**What is Perl Modules and Packages**

Modules and Packages are closely related to each other and are independent. Package: A Perl package is also known as namespace and which have all unique variables used like hashes, arrays, scalars, and subroutines. Module: Module is a collection of reusable code, where we write subroutines in it. These modules can be loaded in Perl programs to make use of the subroutines written in those modules.

**What are Perl Modules?**

Standard modules will get installed during installation of Perl on any system. CPAN: Comprehensive Perl Archive Network – A global repository of Perl modules. Our own customized Perl Modules which can be written by us. Basically, A module when loaded in any script will export all its global variables and subroutines. These subroutines can directly call as if they were declared in the script itself. Perl Modules can be written with .pm extension to the filename Ex : Foo.pm. A module can be written by using 'package Foo' at the beginning of the program.

**Basic Perl module:**

#!/usr/bin/perl

package Arithmetic;

sub add

{

my $a=$\_[0];

my $b=$\_[1];

return ($a+$b);

}

sub subtract

{

my $a=$\_[0];

my $b=$\_[1];

return ($a-$b);

}

1;

No Output

To use this Perl module, we have to place it in currently working directory.

We can load a Perl module using require or use anywhere in the code. The major difference between require and use is, require loaded module during runtime and use loads during compile time.

#!/usr/bin/perl

require

Arithmetic;

print Arithmetic::add(5,6);

print Arithmetic:: subtract (5,6);

Here, in the above example, we are accessing the subroutines using fully qualified module name.

We can also access the package using 'use Arithmetic'.

**Exporter:**

This module has a default functionality of importing methods.

#!/usr/bin/perl

package Arithmetic;

require Exporter;

@ISA= qw(Exporter); # This is basically for implementing inheritance.

@EXPORT = qw(add);

@EXPORT\_OK = qw(subtract);

sub add

{

my $a=$\_[0];

my $b=$\_[1];

return ($a+$b);

}

sub subtract

{

my $a=$\_[0];

my $b=$\_[1];

return ($a-$b);

}

1;

@EXPORT array can be used to pass a list of variables and subroutines which by default will be exported to the caller of the Module.

@EXPORT\_OK array can be used to pass a list of variables and subroutines which will be exported on demand basis, where the user has to specify while loading the module.

#!/usr/bin/perl

use

Arithmetic qw(subtract);

print add(5,6);

print subtract (5,6);

By default, add subroutine will be exported. Subtract method won't be exported if it not specified while loading the module.

**Object Oriented Programming in Perl**

In this section, we will learn how to create Perl Object oriented Modules. First, let's see what is the object? An object is an instance using which we can access, modify and locate some data in any Perl module. This is nothing but making your existing Perl package, variables and subroutines to act like class, objects, and methods in reference to other Programming Languages.

**Create Class**

We already know how to create modules from the previous topic. The purpose of the class is to store methods and variables. A Perl Module will have subroutines which are methods. We need to access those variables and subroutines objects.

**Perl Constructor**

A constructor in Perl is a method which will execute and return us a reference with the module name tagged to the reference. This is called as blessing the class. We use a particular variable for blessing a perl class, which is bless.

#!/usr/bin/perl

package Arithmetic;

sub new

{

my $class=shift;

my $self={};

bless $self, $class;

return $self;

}

sub add

{

my $self= shift;

my $a=$\_[0];

my $b=$\_[1];

return ($a+$b);

}

sub subtract

{

my $self= shift;

my $a=$\_[0];

my $b=$\_[1];

return ($a-$b);

}

1;

The new method used as a constructor for a class, This constructor will create an object for us and will return to the script which is calling this constructor.

#!/usr/bin/perl

use Arithmetic;

my $obj= Arithmetic->new();

my $result= $obj->add(5,6);

print "$result";

$result = $obj->subtract(6,5);

print "$result";

Here, we need to understand how the object created. Whenever we try to create an object for the class, we need to use the full name of the class. Suppose, if the perl class is located in some lib\Math\Arithmetic.pm. And, if we want to access this perl class from the lib directory then we have to provide the entire path of to the class while calling in the script.

**use lib::Math::Arithmetic;**

my $obj = lib::Math::Arithmetic->new();

This is how the object creation in Perl happens.

**@INC:**

How does Perl script know where library module exists? Perl only knows about current directory of the script and the Perl inbuilt library path. Whenever we use, and Perl module, which is not located in the current directory or Perl library Path, the script will always fail. About @INC, this is an array, which holds all directory paths where it has to look for the Perl modules. Try to execute this command and see what will be the output.

perl –e "print @INC"

This will give some output, and that is the path where the lib Modules will be available. Whenever we use any new library module, we need to tell Perl, interpreter, to look into that particular location where Perl module is available.

push(@INC, "PATH TO YOUR MODULE");

Make this as your first line of code. This will tell your interpreter to look into that Path. or use

lib Arithmetic; # List here is your Perl Module location