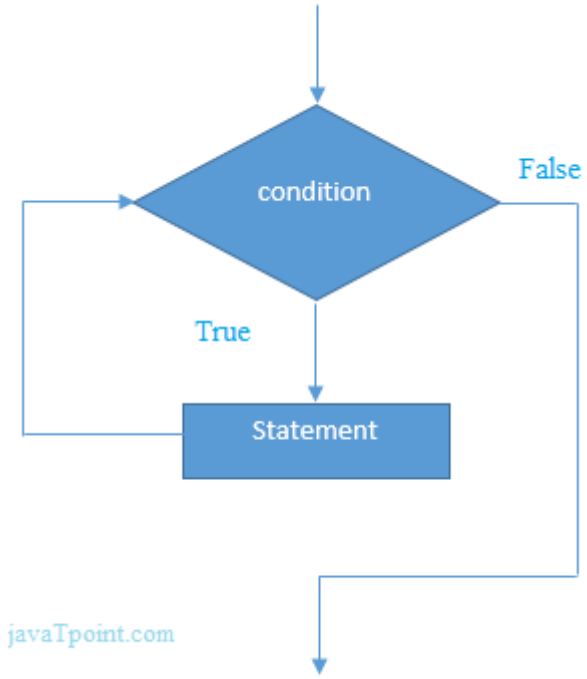
In while loop, condition is given before the statement. When loop execution starts, it first checks whether the condition is true or false. If condition is true,loop executes. If condition is false, the loop terminates out of the loop.

Syntax of while loop in C language

The syntax of while loop in Perl language is given below:

```
1. while(condition){
2. //code to be executed
3. }
```

Flowchart of while loop in Perl



Perl while loop Example

```
1. $i = 1;
2. # while loop execution
3. while( $i <= 10 ){
4.    printf "$i\n";
```

```
5.    $i++;
6. }
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Perl do while Loop

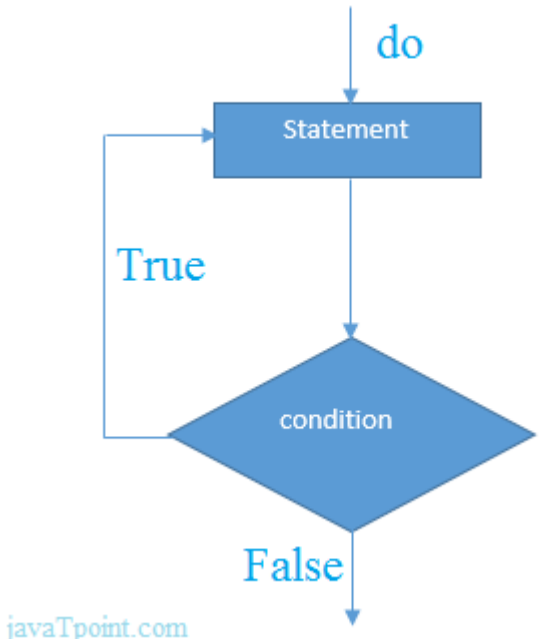
Unlike for and while loop, the do while loop checks its condition at the bottom of the loop. So do while loop will execute at least once.

do while loop syntax

The syntax of Perl language do-while loop is given below:

```
1. do{
2. //code to be executed
3. }while(condition);
```

Flowchart of do while loop



Perl do while Example

This is the simple program of Perl do while loop where we are printing the table of 1.

```
$i = 1;
# do...while loop execution
do{
   printf "$i\n";
   $i++;
}while( $i <= 10 );
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Perl Functions and Subroutines

Perl functions and subroutines are used to reuse a code in a program. You can use a function at several places in your application with different parameters.

There is only one difference in function and subroutine, subroutine is created with **sub** keyword and it returns a value. You can divide your code into separate subroutines. Logically each function in each division should perform a specific task.

Syntax of subroutine:

```
sub subName{
body
}
```

Perl define and call subroutine function

The syntax for Perl define a subroutine function is given below:

```
sub subName{
body
}
   OR
subName(list of arguments);
&subName(list of arguments);
```

In the following example, we are defining a subroutine function 'myOffice' and call it.

```
#defining function
sub myOffice{
   print "javaTpoint!\n";
}
# calling Function
myOffice();
```

Output:

```
javaTpoint!
```

Perl subroutine Function with Arguments

You can pass any number of arguments inside a subroutine. Parameters are passed as a list in the special @_ list array variables. Hence, the first argument to the function will be $_[0], second will be $_[1] and so on.

In this example, we are calculating perimeter of a square by passing a single parameter.

```
$squarePerimeter = perimeter(25);
print("This is the perimter of a square with dimension 25: $squarePerimeter\n");
sub perimeter {
   $dimension = $_[0];
   return(4 * $dimension);
 }
```

Output:

```
100
```

What are Packages?

The **package** statement switches the current naming context to a specified namespace (symbol table). Thus −

- A package is a collection of code which lives in its own namespace.
- A namespace is a named collection of unique variable names (also called a symbol table).
- Namespaces prevent variable name collisions between packages.
- Packages enable the construction of modules which, when used, won't clobber variables and functions outside of the modules's own namespace.
- The package stays in effect until either another package statement is invoked, or until the end of the current block or file.
- You can explicitly refer to variables within a package using the **::** package qualifier.

Following is an example having main and Foo packages in a file. Here special variable __PACKAGE__ has been used to print the package name.

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

```
# This is main package
$i = 1;
print "Package name : " , __PACKAGE__ , " $i\n";

package Foo;
# This is Foo package
$i = 10;
print "Package name : " , __PACKAGE__ , " $i\n";

package main;
# This is again main package
$i = 100;
print "Package name : " , __PACKAGE__ , " $i\n";
print "Package name : " , __PACKAGE__ ,  " $Foo::i\n";

1;
```

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

Package name : main 1
Package name : Foo 10
Package name : main 100
Package name : main 10

BEGIN and END Blocks

You may define any number of code blocks named BEGIN and END, which act as constructors and destructors respectively.

BEGIN { ... }
END { ... }
BEGIN { ... }
END { ... }

- Every **BEGIN** block is executed after the perl script is loaded and compiled but before any other statement is executed.
- Every END block is executed just before the perl interpreter exits.
- The BEGIN and END blocks are particularly useful when creating Perl modules.

Following example shows its usage −

Live Demo

```
#!/usr/bin/perl

package Foo;
print "Begin and Block Demo\n";
```

```
BEGIN {
   print "This is BEGIN Block\n"
}

END {
   print "This is END Block\n"
}

1;
```

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

This is BEGIN Block
Begin and Block Demo
This is END Block

What are Perl Modules?

A Perl module is a reusable package defined in a library file whose name is the same as the name of the package with a .pm as extension.

A Perl module file called **Foo.pm** might contain statements like this.

```
#!/usr/bin/perl

package Foo;
sub bar {
   print "Hello $_[0]\n"
}

sub blat {
   print "World $_[0]\n"
}
1;
```

Few important points about Perl modules

- The functions **require** and **use** will load a module.
- Both use the list of search paths in **@INC** to find the module.
- Both functions **require** and **use** call the **eval** function to process the code.
- The **1;** at the bottom causes eval to evaluate to TRUE (and thus not fail).