Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Perl

Lesson Objectives

- After completing this lesson you should be able to understand
  - Subroutines
  - Regular Expressions
  - File Handling in Perl

Subroutine

subroutine

- Subroutine is a name given to a section of code
- It is similar to a user-defined function in C
- It is mainly created to:
  - Reuse code
  - Manage code
- It can be placed anywhere in the program (at the beginning or the end)

subroutine

- There are three sections in the declaration of the subroutine
  - The sub keyword
  - Name of the subroutine
  - Block of code
- The @_ list array variable is the special variable in PERL that gets created for every subroutine and holds the arguments passed to the subroutine. Thus the first argument to the function is in $_[0], the second is in $_[1], and so on.

subroutine

- syntax

```
sub subname
{
        code block
}
```

subroutine

- Code snippet

```
sub fun
{
        print “Hello World";
}
fun;
```

```
sub greet{
        @names = @_;
         foreach my $name (@names)
        {
                print “Hello , $name!\n";
        }
}
print greet( “john", “Harry", “Maggie");
```

Reference to a subroutine

```
sub displayhash{
   my (%hash) = @_;

   foreach my $item (keys %hash){
      print "Item : $item\n";
   }
}

my %hash = ('name' => ‘Rahul', 'age' => 29);

# Create a reference to above function.
my $cref = \& displayhash;

# Function call using reference.
&$cref(%hash);
```

Returning value from a subroutine

```
sub addnumbers{
	my @numbers=@_;
	my $n=scalar(@_); #get total number of arguments passed
	my $sum=0;
	foreach my $num (@numbers){
		$sum=$sum+$num;
	}
	print ("\n Sum of numbers is $sum");
	my $average=$sum/$n;
	return $average;
}

my $avg1=addnumbers(1,2,3);
print "\nAverage is $avg1";
```

local variables with subroutine

- It is used when current value of a variable must be visible to called subroutine
- It is called as dynamic scoping in Perl

local variables with subroutine

```
# Global variable
our $string = "Hello, World!";

sub PrintHello{
   # Private variable for PrintHello function
   local $string;
   $string = "Hello, Perl!";
   PrintMe();
   print "Inside the function PrintHello $string\n";
}
sub PrintMe{
   print "Inside the function PrintMe $string\n";
}
# Function call
PrintHello();
print "Outside the function $string\n";
```

state variables with subroutine

- They are similar to private variables but they maintain their state and they do not get reinitialized upon multiple calls of the subroutines.
- These variables are defined using the stateoperator and available starting from Perl 5.9.4

state variables with subroutine

```
use feature 'state';
#use 5.010;

sub PrintCount{
   state $count = 0; # initial value

   print "Value of counter is $count\n";
   $count++;
}

for (1..5){
   PrintCount();
}
```

Regular Expressions

Introduction to Regular Expressions

- Regular Expression is a string used to describe or match a string as per the specified expression or pattern.
- A regular expression is made up of many parts:
  - Modifiers
  - Character classes
  - Alternative match patterns
  - Quantifiers
- The =~ operator is used to test the match and !~ is used to negate the match.

Regular Expressions - Modifiers

- Modifiers are used to match or replace a pattern.
- They are as follows:
  - **m*//*** : Matches a pattern
  - **s *///*** : Substitutes the pattern matched with a string.

Regular Expressions - Modifiers (Contd…)

- **m//**

  Used to match the specified pattern

  Pattern match is case sensitive

  Returns TRUE/FALSE

  Syntax:

```
m/pattern/
```

Regular Expressions - Modifiers (Contd…)

- Code Snippet

```
$str = “How are you";
if ($str =~ m/are/)
{
        print "Match found";
}
```

Regular Expressions - Modifiers (Contd…)

- **s///:**
  - Used to match the specified pattern and substitute it with another string.
  - Pattern match is case sensitive.
  - Returns TRUE/FALSE
  - Syntax:

```
s/pattern_to_search/pattern_to_be_replaced/
```

Regular Expressions - Modifiers (Contd…)

- Code Snippet

```
$str = “How are you ?";
if ($str =~ s/you/they/)
{
        print “\n $str";
}
```

Regular Expressions - Characters

- Some of the special characters used with regular expressions are as follows:
  - **\D** : Non digit character
  - **\d** : Digit character
  - **\S** : Non white-space character
  - **\s** : White-space character
  - **\W** : Non word-character
  - **\w** : word-character (alphanumeric as well _)

Regular Expressions - Characters (Contd…)

- Code Snippet

```
$str = "Patni Computer Systems - 13";
if ($str =~ m/\D/) {
        print "\n Found Non-digit Character";
}else{
        print "\n Found Digit Character";
}
$str = "25345";
if ($str =~ m/\D/) {
        print "\n Found Non-digit Character";
}else{
        print "\n Found Digit Character";
}
```

Regular Expressions - Characters (Contd…)

- Code Snippet

```
$str = "Patni Computer Systems - 13";
if ($str =~ m/\d/) {
            print "\n Found Digit Character";
}else{
             print "\n Found Non-digit Character";
}
$str = "25345";
if ($str =~ m/\d/) {
             print "\n Found Digit Character";
}else{
             print "\n Found Non-digit Character"; }
```

Regular Expressions - Characters (Contd…)

- Code Snippet

```
$str = "Patni Computer Systems - 13";
if ($str =~ m/\s/) {
        print "\nFound White space Character";
}else{
        print "\nFound No White space Character";
}
$str = "25345";
if ($str =~ m/\s/) {
        print "\nFound White space Character";
}else{
        print "\nFound No White space Character";
}
```

Regular Expressions - Characters (Contd…)

- Code Snippet

```
$str = "Patni Computer Systems - 13";
if ($str =~ m/\w/) {
        print "\nFound Word";
}else{
        print "\nFound Special Characters";
}
$str = "**\\!";
if ($str =~ m/\w/) {
        print "\nFound Word";
}else{
        print "\nFound Special Characters";
}
```

Regular Expressions – Character class

- Characters can be grouped into character class and the class matches one character inside it
- Character class is represented using []
- You can also specify the range of characters within character classes using -

Regular Expressions – Character class

- Code snippet

```
$str = "Patni Computer Systems _ 13";
if ($str =~ m/[aeiou]/) {
        print "There are vowels";
} else {
        print "\n There are no vowels";
}
$str = "Hw W'll";
if ($str =~ m/[aeiou]/) {
        print "There are vowels";
} else {
        print "\n There are no vowels";
}
```

Regular Expressions – Character class

- Code snippet

```
$str = "Patni Computer Systems _ 13";
if ($str =~ m/[0-9]/)
{
        print "The string contains numerals";
}
else
{
        print "\n There are no numerals within the string";
}
```

Regular Expressions – Alternative Match Patterns

- We can also search for more than one alternate possibilities
- Character used to search for alternatives is |

Regular Expressions – Alternative Match Patterns

- Code Snippet

```
$str = "Patni Computer Systems _ 13";
if ($str =~ m/(Patni|patni|PATNI)/)
{
        print "The string contains the word patni";
}
else
{
        print "\n The string does not contain the word patni";
}
```

Regular Expressions – Quantifiers

- Quantifiers are used to specify that a pattern should be repeated a specific number of times
- The quantifiers are as follows:
  - ***** : Matches zero or more of the preceding character
  - **.***: Matches zero or more of any character, as many as possible
  - **.***?:Matches zero or more of any character, as few as possible
  - **+** : one or more of the preceding, as many as possible
  - **?** : one or zero times
  - **{n}** : n times exactly
  - **{n,}** :at least n times
  - **{n, m}**: at least n times and at the most m times.

Regular Expressions – Quantifiers

- Code Snippet

```
$_ =  "PACE Of Patni Computers is located in
Airoli, Mumbai, Maharashtra,India.";

if ( /Of (.*),/ )
{
        print "$1\n";
}
```

Regular Expressions – Quantifiers

- Code Snippet

```
$str = "The programming republic of Perl";

$str =~ /(m{1,3})(.*)/;
 # matches, $1 = 'mm‘  $2 = 'ing republic of Perl'

print "\n $1 \n $2 ";
```

Regular Expressions-Assignment

- Write a program to validate the below mentioned email ids. Iterate through each mail and use regular expression for this assignment

```
my @emails = (
                'john@caveofprogramming.com',
                'hello',
                '@llkj.com',
                'jklj778dd@somewhere77.com',
                'lkjl@7788.',
        );
```

File Handling

File Handling

- The open function is used to open a file.
- The function creates an input or output channel depending on the mode in which the file has been opened.
- It returns TRUE if successful and undefined otherwise.
- It mainly takes three arguments:
  - First Argument: **FILEHANDLE**
  - Second Argument: Specifies mode in which the file is to be opened
  - Third Argument: Specifies list of files to be opened

Opening File

- Syntax

```
open FILEHANDLE ,MODE ,names.
```

Opening File

- The Different modes of opening a file are as follows:
  - **<**: Read Mode
  - **>**: Write Mode
  - **>>**: Append Mode
  - **+>** or **+<**: Read and Write Mode

Reading from File

- The angle operator <> is used for reading from a file
- The operator returns the next line of input from the file
- Syntax

```
< FILEHANDLE>
```

- If FILEHANDLE is omitted, it reads from STDIN

Reading from File

- By default the file would be opened in the read mode

```
sub main{

        my $file='D:\Presentation\Perl\MyWorkSpace\Tutorial3\file1.txt';
        open(INPUT, $file) ;
        while(<INPUT>){
            print “$_”;
          }
        close(INPUT);
}
main();
```

Reading from File

- The getc function reads character by character
- It returns the character read or undefined if end of the file has been reached
- Syntax
  - getc HANDLE

Reading from File

- By default the file would be opened in the read mode

```
sub main{
        my $file='D:\Presentation\Perl\MyWorkSpace\tutpoint\file1.txt';
        open(INPUT, $file) or die "Input file $file not found\n";

        while($char=getc INPUT ){
                print "$char";
        }
        close(INPUT);
}
```

Writing to File

- The Print function is used for writing to a file
- It returns TRUE if the write operation is successful
- FILEHANDLE has to be specified to print or write into the file
- If FILEHANDLE is not specified, it will be written to STDOUT
- Syntax
  - Print FILEHANDLE LIST

Writing to File

```
sub main{
        my $output="outputfile.txt";
        open(OUTPUT,'>'.$output) or die "cant create $output \n";
        print OUTPUT "Hello world\n";
        close(OUTPUT);
}

main();
```

Assignment

- Write a program to read data from file1.txt and write it to file2.txt Replace the word “has” in file1 and write the word “is” to file2

Assignment

```
sub main{

        my $input='D:\Presentation\Perl\MyWorkSpace\tutpoint\file1.txt';
        open(INPUT, $input) or die "Input file $input not found\n";

        my $output="outputfile.txt";
        open(OUTPUT,'>'.$output) or die "cant create $output \n";

        while(my $line=<INPUT>){
                $line=~ s/has/is/i; # to find word "has"and replace it with word "is"
                #$line=~ s/has/is/ig; # to find word "has"and replace all occurenece with word "is"
                Print OUTPUT $line;

        }
close(INPUT);
close(OUTPUT);
}
main();
```

Assignment

- Write a program to read data from file1.txt and search for lines which had word starting with letter ‘h’ and ending with letter ‘s’

Assignment

```
use strict;
use warnings;
 sub main{
		my $file='D:\Presentation\Perl\MyWorkSpace\Tutorial3\file1.txt';
		open(INPUT, $file) or die "Input file $file not found\n";
		while(my $line=<INPUT>){
			if($line =~ / h.s /){
				print $line;
			}

		}
		close(INPUT);
}

main();
```

Assignment

- Write a program to read data from file1.txt and search for lines using groups. Group 1: lines starting with letter ‘I’ followed by any two letter followed by letter ‘a’ followed by any single letter. Group2:Any three letter followed by group1

- **File1.txt**

I had too use this

I said that I want to use

I took her to the city

If any of you have to come then you are welcome

I was standing near the hotel

Assignment

```
use strict;
use warnings;
sub main{

        my $file='D:\Presentation\Perl\MyWorkSpace\Tutorial3\file1.txt';
        open(INPUT, $file) or die "Input file $file not found\n";

        while(my $line=<INPUT>){
                if($line =~ /(I..a.)(...)/) { # groups with reg ex using parenthisis
                        print "First match: '$1'; second match:'$2'\n";
                }

        }
        close(INPUT);
}
main();
```

Assignment

Suppose we have a string containing the following value.

```
my $text = 'I am 117 years old tomorrow.';
```

Write a regex pattern to display only digit from the text

Assignment

```
my $text = 'I am 117 years old tomorrow.';

if($text =~ /(\d+)/) {
        print "Matched: '$1'\n";
}
```

Assignment

Write a code to pick up the machine code from the text given below

my $text = "The code for this device is GP8765.“

Assignment

```
use strict;
use warnings;

sub main{

        my $text = "The code for this device is GP8765.";

        if($text =~ /(\w\w\d{2,6})\./){
                print "The code is : $1\n";
        }else{
                print "Not found\n";
        }
}

main();
```

File Handling Functions

- There are some in-built functions provided by PERL to handle files.
  - **copy:** copies one file to another
  - **move:** moves the **FILEHANDLE** function
  - **rename:** renames the file
  - **unlink:** deletes the file
  - **seek:** moves the **FILEHANDLE** function to a particular position
  - **tell:** returns the current position of **FILEHANDLE**

File Handling Functions

- The copy function copies one file to another
- To use the copy command use the module: use File::Copy
- It takes two parameters
  - File to be copied
  - Name of the copy file to be created
- Syntax

```
copy($filetobecopied, $newfile)
```

File Handling Functions

- The rename function renames the file.
- It returns TRUE, if the file has been renamed successfully, and FALSE otherwise.
- Syntax

```
rename OLDFILE , NEWFILE
```

File Handling Functions

- The unlink function deletes the file.
- It returns TRUE, if the file has been deleted successfully.
- Syntax

```
unlink (filename)
```

File Handling Functions

- The tell function returns the current position of FILEHANDLE.
- Syntax

```
tell FILEHANDLE
```

File Handling Functions

- The seek function moves the FILEHANDLE function to a particular position.
- It takes three parameters:
  - First Parameter: **FILEHANDLE**
  - Second Parameter: The byte position to which the **FILEHANDLE** must move to
  - Third Parameter: Options regarding the position (can be 0,1,2)
- Syntax

```
seek FILEHANDLE, POSITION, OPTION;
```

File Handling Functions

- Code Snippet

```
open (HANDLE, "< test.txt") or die "oops: $!";
seek HANDLE, 10, 0;
print tell HANDLE;
close(HANDLE);
```

File Handling Functions

```
use warnings;


sub main(){
		my $file='D:\Presentation\Perl\MyWorkSpace\Tutorial3\file1.txt';
		open(INPUT, $file) or die "Input file $file not found\n";
		seek(INPUT,3,0);
		seek(INPUT,5,1);
		while(<INPUT>){
			print "$_"
		}
		close(INPUT)
}
main();
```

File Tests

- There are certain tests that can be performed on FILEHANDLEs to understand the behavior of certain files.
- They include the following:
  - **-r** : File or directory is readable
  - **-w** : File or directory is writable
  - **-x** : File or directory is executable
  - **-o** : File or directory is owned by user
  - **-e** : File or directory exists
  - **-z** : File exists and has zero size (directories are never empty)

File Tests

- **-s**: File/directory exists and has a nonzero size (the value is the size in bytes).
- **-d**: Entry is a directory.
- **-T**: File is "text“.
- **-B**: File is "binary“.

File Tests

- Code Snippet

```
$neededfile=“trial.pl";
if (-e $neededfile)
{
        print(“File Does Exist");
}
else
{
      print ("File Does not exist");
}
```

Working with csv files

- How to read data from csv file
- Join the csv file data with a pipe symbol

Summary

- In this lesson you learnt
  - Subroutines
  - Regular Expressions
  - File Handling