Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Perl

Lesson Objectives

- After completing this lesson you should be able to understand
  - Introduction to diff type of applications
  - Introduction to Perl language and its features
  - Variables and data types
  - Decision making constructs and loops
  - Some functions

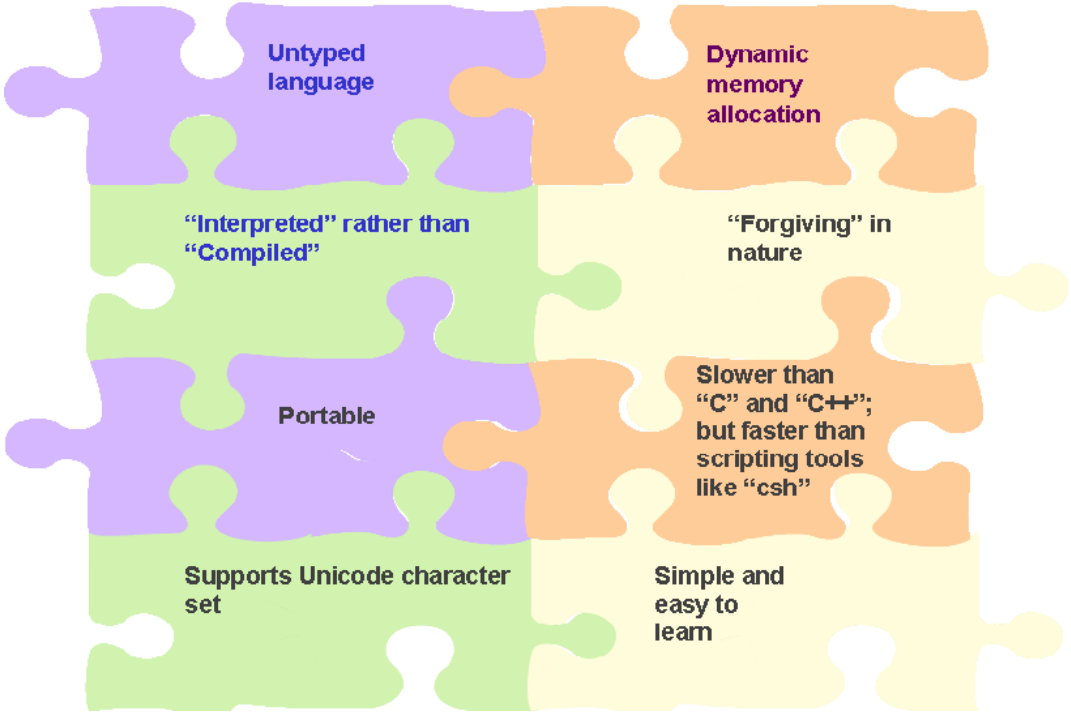
Perl –What is ?

- Perl is a stable, cross platform programming language
- Though Perl is not officially an acronym but few people used it as Practical Extraction and Report Language.
- Perl was created by Larry Wall
- Its first release was in December 1987

Perl - Purpose

- System Administration
- Text Processing
- Creating dynamic web sites
- Managing and configuring network

Perl features



Perl Installation

- Perl is available on variety of platforms
  - Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX etc.)
  - Win 9x/NT/2000/
  - WinCE
  - Macintosh (PPC, 68K)
  - Solaris (x86, SPARC)
  - OpenVMS
  - Alpha (7.2 and later)
  - Symbian
  - Debian GNU/kFreeBSD
  - MirOS BSD
  - And many more...

Getting Perl

- Perl Official Website – https://www.perl.org/
- Perl Documentation Website – https://perldoc.perl.org

Perl Installation for windows

- Follow the link for the Active Perl installation on Windows : https://www.activestate.com/activeperl/downloads
- Download either 32bit or 64bit version of installation.
- Run the downloaded file by double-clicking it in Windows Explorer. This brings up the Perl install wizard, which is really easy to use. Just accept the default settings, wait until the installation is finished, and you're ready to roll!

Perl –Eclipse plugin

- Download eclipse plug in from the url: http://www.epic-ide.org/download.php
- Add this to Eclipse by selecting Help-Install new software
- Select the archive option
- Restart Eclipse
- Create a first project in Eclipse

First Program - Perl

- Open Eclipse
- Select File-New-Other
- Type perl and select Perl Project
- Type the project name : ProjectTraining
- Inside this project create new perl file
- Right click on the Project name – select new file – perl file- give the file name : main.pl and type the following code

```
use strict;
use warnings;
sub main{
        print "Hello Perl!!!";
}
main();
```

Escape Sequences

- Escape characters are used to perform text formatting.
- These characters are preceded by a backslash(\).
- Some of the escape characters as follows:
  - \n – New Line
  - \t – Tab
  - \” – Double quotation
  - \’ – Single Quotation
  - \\ - Backslash
  - \0 – Octal characters
  - \x –Hexadecimal characters

Single v/s double quotes

- Escape sequence are not understood when you put the statement inside single quotes, however they are interpreted well when we put the statement inside double quoutes

- print ‘\nhello’; # will print the output as \nhello
- print “\nhello” # will print the output hello on the next line

Variables

- Temporary location in the memory to hold value of your application
- Three types of variables allowed in Perl
  - Scalar variable: $: to hold single values
  - Array variable: @: to hold multiple values
  - Associative Array variables (Hashes): %: to hold key value combination
- Variable names are case sensitive and they should
  - Specify the type of variable(scalar, array, hash)
  - Begin with alphabet or an underscore

Scalar Variables

- Scalar Variable is the name for a data space in memory.
- It is represented by a variable name preceded by $.
- Each scalar variable holds a single value.
- The scalar variables are untyped variables.
- Their values can be numeric, string, undefined or reference.
- Assignment operator “=“ is used to assign a literal value.
- Type of data is determined in the context of uses of the variables.
- These are global variables by default.

Example of Variable declaration

```
$age = 25;              # An integer assignment
$name = "John Paul";    # A string
$salary = 1445.50;      # A floating point
```

Variables Scope

- Scope of a variable refers to the visibility of the variable in the code.
- By default, variables are global in PERL.
- Variables are categorized into two types:
  - Lexical Variables (confined to the block in which they are defined)
  - Dynamic Variables

Lexical Variables

- Confined to the block in which they are declared.
- Declared using the my keyword.
  - For example,

    my $x=10;
- Stored in a scratch patch (private symbol table and not package's symbol table), when they are created instead of symbol table.
- Erased from memory as soon as they go beyond the scope of the block.

Dynamic Variables

- Belong to the package in which they are declared
- Global in nature
- Declared using the our or local keyword
  - For example,

    our $x=10;
- Stored in symbol table
- Accessed using the name of the package in which they are defined

Variables - Demo

```
use strict;
use warnings;

sub global{
    our $global=100;
    print ("\nGlobal = $global");
}

sub main(){
    my $name='pravin';
    my $age=10;

    my $status=($age>18?'Eligible for voting':'not eligible');
    print("\nstatus is $status");
}
main();

global();
print "\n$main::global";
```

Control structures

.No.	Statement & Description
1.	if statementAn **if statement** consists of a boolean expression followed by one or more statements.
2.	if...else statementAn **if statement** can be followed by an optional **else statement**.
3.	if...elsif...else statementAn **if statement** can be followed by an optional **elsif statement**and then by an optional **else statement**.
4.	unless statementAn **unless statement** consists of a boolean expression followed by one or more statements.
5.	unless...else statementAn **unless statement** can be followed by an optional **else statement**.
6.	unless...elsif..else statementAn **unless statement** can be followed by an optional **elsif statement** and then by an optional **else statement**.
7.	switch statementWith the latest versions of Perl, you can make use of the **switch**statement. which allows a simple way of comparing a variable value against various conditions.

If-else

- Syntax

```
if (expr_1) {
      statement_block_1
}
 elsif (expr_2) {
      statement_block_2
 }
else {
     statement_block_4
     ...
}
```

If-else

- Code snippet

```
$x=30;
if($x > 4)
{
      print "Number is greater than 4";
}
elsif ($x < 4)
{
      print "Number is less than 4";
}
else
{
      print "Numbers are Equal";
}
```

unless

- Syntax

```
unless (expression)
{
        statement_block 1;
}
elsif (expression)
{
        statement_block 2;
}
else
{
        statement_block 3;
}
```

unless

- Code snippet

```
$x=30;
unless($x > 4)
{
      print "Number is greater than 4";
}
elsif ($x < 4)
{
      print "Number is less than 4";
}
else
{
      print "Numbers are Equal";
}
```

Ternary Operator ?:

```
use strict;
use warnings;

sub main(){
        my $name='pravin';
        my $age=10;
        my $status=($age>18?'Eligible for voting':'not eligible');
        print("\nstatus is $status"); #will print not eligible
}
main();
```

Loops

S.No.	Loop Type & Description
1.	**while loop**Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2.	**until loop**Repeats a statement or group of statements until a given condition becomes true. It tests the condition before executing the loop body.
3.	**for loop**Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
4.	**foreach loop**The foreach loop iterates over a normal list value and sets the variable VAR to be each element of the list in turn.
5.	**do...while loop**Like a while statement, except that it tests the condition at the end of the loop body
6.	**nested loops**You can use one or more loop inside any another while, for or do..while loop.

Loops

- while
- do-while
- until
- do-until
- for
- foreach

Loops –while

```
$count = 1;
print (“\n Numbers from 1 to 5 \n");
while ($count <= 5)
{
          print  $count. "\n";
         $count = $count + 1;
 }
 print ("End of loop.\n");
```

```
$count = 1;
print (“\n Numbers from 1 to 5");
do
{
          print  $count. "\n";
         $count = $count + 1;
 } while ($count <= 5) ;

 print ("End of loop.\n");
```

Loops - until

```
$count = 1;
print (“\n Numbers from 1 to 5\n");
until ($count > 5)
{
        print  $count. "\n";
        $count = $count + 1;
 }
 print ("End of loop.\n");
```

```
$count = 1;
print ("\n Numbers from 1 to 5\n");
do
{
        print  $count. "\n";
        $count = $count + 1;
 } until ($count > 5);
 print ("End of loop.\n");
```

Loop - for

```
print (“\n Numbers from 1 to 5\n");
for (my $count=1;$count<=5;$count++)
{
        print  $count. "\n";
 }
 print ("End of loop.\n");
```

```
print (“\n Numbers from 1 to 5 \n");
foreach $x (1..5)
{
        print  $x. "\n";
 }
 print ("End of loop.\n");
```

Other control structures

- PERL supports some other control structures also which can be used in conjunction with the basic flow structures to change the flow of control
- They are as follows:
  - continue
  - next
  - last
  - redo
  - goto

Continue block

- Continue block is normally attached to a block (while, until or foreach).
- It is executed before the condition is evaluated for the next iteration.
- It is used in situations, where the code is to be executed after each iteration of loop.

Continue block

```
continue
{
        statement_block
 }
```

```
$count = 1;
print (“\n Numbers from 1 to 5");
while($count<=5)
{
        print $count."\n";
}
continue
{
        $count++;
}
 print ("End of loop.\n");
```

Next

- The next alters the flow of execution within the loop body.
- It is also known as a loop modifier. Other modifiers are last and redo.
- It skips the rest of processing of body to go forward with the next iteration of loop.
- It executes the continue block, if present before the start of the next iteration.

next

next [LABEL];

```
$count=1;
print "Odd numbers \n";
while($count<=5)
{
        if ($count%2==0)
        {       next;     }
        else
        {       print $count."\n"; }
}
continue
{       $count++;   }
print "\n End of Loop";
```

last

- The last modifier skips the rest of processing of the body to exit the loop.
- The control is transferred beyond the last iteration of the loop.
- It doesn’t execute the code in the continue block.

last

last [LABEL];

```
$count = 0;
print ("\nNumbers from 1 to 5");
while ($count<=10)
{
$count++;
print "\n$count";
last if $count==5;
}
 print ("\nEnd of loop.");
```

Redo

- This modifier restarts with the current iteration of the loop.
- Loop condition is not re-evaluated.
- It retains the current value of the loop iterator.
- Continue block is not executed.

Redo

redo;

```
$count=1;
while ($count<=10)
{
        $count++;
        print"$count\n";
        if ($count==5)
        {
                print "$count \n";
                redo;
        }
}
```

goto

- The goto modifier is used to jump over iterations or statement thereby altering the normal flow of control
- As soon as goto is encountered, control is transferred over to the LABEL or expression that evaluates to a LABEL or to the subroutine being referred to

goto

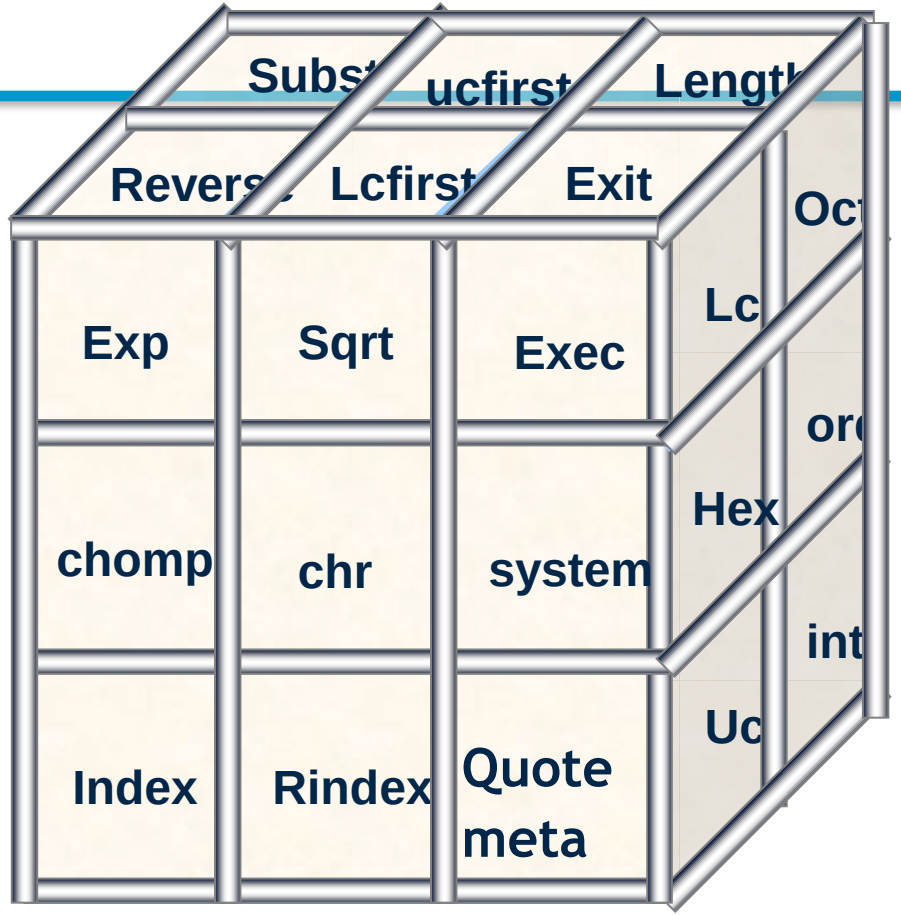
- Syntax

```
goto LABEL
Or
goto EXPR
Or
goto &NAME
```

```
print "\n Hi";
goto OUTER;

print "\n So let's part";
goto INNER;
INNER: print "Good Bye";
OUTER: print “ Hello";
```

Built in Functions



Abs function

- The **abs** function returns the absolute value of the given value.
- If no argument is passed, it returns the absolute of the value contained in $_.

For example,

```
print abs(-3);
#prints the output as 3.
```

Sqrt function

- The **sqrt** function returns the square root of the given value.
- It gives an error, if the specified value is a non-numeric value or expression, or if the value evaluates to a negative number.

For example,

```
print sqrt(4);
#prints the value as 2.
```

Exp and Int function

- The **exp** function returns the value of e to the power of the given value.

  For example,

  ```
  print exp(-3);
  ```

  #prints the output as e-3. i.e 0.498706

- The **int** function returns the integer part of the given value.

  For example,

  ```
  print int(-3.9);
  ```

  #prints the output as -3.

Chomp function

- The **chomp** function returns the value after removing the new line-character.

  For example,

```
my $str=“Hello World \n”;
chomp( $str);
print $str;
#prints the value Hello World but does not take the cursor to the next line.
```

Chr function

- The **chr** function returns the character corresponding to a specific ASCII code.

  For example,

```
my $asc=65;
print chr($asc);
#prints A.
```

Ord function

- The **ord** function returns the ASCII code for a particular character.

  For example,

```
my $chr=“A”;
print ord($chr);
#prints 65.
```

Length lc and uc function

- The **length** function returns the number of characters in the given string.

  For example,

  ```
  print length(“hello”);
  #prints the value as 5.
  ```

- The **lc** function converts all the characters of the given string to lowercase.

  For example,

  ```
  print lc(“Hello World”);
  #prints the value as hello world.
  ```

- The **uc** function converts all the characters of the given string to uppercase.

  For example,

  ```
  print uc(“Hello World”);
  #prints the value as HELLO WORLD.
  ```

Lc and Uc function

- **The Lcfirst Function:**
  - The **lcfirst** function converts the first character of the given string to lowercase.

    For example,

    ```
    print lcfirst(“Capgemini”);
    #prints the value as capgemini.
    ```

- **The Ucfirst Function:**
  - The **ucfirst** function converts the first character of the given string to uppercase.

    For example,

    ```
    print ucfirst(“world”);
    #prints the value as World.
    ```

Reverse function

- The **reverse** function reverses the characters in the given string or array or hash.
- It does not make changes in the actual data structure.

For example,

```
$str=reverse("pravin");
 print "\n",$str;
 #prints the value nivarp.
print "\n".reverse("A B C D");
 #prints the value D C B A.
print "\n".reverse("101 102 103");
 #prints the value as 301 201 101.
```

Substr function

- The **substr** function returns a part of the given string.
- Syntax:

```
substr($str, $offset, $length)
```

- String is the string for which the subpart is to be fetched.
- Offset is the position from which the values should be fetched.
- Length is the number of characters to be fetched from the specified offset.

Summary

- In this lesson you learnt
  - Introduction to diff type of applications
  - Perl language and its features
  - Variables and data types
  - Decision making constructs and loops
  - Some functions